Profit Maximization for Service Placement and Request Assignment in Edge Computing via Deep Reinforcement Learning

Yuchen Li
The Australian National University
Canberra, Australia

Weifa Liang
City University of Hong Kong
Hong Kong, P. R. China

Jing Li
The Australian National University
Canberra, Australia

ABSTRACT

With the integration of Mobile Edge Computing (MEC) and Network Function Virtualization (NFV), service providers are able to provide low-latency services to mobile users for profit. In this paper, we study the problem of service instance placement and request assignment in an MEC network for a given monitoring period, where service requests arrive into the system without the knowledge of future arrivals. Each incoming request requires a specific service with a maximum tolerable service delay requirement. The problem is to maximize the profit of the service provider by admitting service requests for the monitoring period, which can be achieved by pre-installing service instances into cloudlets to shorten service delays, and accommodating new services by removing some idle service instances from cloudlets due to limited computing resources. We then devise an efficient deep-reinforcement-learning-based algorithm for this dynamic online service instance placement problem. We finally evaluate the performance of the proposed algorithm by conducting experiments through simulations. Simulation results demonstrate that the proposed algorithm is promising.

CCS CONCEPTS

• **Networks** → **Network services**.

KEYWORDS

Mobile edge-cloud networks, service request provisioning, service instance placement, profit maximization.

ACM Reference Format:
Yuchen Li, Weifa Liang, and Jing Li. 2021. Profit Maximization for Service Placement and Request Assignment in Edge Computing via Deep Reinforcement Learning. In *Proceedings of the 24th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM ’21), November 22–26, 2021, Alicante, Spain.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3479239.3485673

1 INTRODUCTION

Mobile edge computing (MEC) has been envisioned as a promising solution to provide adequate computing resources with high Quality of Service (QoS) through deploying cloudlets (edge servers) within the proximity of mobile users [7]. Combining MEC with virtualization techniques such as Network Function Virtualization (NFV), service providers now can rent cloudlet resources from infrastructure service providers to provide services for mobile users by implementing their services as VM instances in cloudlets for profits. Usually, when a user requests a service, its service provider will admit the request and initialize a service instance for the user. Once the requested service finishes, the service instance will be released back to the system to reduce the operational cost of the service provider and available resources can be utilized by future service requests. In contrast, we consider that the service providers can pre-install some service instances before the arrivals of service requests, since the locations and arriving rates of different service requests usually follow certain spatial or temporal patterns. The pre-installed service instances can be allocated to users without service initialization delays, which further shortens end-to-end service delays. Consequently, this will bring more profits by admitting more user requests while satisfying their end-to-end delay requirements.

The challenge thus lies in the fact that we can only install limited numbers of service instances at a cloudlet. It is challenging to decide how many instances of a specific service should be installed at cloudlets in order to maximize the profit of the service provider, and there are several critical questions to be addressed. For example, when should we remove idle VNF instances? the removal of an idle VNF instance may reduce the operational cost immediately, but if newly arrived requests demand the removed instances, they may not be admitted due to the violation of user QoS requirements. Also, it causes an initialization cost if we frequently install and remove service instances. Another challenge is how to predict future arrivals of user service requests. An inaccurate prediction may incur a large initialization cost.

Several studies on service provisioning and Virtual Network Function (VNF) instance placement in MEC environments have been conducted. For example, Xu *et al.* [12] jointly considered the placement of VNF instances for data processing and data traffic routing path planning for user requests in a multi-tier edge cloud network to maximize network throughput. Cziva *et al.* [2] studied the dynamic VNF placement problem to minimize the end-to-end latency of users, considering the network dynamics, user resource demands and user mobility. Li *et al* [6] considered a VNF provisioning problem to maximize the revenue by admitting user requests with service reliability requirements. They proposed two online algorithms with provable competitive ratios for the problem by adopting the primal-dual technique. He *et al.* [3] focused on the service placement and request scheduling while considering sharable and non-shareable resources in an MEC network with the aim to maximize the number of requests admitted. Xu *et al.* [11] considered the assignment of requests to shared VNF instances to maximize the network throughput while minimizing the operational cost. They proposed a prediction mechanism to dynamically adjust the number of VNF instances needed in cloudlets, which later will serve as a baseline algorithm in our experimental section.

