Deep Reinforcement Learning for Dependency-aware Microservice Deployment in Edge Computing

Chenyang Wang*, Bosen Jia*, Hao Yu†, Xiuhua Li‡, Xiaofei Wang* and Tarik Taleb†
*College of Intelligence and Computing, Tianjin University, Tianjin, China.
†Information Technology and Electrical Engineering, University of Oulu, Oulu, Finland.
‡School of Big Data & Software Engineering, Chongqing University, Chongqing, China.

***Abstract*—Recently, we have observed an explosion in the intellectual capacity of user equipment, coupled by a meteoric rise in the need for very demanding services and applications. The majority of the work leverages edge computing technologies to accomplish the quick deployment of microservices, but disregards their inter-dependencies. In addition, while constructing the microservice deployment approach, several research disregard the significance of system context extraction. The microservice deployment issue (MSD) is stated as a max-min problem by concurrently evaluating the system cost and service quality. This research first analyzes an attention-based microservice representation approach for extracting system context. The attention-modified soft actor-critic method is proposed to the MSD issue. The simulation results reveal the ASAC algorithm's priorities in terms of average system cost and system reward.**

I. Introduction

The exceptional growth of intelligent devices and the need to pave the way for 5G or beyond 5G communication systems promote a wide variety of high-demanding services and applications, e.g., face recognition, virtual/augmented reality (AR/VR), and 3D games [1]. These applications are resource-hungry, and rapid response is required. To cope with the challenges above, some studies [2]–[5] deploy the services in cloud servers with the benefit of rapid elasticity, on-demand resource pooling, and self-configuration. However, transmitting massive data from devices to the remote cloud center usually produces unpredictable latency and excessive resource consumption. It is crucial to design revolutionary schemes of microservice deployment technologies to address these challenges.

Amount of studies focus on service deployment by utilizing edge computing technology [6] to leverage the computation resources in proximity to data sources, e.g., literatures [7], [8] investigate the optimization strategies of service deployment under cloud-edge architecture to achieve the minimum service completion time. However, the system environment and dependencies between services are ignored. Typically, a service is usually decoupled into multiple microservices with the characteristics of low cost, flexibility, and scalability, to achieve the fast response and dynamic deployment of various services and applications [9]. Generally, the dependency of microservices is modeled by a directed acyclic graph (DAG), reflecting the order in which microservices are executed [10]. The problem with different dependencies of services is that the communication protocols are heterogeneous, i.e., different microservices may be transmitted by various types of channels, resulting in extra execution/response time even for the same microservice deployed in the different locations.

To depict the heterogeneity and dynamics of the environment from the changes in infrastructure conditions (e.g., the waiting queue of service in an edge server) and the services involved (e.g., the topology of service DAG). Other studies focus on the utilization of artificial intelligence (AI) technologies, e.g., deep learning (DL), deep reinforcement learning (DRL), to design the service deployment strategies [11]–[13]. For instance, the literature [13] utilizes a DL method to learn branching/pruning policies for optimizing the service chain implementation. Therefore, it is necessary to abstract the key features of the system context to better represent the inner dependencies between the microservices and pass the information of infrastructure influenced.

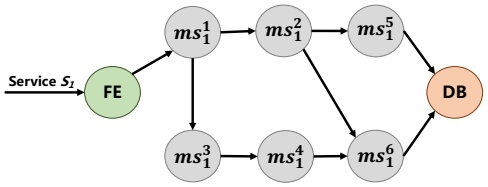


Fig. 1. Illustration of service DAG.