Unlike the aforementioned studies that only determine the service placement when service requests arrive or have the knowledge of future arrivals, The novelties of this paper lie in that we focus on placing service instances prior to request arrivals, which is much more difficult as there is no way to know when and how many requests will arrive in advance.

The main contributions of this paper are given as follows. We first formulate the profit maximization problem of delay-sensitive service provisioning in MEC networks, by assigning service requests to cloudlets and pre-installing or removing service instances in cloudlets dynamically, through analyzing historical user service request patterns, subject to computing capacity on each cloudlet. We then develop an efficient algorithm for the problem, which utilizes a deep-reinforcement-learning-based prediction mechanism that is formulated as a Multi-agent Markov decision process to predict which types of services arrivals in the future, and pre-install service instances accordingly.

2 PRELIMINARIES

In this section, we introduce the system model and define the problem precisely.

2.1 System model

We consider a mobile edge computing network $\mathcal{G} = (\mathcal{V} \cup S, E)$ that consists of a set $\mathcal{V}$ of access points (APs) and a set E of links between the access points. For each link $e = (i, j) \in E$, there is a transmission delay $l_e = l_{(i,j)}$ on it. Also, a set of cloudlets S is co-located with some of the APs. For each cloudlet $c_j \in S$, it has limited computing capacity CAP_j for hosting service instances. Let $f_k \in F$ with $k = 1, 2, \ldots, |F|$ be a service function in the service set F provided by the MEC network. We assume that a service instance of f_k can only serve one user's request. We denote by $c(f_k)$ the computation requirement of the service instance of f_k. We further assume that the monitoring time period T is divided into equal time slots, denoted by t the time slot index with $1 \le t \le T$, and each time slot is split into two stages: *the service placement* stage and *the request admission* stage, respectively.

Let $n^t_{j,k} \in Z^+ \cup \{0\}$ be the number of instances of service f_k at cloudlet c_j after the service placement stage at time slot t. Since cloudlets have limited computing capacity, the number of installed service instances in cloudlet c_j is constrained by $\sum_{k=1}^{|F|} n^t_{j,k} \cdot c(f_k) \le CAP_j$. In service placement stage, we can add or remove service instances from cloudlets. Denote by $\delta^t_{j,k}$ the number of instances to be installed or deleted at the service placement stage of time slot t. If $\delta^t_{j,k} > 0$ then c_j instantiates $\delta^t_{j,k}$ instances of f_k at the service placement stage. If $\delta^t_{j,k} < 0$, $|\delta^t_{j,k}|$ instances of f_k will be released from c_j. During the service placement stage, we need to consider that some instances are serving unfinished tasks. These instances should not be revoked. The number of instances at each cloudlet after the service placement stage should be larger than the number of unfinished requests at the cloudlet. For convenience, we use $\phi^t_{j,k}$ to denote the number of requests that are still being processed but arrive before time slot t, and we have $\phi^t_{j,k} \le n^t_{j,k}$.

Mobile users can offload their tasks to cloudlets via their nearby APs. Let R be the set of service requests arrived at different APs during the monitoring period of T. We assume that at time slot t, a subset of requests $R(t)$ ($\subset R$) arrives from APs, where request $r_i \in R$ can be described by a tuple $(t_i, f_{k_i}, v_i, \tau_i, d_i)$, where t_i is the time slot of its arrival; $f_{k_i} \in F$ represents its service type, $v_i \in \mathcal{V}$ represents its closest AP to the device, τ_i represents its execution duration, and d_i is its end-to-end delay requirement.

At the arrival of request r_i, we use a binary variable $x^t_{i,j} \in \{0, 1\}$ to denote whether request r_i is admitted by cloudlet c_j or not. Request r_i can only be admitted by one cloudlet at the time slot of its arrival. We have $\sum_{t=1}^{T} \sum_{j=1}^{|S|} x^t_{i,j} \le 1$ and $x^t_{i,j} = 0$.

Each request has an execution duration of τ_i time slots and once a request is admitted, the request leaves the MEC system at the end of $(t_i + \tau_i - 1)$-th time slot. We use $P^t_{i,j} \in \{0, 1\}$ to denote that request i is being processed by cloudlet c_j at time slot t, and thus

$$\forall i, j, t \quad P^t_{i,j} = \begin{cases} x^{t_i}_{i,j}, & t_i \le t < t_i + \tau_i \\ 0, & otherwise \end{cases} \tag{1}$$

We use a binary variable $x^{idle}_{t,i,j} \in \{0, 1\}$ to indicate if r_i is offloaded to an idle instance, and $x^{t^{new}}_{i,j} \in \{0, 1\}$ to indicate if request r_i is served by initializing a new instance.

$$x^t_{i,j} = x^{idle}_{t,i,j} + x^{new}_{t,i,j}, \quad \forall i, j, t. \tag{2}$$

The number of requests offloaded to idle instances on c_j should be smaller than the number of idle instances on c_j, and thus

$$\sum_{i \in \{i' \mid f_{k'_i} = f_k\}} x^{idle}_{t,i,j} \le n^t_{j,k} - \phi^t_{j,k}, \quad \forall j, k, t. \tag{3}$$

Also, cloudlet c_j must have sufficient resources to initialize new service instances for admitted requests, then,

$$\sum_{k=1}^{|F|} n^t_{j,k} \cdot c(f_k) + \sum_{i \in \{i' \mid t'_i = t\}} x^{new}_{t,i,j} \cdot c(f_{k_i}) \le CAP_j, \quad \forall j, t \tag{4}$$

Recall that $\phi^t_{j,k}$ is the number of unfinished requests for service f_k at the service placement stage of time slot t, we have

$$\phi^t_{j,k} = \sum_{i \in \{i' \mid f_{k_{i'}} = f_k \ and \ t_{i'} \neq t\}} P^t_{i',j}, \quad \forall t, j, k, \tag{5}$$

The number of instances of service f_k at time slot $t + 1$ then is

$$n^{t+1}_{j,k} = n^t_{j,k} + \sum_{i \in \{i' \mid f_{k'_i} = f_k\}} x^{new}_{t,i,j} + \delta^{t+1}_{j,k}, \quad \forall t, j, k. \tag{6}$$

Assigning request r_i to a cloudlet c_j will experience the network delay and potential initialization delay. We define the network delay as the accumulative delay along the shortest path from the arrived AP to the target cloudlet. The network delay of offloading request r_i from v_i to cloudlet c_j is $L_{i,j} = \sum_{e \in p_{i,j}} l_e$, where $p_{i,j}$ is the shortest path in the MEC network between AP v_i and AP v_j. We assume that the initialization time of service f_k is ins_k. If request r_i is offloaded to cloudlet c_j while c_j does not have an idle instance for f_{k_i} at the moment, the cloudlet will instantiate a new instance for f_k. This will result in an initialization delay. The latency constraint thus is:

$$L_{i,j} \cdot x_{i,j}^t + x_{t,i,j}^{new} \cdot ins_{k_i} \le d_i, \quad \forall i, j, t. \tag{7}$$

The operational expenditure of the MEC network consists of three components: the energy consumption for hosting service instances even the instances are idle; the processing energy consumption for serving a service instance; the initialization for the creation of service instances. Let $C^{idle}(f_k)$ be the energy consumption per time slot for hosting an instance of service f_k, $C^{ser}(f_k)$ be the energy cost per time slot for processing a request of service f_k, and $C^{ins}(f_k)$ be the initialization cost for an instance of service. The operational cost $C_{c_j}(t)$ of cloudlet c_j at time slot t thus is

$$\begin{aligned} C_{c_j}(t) = & \sum_{i \in \{i' \mid t'_i = t\}} x_{t,i,j}^{new} \cdot C^{ins}(f_{k_i}) + \sum_{k=1}^{|F|} [\delta_{j,k}^t]^+ \cdot C^{ins}(f_k) \\ & + \sum_{i \in \{i' \mid t'_i = t\}} x_{t,i,j}^{new} \cdot C^{idle}(f_{k_i}) + \sum_{k=1}^{|F|} n_{j,k}^t \cdot C^{idle}(f_{k_i}) \\ & + \sum_{i \in \{i' \mid t'_i = t\}} P_{i,j}^t \cdot C^{ser}(f_{k_i}) \end{aligned} \tag{8}$$

where $[a]^+ = \max(a, 0)$.