In this paper, we investigate an efficient microservice deployment strategy for edge computing, we consider a three-layer *UE-Edge-Cloud* architecture, the microservices are deployed in any layer according to the optimization model, and the dependencies between microservices are modeled as a DAG. Then an attention mechanism-based microservice representation (AMR) algorithm is carried out to extract system context information. Finally, distinguished from the state-of-the-art, we model the microservice deployment problem (MSD) as a Markov Decision Process (MDP), and the attention-modified soft actor-critic (ASAC) algorithm is proposed to derive the optimal decision making. We summarize the main

contributions of this paper as follows:

- We introduce the dependency of microservices as a directed acyclic graph (DAG), making it more practical for deployment, the microservices deployment problem (MSD) is modeled as the *NP-hard* max-min problem by jointly considering the optimization of overall system cost and the Quality of Service (QoS) of UEs.
- We extract the comprehensive network information using an attention-based microservices representation (AMR) method. The system context of both infrastructure and microservice features is embedded and concatenated into a meta-chain by a multi-head attention mechanism.
- We model the max-min problem as a Markov decision process (MDP) and propose an attention-modified soft actor-critic algorithm named ASAC to solve the above problem. Simulation results also show the priorities of the proposed algorithms.

The rest of this article is organized as follows: Section II introduces the system model and the optimization problem, the proposed algorithm is derived in Section III and the experimental simulation is conducted in Section IV. Finally, we conclude this paper in Section V.

II. System Model and Problem Formulation

We introduce the overall three-layer architecture consisting of a cloud data center C in the *Cloud Layer*, edge servers $K = \{1, 2, ..., k\}$ in the *Edge Layer*, and mobile users $M = \{1, 2, ..., m\}$ in the *UE Layer*. Given $M = \{1, 2, ..., m\}$ UE, $S = \{S_1, S_2, ..., S_n\}$ services, each service is represented by a DAG. Denote the service DAG $S_n^D = \{MS_n, \mathcal{E}_n\}$, where $MS_n = \{ms_n^i | i = 1, 2, ..., I\}$ is the set of microservice of S_n and $\mathcal{E}_n = \{e_n^{ij} | i, j \in \{1, 2, ..., I\}, i \neq j\}$ represents the set of dependencies of microservices, *i.e.*, the precedence relation such that microservice ms_n^i is completed before ms_n^j starts.

As illustrated in Fig.1, when service S_1 consisting of 6 microservices is delivered to a forward-equipment (FE), note that ms_1^2 cannot start its execution before ms_1^1 finished, and the database (DB) receives S_1 when all the microservices are accomplished. Specifically, we define FE and DB as two virtual nodes ms_n^o and ms_n^{I+1} indicating the entry and exit of service S_1, respectively. Each $ms_n^i \in \mathbb{S}_n$ is associated with the two-tuples (c_n^i, d_n^i), where c_n^i is the required CPU cycles to finish ms_n^i and d_n^i denotes the size of input microservice.

We define a deployment variable $\alpha_n^{i,l} = \{0, 1\}$, $l \in \mathcal{I}$, $\mathcal{I} = \{M, K, C\}$, e.g., when $l = k$ means that ms_n^i is deployed on edge server k, $\alpha_n^{i,l} = 1$ means that ms_n^i is deployed, 0 is otherwise. We have the following definitions to introduce the time and energy consumption of service execution:

Definition 1 (*Ready Time*): The time that ms_n^i has all prerequisites for execution, denoted as $RT_n^{i,l}$, $l \in \mathcal{I}$.

Definition 2 (*Finish Time*): The time that ms_n^i accomplishes all the workload c_n^i, denoted as $FT_n^{i,l}$, $l \in \mathcal{I}$.

Definition 3 (*Wireless Receiving Time*): Define $RT_n^{i,wt,l}$, $FT_n^{i,wt,l}$, $l \in \mathcal{I}$ as the ready time and finish time of ms_n^i when receiving the wireless channel from the *Edge Layer* and *Cloud Layer*, respectively.

A. UE Layer Execution Model

Assume that each UE has σ^m cores with the c^m CPU frequency. Denote $FT^{\sigma,m}$ as the minimum finish time in UE m, then the ready time is $RT_n^{i,m} = \max_{m\prime \in pre\{m\}} g_n^{i,m}$, and $g_n^{i,m} = max\{FT_n^{i,m\prime}, FT_n^{i,wr,m\prime}, FT_n^{i,wr,k}, FT_n^{i,wr,c}, FT^{\sigma,m\prime}\}$, where $pre\{m\}$ is the set of immediate predecessors of ms_n^i, note that the execution of ms_n^i will not start unless all the predecessors have been accomplished. Accordingly, we have the local processing time as $T_n^{i,m} = \frac{d_n^i}{c_m^{UE}}$. Thus, the finish time of ms_n^i at *UE Layer* is $FT_n^{i,m} = RT_n^{i,m} + T_n^{i,m}$. The corresponding energy consumption of ms_n^i at local execution can be obtained as $\epsilon_n^{i,m} = \kappa_m d_n^i (c_m^{UE})^2$ [14], where κ_m is the coefficient related on chip types. Note that we have $\epsilon_n^{0,m} = \epsilon_n^{I+1,m} = 0$ for the FE and DB, respectively.

B. Edge Layer Execution Model

When microservice ms_n^i is deployed at an edge server k, assume that UE m directly sends ms_n^i to k via cellular links, denote the transmission time as $T_n^{i,w,k} = P_n^i / v_{m,k}$, where $v_{m,k}$ is the uplink transmission rate [15]. We set the channel gain as $g^{m,k} = -4\ db$ power of the distance between UE m and edge server k. In this case, the energy consumption of UE m is $\epsilon_n^{i,k} = g^{m,k} \times T_n^{i,w,k}$. Thus, the ready time on *Edge Layer* is $RT_n^{i,k} = \max_{m\prime \in pre\{m\}} g_n^{i,k} + T_n^{i,w,k}$, and we have $g_n^{i,k} = max\{FT_n^{i,m\prime}, FT_n^{i,wr,m\prime}, FT_n^{i,wr,k}, FT_n^{i,wr,c}\}$. Suppose that each edge server equips σ_m^k cores with the c_m^k CPU frequency, and the minimum accomplishment time for all the microservices is denoted as $FT_m^{\sigma^k}$. Note that we consider both the edge servers and cloud server can satisfy the demand to potentially perform the concurrent microservices, thus we have $\sigma_m^k = \infty$. Therefore, the execution time of ms_n^i can be calculated as $T_n^{i,k} = \frac{d_n^i}{c_m^k}$. Consequently, the finish time of ms_n^i on *Edge Layer* is $FT_n^{i,k} = RT_n^{i,k} + T_n^{i,k}$. The energy consumption in edge server k is $\epsilon_n^{i,k} = \kappa_k d_n^i (c_m^k)^2$.

C. Cloud Layer Execution Model

If the microservice ms_n^i is deployed in the *Cloud Layer*, similar to the execution on *Edge Layer*. We consider that the ms_n^i is first sent to edge server then delivered to cloud server directly via fiber connections, the data transmission delay can be ignored in this way. Thus, the ready time of ms_n^i on *Cloud Layer* can be regraded as $RT_n^{i,C} = FT_n^{i,k}$. The CPU capability of the cloud is denoted as c^C, and the execution time of ms_n^i is $T_{n,i}^C = d_n^i / c^C$. Accordingly, the energy consumption in *Cloud Layer* can be obtained as $\epsilon_n^{i,C} = \kappa_C d_n^i (c^C)^2$. Therefore, the finish time of ms_n^i is presented as $FT_n^{i,C} = RT_n^{i,C} + T_n^{i,C}$.