We assume that the service provider will earn the amount $RV(f_k)$ of revenues per time slot for processing each request of service f_k. Therefore, the earned revenue by cloudlet c_j at time slot t is

$$RV_{c_j}(t) = \sum_{r_i \in R(t)} x_{i,j}^t \cdot \tau_i \cdot RV(f_{k_i}). \tag{9}$$

The profit of cloudlet c_j at time slot t is $RV_{c_j}(t) - C_{c_j}(t)$. Thus, the total profit of the service provider obtained by admitting user service requests for the monitoring period T is

$$\sum_{j=1}^{|S|} \sum_{t=1}^{T} (RV_{c_j}(t) - C_{c_j}(t)). \tag{10}$$

2.2 Problem formulation

The *online service placement and request assignment* problem is defined as follows. Given a mobile edge-cloud network $\mathcal{G} = (\mathcal{V} \cup S; E)$, a set F of services provided by the network, a finite time horizon that is divided into equal T time slots, and a sequence of offloading task requests arriving one by one without the knowledge of future arrivals, let R be the set of arrived requests within the given time horizon, where each request $r_i \in R$ demands a service f_{k_i} with a tolerable maximum delay requirement d_i. The problem is to assign the requests to different cloudlets while meeting their delay requirements through (1) pre-deploying different service instances in F to cloudlets and determining the number of each of such service instances deployed to meet service delays of admitted requests; and (ii) removing some existing idle service instances from the system to accommodate new service instances due to limited computing resources in cloudlets, such that the accumulative profit $\sum_{j=1}^{|S|} \sum_{t=1}^{T} (RV_{c_j}(t) - C_{c_j}(t))$ of admitted requests within the given time horizon is maximized, subject to the computing capacity on each cloudlet in $\mathcal{G}$.

3 AN ONLINE ALGORITHM

In this section we devise an online algorithm for the online service placement and request assignment problem.

We start with formulating the online service placement and request assignment problem as a multi-agent Markov Decision Process (MMDP) as follows.

1) Agent. Each cloudlet c_j with $j \in \{1, 2, \ldots, |S|\}$ is modeled by an agent $\mathcal{N}_j$ to manage its service placement.

2) State. The state $s \in \mathcal{S}$ describes the status of the MEC network. The system state $s^t \in \mathcal{S}$ at time slot t is defined by a tuple $(R_\eta(t), n(t), R^*(t), t)$, where $R_\eta(t) = R(1) \cup R(2) \cup \ldots \cup R(t-1)$ is the set of service requests that arrive before time slot t, $n_{j,k}^t \in n(t)$ is the number of instances of service k at cloudlet c_j, $R^*(t)$ denotes the set of service tasks that are currently being processed at cloudlets and t is the current time slot.

3) Action. In service placement stage of each time slot, the agents decide to install or delete some instances at their own cloudlets. Action $a_j \in \mathcal{A}$ of agent $\mathcal{N}_j$ is defined as a vector $a_j = (a_{j,0}, a_{j,1}, a_{j,2}, \ldots, a_{j,|F|})$, $\sum_{k=0}^{|F|} a'_{j,k} = 1$, where $a'_{j,k}$ represents the percentage of available computational capacity to be allocated to service f_k if $k \neq 0$ and $a'_{j,0}$ represents the percentage of available computational capacity will remain unused. The available computational capacity that we can allocate on c_j at time slot t is $cap_{t,j}^{rej} = CAP_j - \sum_{k=1}^{|F|} \phi_{j,k}^t \cdot c(f_k)$. The service placement decision variable $\delta_{j,k}^t$ can be obtained by $\delta_{j,k}^t = \lfloor \frac{cap_{t,j}^{rej} \cdot a'_{j,k}}{c(f_k)} \rfloor$.

4) State transition. After the agents take actions, the system state will transit to the next state, following the state transition function $P(s^t, a_1^t, a_2^t, \ldots, a_{|S|}^t, s^{t+1})$. First, the number of instances at any cloudlet $c_j \in S$ changes under the agent's action a_j in the service placement stage, that is, $n_{j,k}^t \leftarrow n_{j,k}^t + \delta_{j,k}^t$. Second, after the placement of service instance, a set of requests R^t arrives and we apply `Procedure 1` to assign the requests to cloudlets. After assigning the requests, since some running requests have finished and left in the end of each time slot, set $R^*(t+1)$ will be updated to exclude these finished requests.

5) Reward. To encourage each agent to maximize its contribution to the total profit, we define a reward function $\mathcal{R}_j(s^t, a_1^t, a_2^t, \ldots, a_{|S|}^t, s^{t+1})$ that describes how much reward an agent $\mathcal{N}_j$ receives after a state transition. Assume that we have a state transition $s^t = (R_\eta(t), n(t), R^*(t), t) \xrightarrow{a_1^t, a_2^t, \ldots, a_{|S|}^t} s^{t+1} = (R_\eta(t+1), n(t+1), R^*(t+1), t+1)$, and let $x_{t,i,j}^{new}$ and $x_{t,i,j}^{idle}$ be the results during the transition. The revenue and operation cost by cloudlet c_j is calculated, by both Eq. (8) and Eq. (9). The reward function of agent $\mathcal{N}_j$ thus is defined as follows

$$\mathcal{R}_j(s^t, a_1^t, a_2^t, \ldots, a_{|S|}^t, s^{t+1}) = RV_{c_j}(t) - C_{c_j}(t). \tag{11}$$

The cumulative reward of agent $\mathcal{N}_j$ from time slot t' to T then is defined as

$$G_j(t') = \sum_{t=t'}^{|T|} \mathcal{R}_j(s^t, a_1^t, a_2^t, \ldots a_{|S|}^t, s^{t+1}), \tag{12}$$

Procedure 1 A heuristic procedure for request assignment

```
Input: An MEC network $\mathcal{G} = (\mathcal{V} \cup S; E)$ with a set of cloudlets $C$, and a set of requests $R(t)$ that arrives at time slot $t$.
Output: Find an assignment of requests in $R(t)$ at time slot $t$
1: for $i \leftarrow 1, 2, \ldots, |R(t)|$ do
2:     Find a set $S'$ of cloudlets that has enough capacity and the delay requirement of $r_i$ can be fulfilled and a set $S'_{idle} \subset S'$ of cloudlets that has idle service instance $f_{k_i}$.
3:     if $S' \neq \emptyset$ then
4:         If $S'_{idle} \neq \emptyset$, find the cloudlets $c_j$ with maximum residual capacity $CAP_j - \sum_{k=1}^{|F|} n^t_{j,k} \cdot c(f_k)$, and set $x^{idle}_{i,j,t} \leftarrow 1$. Otherwise, $r_i$ is assigned to the cloudlet $c_j \in S' \backslash S'_{idle}$ with maximum residual capacity. We then update $x^{new}_{i,j,t} \leftarrow 1$.
5:     else
6:         $r_i$ is rejected and $x^t_{i,j} \leftarrow 0$.
7:     end if
8: end for
9: return $x^{new}_{i,j,t}$ and $x^{idle}_{i,j,t}$ for each request $r_i$ and cloudlets $c_j$.
```

We can see that $\sum_{j=1}^{|s|} G_j(0)$ is equivalent to (10).