Recall that the deployment variable $\alpha_n^{i,l} = \{0, 1\}$, here we define the average cost of time-varying energy (CTE) ξ^m as:

$$\xi^m = \omega_t \times (FT_n^{I+1,l} - FT_n^{0,l}) + \omega_e \times E, \tag{1}$$

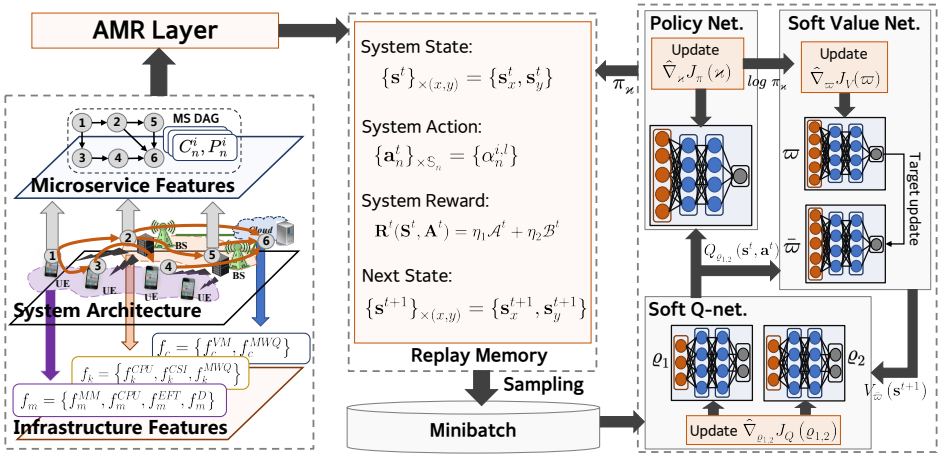


Fig. 2. Implementation of ASAC scheme.

where E is the total energy consumption of all devices, holding $\omega_t + \omega_e = 1$, are the coefficients of execute time and energy consumption, respectively. Besides, we define the microservices deployment fee (MDF) as ζ^l, $l \in \mathcal{I}$, where $\zeta^m \gg \zeta^k \gg \zeta^C$. The overall system cost is expressed as:

$$\mathcal{A} = \sum_{i \in ms_n^i} \sum_{x \in X} \sum_{n \in S_n^D} \left(\alpha_n^{i,l} \zeta^l + \frac{1}{X} \sum_{x=1}^{X} \xi^x \right), \tag{2}$$

where $X \subseteq \mathcal{I}$ is the number of the devices executing the microservices.

Define $\delta = t_{UE-FE} + t_{ms_n^i}$ is the whole time consumption where t_{UE-FE} is the communication latency between UE and FE, and $t_{ms_n^i}$ is the serving time which is regarded as ms_n^i processing time related on the hardware conditions of devices. Furthermore, assume that δ^{max} is the maximal tolerant time of the UE with the demand for the delay-sensitive services, and we denote $q_n^{i,x}$ as the indicator function that satisfies the demand of a UE when executing the microservices.

$$q_n^{i,x} = \begin{cases} 1, & \delta_n^{i,x} < \delta_{max} \\ 0, & otherwise \end{cases}. \tag{3}$$

Thus, the Quality of Service (QoS) can be obtained as:

$$\mathcal{B} = \sum_{i \in ms_n^i} \sum_{x \in X} \sum_{n \in S_n^D} \alpha_n^{i,l} q_n^{i,x} \tag{4}$$

Finally, the problem of microservices deployment (MSD) can be modeled as the max-min joint optimization problem, which minimizing the overall system cost to promise the QoS of all the UE in the system, shown as follows:

$$\begin{aligned} \max_{q} \min_{\alpha} \quad & \mathcal{Z}(\mathcal{A}, \mathcal{B}) \\ s.t. \quad & \omega_t + \omega_e = 1 \\ & \alpha = \{0, 1\} \\ & \zeta^m \gg \zeta^k \gg \zeta^C \\ & q_n^{i,x} \in \{0, 1\} \end{aligned} \tag{5}$$

Due to the terms $\alpha_n^{i,l}\zeta^l$ in the objective (2), the problem can be regarded as a mixed binary integer linearly constrained programming (MBILP). Meanwhile, the objective (4) is a binary integer linearly constrained quadratic programming (LCQP) problem for the presence of the quadratic terms $\alpha_n^{i,l} q_n^{i,x}$. A similar problem is proved as *NP-hard* [16], thereby making it not feasible to solve it by heuristic algorithm or dynamic programming because of its high computational and spatial complexity and large scale. Thus, we carry out an attention-modified DRL method to solve the aforementioned problem.