The basic idea of the algorithm is that for any agent $\mathcal{N}_j$, we deploy a deep neural network(DNN) called *actor network* to approximate policy function π_j, where $\pi_j(a_j \mid s; \theta_j)$ is the distribution of probability of service placement decision on c_j under system state s and θ_j is the learnable DNN parameters. We make use of the Advantage Actor-Critic(A2C) algorithm to train the DNNs for better performance [10]. Specifically, define the expected cumulative rewards of agent $\mathcal{N}_j$ from time slot t as a function J^t_j with respect to parameter θ_j as follows

$$J^t_j(\theta_j) = \mathbb{E}_{a_j \sim \pi_{\theta_h}}[G_j(t); \theta_j]. \tag{13}$$

which means, our goal is to find the optimal parameter θ^*_j, to maximize $J^t_j(\theta_j)$. Stochastic gradient descent(SGD) is used to optimize the parameters [5]. In SGD, θ_j is updated as:

$$\theta_j \leftarrow \theta_j + \eta \nabla_{\theta_j} J^t_j(\theta_j), \tag{14}$$

where $\nabla_{\theta_j} J^t_j(\theta_j)$ is the gradient of $J^t_j(\theta_j)$ and η is the learning rate to control the step of gradient descent. As claimed by [1], we use

$$\nabla_{\theta_j} \log \pi_{\theta_j}(a^t_j \mid s^t; \theta_j)(G_j(t) - \mathbb{E}[G_j(t) \mid s^t]) \tag{15}$$

as the gradient, where $\mathbb{E}[G(t) \mid s^t]$ is the expected cumulative rewards that the agent obtains from state s^t following policy π_j. Thus, we introduce another DNN called *critic network* to output value function $V^{\pi_j}_j(s^t; \omega_j)$ to approximate $\mathbb{E}[G(t) \mid s^t]$. Then, θ_j is updated as follows.

$$\theta_j \leftarrow \theta_j + \eta \nabla_{\theta_j} \log \pi_{\theta_j}(a^t_j \mid s^t; \theta_j)(G_j(t) - V^{\pi_j}_j(s^t; \omega_j)). \tag{16}$$

To minimize the difference between the output $V^{\pi_j}_j(s^t; \omega_j)$ and $\mathbb{E}[G(t)|s^t]$, the value of ω_j is updated as follows

$$\omega_j \leftarrow \omega_j - \eta \nabla_{\omega_j}(G_j(t) - V^{\pi_j}_j(s^t; \omega_j))^2. \tag{17}$$

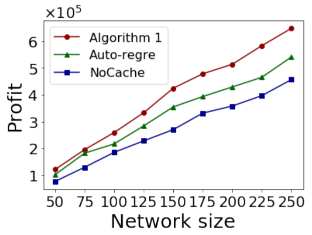
The training process then proceeds as follows. All the agents' actor and critic networks have randomly initialized parameter θ^0_j and ω^0_j. The agents simulate the whole service placement process using a historical monitoring period. From time slot 0 to T, agent

Algorithm 1 Training for the Actor network

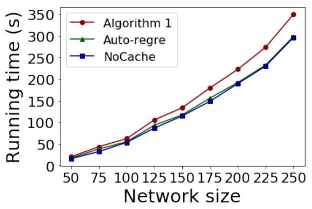
```
Input: An MEC network $\mathcal{G} = (\mathcal{V} \cup S; E)$ with a set of cloudlets $C$, and a set of services $F$ provided by the network and historical requests in the monitoring area.
Output: The service placement decision and request assignment at each time slot.
1: Randomly initialize an actor network with parameter $\theta_j$ and a critic network with parameter $\omega_j$ for each agent $\mathcal{N}_j$;
2: while $\theta_j$ and $\omega_j$ has not converged do
3:     Select a set $R'$ of service requests from request history;
4:     for $t \leftarrow 1, 2, \ldots, T$ do
5:         Let $\pi_j(a^t_j \mid s^t; \theta_j)$ be the output of the actor network of agent $\mathcal{N}_j$ for the input of system state $s^t$. Sample action $a^t_j$ according to the distribution $\pi_j(a^t_j \mid s^t; \theta_j)$ . Initialize or delete service instances according to $\delta^t_{j,k} \leftarrow \lfloor \frac{cap^{rej}_{t,j} \cdot a'^t_{j,k}}{c(f_k)} \rfloor$, $\forall k \in \{1, 2, \ldots, |F|\}$;
6:         Assign the set $R'(t)$ of requests one by one by invoking Procedure 1 and update the system according to the assignment.
7:         Calculate the rewards $G^t_j$ for each agent;
8:     end for
9:     for $t \leftarrow 1, 2, \ldots, T$ do
10:        Input system state $s^t$ into the critic networks of each agent $\mathcal{N}_j$ and update $\theta_j$ and $\omega_j$ following (16) and (17).
11:    end for
12: end while;
13: for $t \leftarrow 1, 2, \ldots, T$ do
14:     Let $\pi_j(a^t_j \mid s^t; \theta_j)$ be the output of the actor network of agent $\mathcal{N}_j$ for the input of system state $s^t$. Sample action $a^t_j$ according to the distribution $\pi_j(a^t_j \mid s^t; \theta_j)$ . Initialize or delete service instances according to $\delta^t_{j,k} \leftarrow \lfloor \frac{cap^{rej}_{t,j} \cdot a'^t_{j,k}}{c(f_k)} \rfloor$, $\forall k \in \{1, 2, \ldots, |F|\}$;
15:     Assign the set $R(t)$ of requests one by one by invoking Procedure 1 and update the system according to the assignment.
16: end for
17: return The service placement decision $\delta^t_{j,k}$ derived from the actions $a^t_j$ and the request assignments $x^{new}_{t,i,j}$, $x^{idle}_{t,i,j}$.
```

$\mathcal{N}_j$ inputs the system state s^t into its actor network and places the service instance in c_j based on the output $\pi_j(a_j \mid s^t; \theta^0_j)$. The system state s^t, actions a^t_j, and reward $\mathcal{R}^t_j$ are recorded at each time slot. After the agents finish the simulation of a historical monitoring period, we input s^t to critic networks to obtain $V^{\pi_j}_j(s^t; \omega^0_j)$. Then, we can calculate the gradient by Eq.(15) and update θ_j following Eq. (16), followed updating ω_j by Eq.(17). After updating θ_j and ω_j, we select another historical monitoring periods and repeat the simulation and training until θ_j and ω_j converge. Having trained the DNNs, the actor networks can then be used for service placement.

In summary, to solve the online service placement problem, we propose its equivalent MMDP formulation and then adopt the Advantage-Actor-Critic algorithm which uses historical requests to train the DNNs. In the end, we use the trained DNNs to decide the



(a)



(b)

Figure 1: Performance of different algorithms by varying the size of the MEC network from 50 to 250.

service placement at each time slot. The detailed process is shown in Algorithm 1.

4 PERFORMANCE EVALUATION

In this section, we evaluate the performance of the proposed algorithm.

4.1 Experimental settings

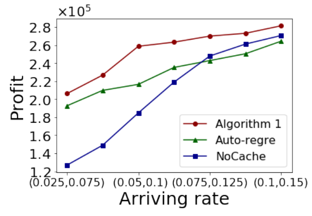
We consider an MEC network that consists of 100 APs, where 20% of them are equipped with cloudlets. The computing capacity of each cloudlet is randomly selected from 2,000 MHz to 4,000 MHz [4]. We assume that the MEC network provides 20 services, and each service instance requires 40 to 400 MHz computing capacity [4]. The network delay on the link between two APs is set from 2 *ms* to 5 *ms* [8]. The initialization delay of service instances varies from 20 *ms* to 40 *ms* [9].The initialization cost of an instance is randomly drawn within $[20, 50]$ [11]. The profit per time slot to process a user request is set within $[0.2, 0.3]$ per MHz in each time slot [11], and the idle cost for a type of service instance is set within $[0.02, 0.03]$ per MHz per time slot. The arrivals of a certain type of service requests at an AP follows a Poisson distribution, where the mean of Poisson distribution λ is randomly chosen within $[0.05, 1]$. The delay requirements of these requests vary from 10 *ms* to 50 *ms* and each request has a duration of 1-5 time slots [4].

We then compare Algorithm 1 against algorithm Auto-regre proposed by [11], where we adopt an auto-regression method to predict the number $n^t_{j,k}$ of required service instances. We also propose a baseline algorithm for service placement which immediately releases idle service instances and does not pre-install service instances, and we refer to this algorithm as NoCache.