III. Attention based Deep Reinforcement Learning

This section first introduces the attention-based microservice representation (AMR) layer to extract the system features using a multi-head attention mechanism. Then, a modified DRL algorithm is derived to solve the above joint optimization problem. The implementation process is shown in Fig.2.

A. Attention based Microservice Representation

*1) **System Features Embedding**:* We extract the system features from two aspects, i.e., the infrastructure and microservices. The infrastructure feature f_x, $x \in \mathcal{I}$ is based on the calculation of service and energy overhead, related to the features of user devices, edge servers, and cloud server.

Aiming to learn the embedding of the infrastructure status, we have the embedding $\mathbf{w}_x = \mathbf{H} \cdot f_x$, where the transformation matrix $\mathbf{H}$ is used to map f_x. Considering the dependencies, we employ the same matrix $\mathbf{H}$ to map the microservice's features $f_y, y \in ms_n^i$ (e.g., the size/type of microservice). Similarly, we can obtain the corresponding embedding as $\mathbf{w}_y = \mathbf{H} \cdot f_y$.

*2) **Microservice Meta-chain Attention**:* We consider the infrastructure-microservice pair as a structure, and a meta-chain of microservice DAG $\mathbf{O}$ is defined to obtain the comprehensive representation. The structure attention τ_{xy} indicates the importance of microservice to infrastructure embedding on $\mathbf{O}$. In this way, we can obtain the representation of the infrastructure by integrating the learnable weighted sum of the neighbouring service as $\mathbf{w}_x^{\mathbf{O}} = \psi(\sum_{x \in N_x^{\mathbf{O}}} \mu_{xy}^{\mathbf{O}} \cdot \mathbf{w}_y)$, where $\mu_{xy}^{\mathbf{O}} = softmax\{\tau_{xy}^{\mathbf{O}}\}$ indicating the operation of graph structure information injection by using masked attention

mechanism. Then, the multi-attention mechanism is utilized to stabilize the learning process [17] by applying U independent heads to compute the hidden states as follows:

$$\mathbf{w}_x^{\mathbf{O}} = \psi(\frac{1}{U}\sum_{u=1}^{U}\sum_{y\in N_x^{\mathbf{O}}}\mu_{xy}^{\mathbf{O}}\cdot\mathbf{w}_y). \tag{6}$$

Finally, we have U groups of chain-specific representations of the infrastructure and microservice embeddings space $\mathbf{W}_x^{\mathbf{O}_u}$ and $\mathbf{W}_y^{\mathbf{O}_u}$, $u = \{1, 2, \ldots, U\}$, respectively. Therefore, from the infrastructure perspective, the score function $\Lambda^{\mathbf{O}_u}$ of service-chain $\mathbf{O}_u$ is introduced to indicate the importance of the different meta-chain, shown as follows:

$$\Lambda^{\mathbf{O}_u} = \frac{1}{\mid U \mid}\sum_{x\in U}^{U}\mathbf{g}^T\cdot tanh\left(\mathbf{W}\cdot\mathbf{w}_x^{\mathbf{O}_u}+\mathbf{b}\right), \tag{7}$$

where $\mathbf{g}$ is attention coefficient vector, $\mathbf{W}$ is the weight matrix and $\mathbf{b}$ denotes the bias vector.

Taking the learned meta-chain attention as coefficients in system, the final infrastructure (FI) embedding $\mathbf{W}_x$ and final service (FS) embedding $\mathbf{W}_y$ can be derived by the aggregation of the service-chain embeddings as:

$$\begin{cases}\mathbf{W}_x = \sum_{u=1}^{U}\omega^{\mathbf{O}_u}\cdot\mathbf{W}_x^{\mathbf{O}_u}, & x\in\mathcal{I}\\ \mathbf{W}_y = \sum_{u=1}^{U}\omega^{\mathbf{O}_u}\cdot\mathbf{W}_y^{\mathbf{O}_u}, & y\in ms_n^i\end{cases}, \tag{8}$$

where $\omega^{\mathbf{O}_u}$ is the normalized softmax function $\Lambda^{\mathbf{O}_u}$.

B. DRL-based Microservice Deployment Strategy Design

We consider the deployment policy learning is in the continuous action spaces, and an infinite-horizon Markov Decision Process (MDP) is deployed as follows:

1) System State: As aforementioned, we obtain the final representation of infrastructure $\mathbf{W}_x$ and service embedding $\mathbf{W}_y$. Here we define the system state vector space with two-tuple $\mathbf{S}^t = \{\mathbf{s}^t\}_{\times(x,y)} = \{\mathbf{s}_x^t, \mathbf{s}_y^t\}$, where $\mathbf{s}_x^t = (\mathbf{W}_x^t)_{x\in\mathcal{I}}$ and $\mathbf{s}_y^t = (\mathbf{W}_y^t)_{y\in ms_n^i}$ indicate the system infrastructure state and microservice placement state, respectively.

2) System Action: The system decides which microservice is executed in which infrastructure. We define the system action space as $\mathbf{A}^t = \{\mathbf{a}_n^t\}_{\times\mathbb{S}_n} = \{\alpha_n^{i,l}\}, \forall n, i\in ms_n^i, l\in\mathcal{I}$.

3) System Reward: It is calculated as the weighted sum of current reward r, we formulate the system reward according to the joint optimization objective, expressed as $\mathbf{R}^t(\mathbf{S}^t, \mathbf{A}^t) = \eta_1\mathcal{A}^t + \eta_2\mathcal{B}^t$, where $\mathcal{A}^t = (\alpha_n^{i,l}\zeta^l + \frac{1}{X}\sum_{x=1}^{X}\xi^x)^t$ denotes the overall system cost and $\mathcal{B}^t = (\alpha_n^{i,l}q_n^{i,x})^t$ is the QoS, respectively. The coefficients hold $\eta_1+\eta_2 = 1$, and $\mathcal{A}$, $\mathcal{B}$ are both non-negative, the system reward meets $\mathbf{R}^t > 0$.

By utilizing the SAC mechanism, we consider the stochastic policy by augmenting the cumulative system reward with the expected entropy of the policy over $\rho_\pi(\mathbf{s}^t)$ in the finite-horizon scenario. The objective is to find the optimal policy π^* as:

$$\pi^\star = arg\ \max_{\pi}\sum_{t=0}^{T}\mathbb{E}_{(\mathbf{s}^t,\mathbf{a}^t)\sim\rho_\pi}\left[r\left(\mathbf{s}^t,\mathbf{a}^t\right)+\phi\mathcal{H}\left(\pi\left(\cdot\mid\mathbf{s}^t\right)\right)\right], \tag{9}$$

where $\phi > 0$ is the trade-off coefficient indicating the relative importance of the entropy term against reward, so as to control the stochastic of the optimal deployment policy. $\mathcal{H}\left(\pi\left(\cdot\mid\mathbf{s}^t\right)\right) = \mathbb{E}_{\mathbf{a}^t\sim\rho_\pi}\left[-log_\pi\left(\mathbf{a}^t\mid\mathbf{s}^t\right)\right]$ is the policy entropy which measures the uncertainty of the random variable.