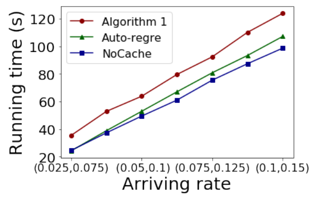
4.2 Performance evaluation for the online algorithm

We study the performance of algorithm Algorithm 1, by varying the network size from 50 to 250. The total profit of Algorithm 1 outperforms those delivered by both Auto-regression and NoCache in all cases, and the performance gap between Algorithm 1 and the other comparison algorithms become larger with the increase on network size. Fig. 1(b) depicts the running times of all algorithms. Among the comparison algorithms, Algorithm 1 has the longest running time. Although algorithms Auto-regression and NoCache have less running time, they earn much less profits.

We then investigate the impact of the arriving rate of tasks on Algorithm 1 by varying the mean of the Poisson distribution λ from within $[0.025, 0.075]$ to within $[0.1, 0.15]$. As shown in Fig. 2(a), the total profit of Algorithm 1 outperforms Auto-regression and NoCache, facing different frequency of request arriving rates. Also, we can see that Algorithm 1's finishes within an applicable time in Fig. 2(b).



(a)



(b)

Figure 2: Performance of different algorithms by varying the Possion distribution parameter λ from the interval $[0.025, 0.075]$ to the interval $[0.1, 0.15]$.

5 CONCLUSIONS

In this paper, we studied the online service placement and service request assignment problem in an MEC network while meeting service delay requirements. We proposed an efficient algorithm the problem and conducted experiments through simulations. Experimental results demonstrate that the proposed algorithm is efficient and superior to the benchmark algorithms.

REFERENCES

[1] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. 2016. Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv:1611.06256* (2016).

[2] Richard Cziva, Christos Anagnostopoulos, and Dimitrios P Pezaros. 2018. Dynamic, latency-optimal vNF placement at the network edge. In *Ieee infocom 2018-ieee conference on computer communications*. IEEE, 693–701.

[3] Ting He, Hana Khamfroush, Shiqiang Wang, Tom La Porta, and Sebastian Stein. 2018. 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)*. IEEE, 365–375.

[4] Mike Jia, Weifa Liang, and Zichuan Xu. 2017. QoS-aware task offloading in distributed cloudlets with virtual network function services. In *Proceedings of the 20th ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems*. 109–116.

[5] Jack Kiefer, Jacob Wolfowitz, et al. 1952. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics* 23, 3 (1952), 462–466.

[6] Jing Li, Weifa Liang, Meitian Huang, and Xiaohua Jia. 2020. Reliability-aware network service provisioning in mobile edge-cloud networks. *IEEE Transactions on Parallel and Distributed Systems* 31, 7 (2020), 1545–1558.

[7] Jing Li, Weifa Liang, Zichuan Xu, and Wanlei Zhou. 2020. Service provisioning for IoT applications with multiple sources in mobile edge computing. In *2020 IEEE 45th Conference on Local Computer Networks (LCN)*. IEEE, 42–53.

[8] Yu Ma, Weifa Liang, Zichuan Xu, and Song Guo. 2018. Profit maximization for admitting requests with network function services in distributed clouds. *IEEE Transactions on Parallel and Distributed Systems* 30, 5 (2018), 1143–1157.

[9] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. 2014. ClickOS and the Art of Network Function Virtualization. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation* (Seattle, WA) *(NSDI'14)*. USENIX Association, USA, 459–473.

[10] Fangxin Wang, Feng Wang, Jiangchuan Liu, Ryan Shea, and Lifeng Sun. 2020. Intelligent video caching at network edge: A multi-agent deep reinforcement learning approach. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2499–2508.

[11] Zichuan Xu, Weifa Liang, Mike Jia, Meitian Huang, and Guoqiang Mao. 2018. Task offloading with network function requirements in a mobile edge-cloud network. *IEEE Transactions on Mobile Computing* 18, 11 (2018), 2672–2685.

[12] Zichuan Xu, Zhiheng Zhang, Weifa Liang, Qiufen Xia, Omer Rana, and Guowei Wu. 2020. QoS-aware VNF placement and service chaining for IoT applications in multi-tier mobile edge networks. *ACM Transactions on Sensor Networks (TOSN)* 16, 3 (2020), 1–27.