Algorithm 1 ASAC Algorithm

Initialize: Infrastructure features $f_x, x\in\mathcal{I}$;
Soft Q-function parameters ϱ_1, ϱ_2;
Value function parameters ϖ and target value network parameters $\bar{\varpi}$. Soft policy parameters $\varkappa$.
for $t = 0, 1, 2, ..., T$ **do**
  Obtain the system state $\{\mathbf{s}^t\}_{\times(x,y)}$ and action $\{\mathbf{a}_n^t\}_{\times\mathbb{S}_n}$ by AMR algorithm;
  for each episode **do**
    Get system action $\mathbf{a}^t$ according to input system state $\mathbf{s}^t$ in actor network, holding $\mathbf{a}^t\sim\pi_\varkappa(\mathbf{a}^t\mid\mathbf{s}^t)$.
    Obtain current reward r^t and next system state $\mathbf{s}^{t+1}$.
    Store the quadruplet into replay memory $\mathcal{D}\leftarrow\{(\mathbf{s}^t,\mathbf{a}^t,r^t,\mathbf{s}^{t+1})\}\cup\mathcal{D}$.
    Randomly sample a batch of $\mathcal{N}$ samples from $\mathcal{D}$.
  end for
  for each gradient step in batch $\mathcal{N}$ **do**
    Update the soft Q-function parameters ϱ_1 and ϱ_2 according to (11).
    Update the soft value function parameters $\bar{\varpi}\leftarrow\tau\varpi+(1-\tau)\bar{\varpi}$ according to (12).
    Update soft policy parameters $\varkappa$ according to (14).
  end for
end for

C. Soft Deployment Policy Model

In the maximum entropy paradigm, policy evaluation and policy improvement alternate in order to learn the optimal maximum entropy policies. With a deterministic deployment policy, we can obtain the soft Q-value starting from any function Q iteratively with a modified Bellman operator $\Gamma^\pi Q\left(\mathbf{s}^t,\mathbf{a}^t\right) \triangleq r\left(\mathbf{s}^t,\mathbf{a}^t\right)+\gamma\mathbb{E}_{\mathbf{s}^{t+1}\sim p}\left[V\left(\mathbf{s}^{t+1}\right)\right]$ where $V\left(\mathbf{s}^t\right) = \mathbb{E}_{\mathbf{a}^t\sim\pi}\left[Q\left(\mathbf{s}^t,\mathbf{a}^t\right)-log\ \pi\left(\mathbf{a}^t|\mathbf{s}^t\right)\right]$ is the soft state-value function.

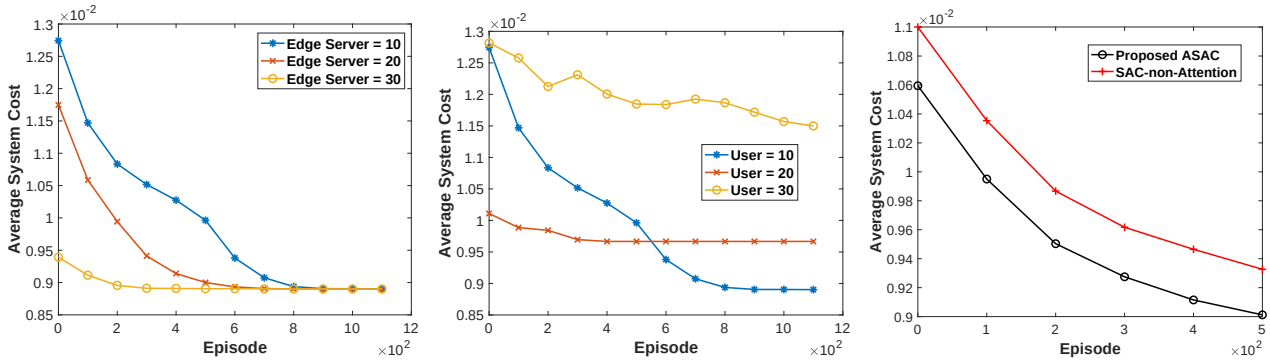
Accordingly, the entropy-augmented soft returns is obtained as $r_{soft}(\mathbf{s}^t,\mathbf{a}^t) \triangleq r\left(\mathbf{s}^t,\mathbf{a}^t\right)+\gamma\mathbb{E}_{\mathbf{s}^{t+1}\sim\rho_\pi}\left[\theta\mathcal{H}\left(\pi\left(\cdot\mid\mathbf{s}^{t+1}\right)\right)\right]$ which indicates that the accumulated returns under the system state $\mathbf{s}^t$ obtained by the current deployment policy π. The main purpose is to find the new deployment policy π_{new} which is better than the current π_{old}. Denote $\pi\in\Pi$ as the set of policies, to obtain the guaranteed deployment policy improvement, we update it by using Kullback-Leibler (KL) divergence, and output the Gaussian distribution as follows:

$$\pi_{new}(\cdot|\mathbf{s}^t) = arg\ \min_{\pi'\in\Pi}D_{KL}\left(\pi'\left(\cdot|\mathbf{s}^t\right)\parallel\frac{exp\left(Q^{\pi_{old}}\left(\mathbf{s}^t\cdot\right)\right)}{Z^{\pi_{old}}\left(\mathbf{s}^t\right)}\right), \tag{10}$$

where $D_{KL}(\cdot)$ is the KL divergence operation, and the partition function $Z^{\pi_{old}}\left(\mathbf{s}^t\right)$ is used to normalize the distribution.

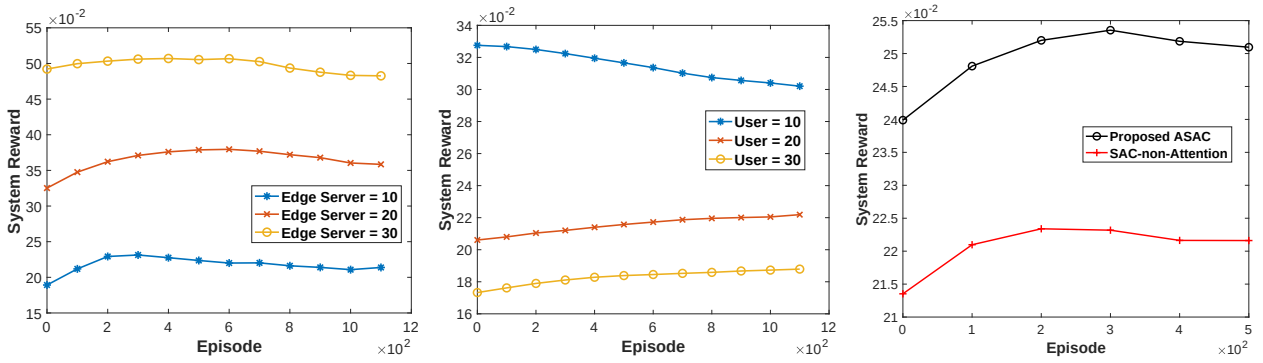
D. ASAC Learning Process

To deal with the large continuous domains obtained from the AMR layer, the function approximators for both Q-function



(a) Different edge servers. (b) Different UEs. (c) The proposed ASAC and SAC.

Fig. 3. Performance of average system cost under different number of edge servers, UEs and comparison between the proposed ASAC and SAC.



(a) Different edge servers. (b) Different UEs. (c) The proposed ASAC and SAC.

Fig. 4. Performance of system reward under different number of edge servers, UEs and comparison between the proposed ASAC and SAC.

and policy are employed by alternating between optimizing bother networks with stochastic gradient descent (SGD).

The soft Q-function parameters can be trained by minimizing the soft Bellman residual, shown as follows:

$$\hat{\nabla}_{\varrho} J_Q(\varrho) = \nabla Q_{\varrho}(\mathbf{s}^t, \mathbf{a}^t)\left(Q_{\varrho}(\mathbf{s}^t, \mathbf{a}^t) - r^t - \gamma V_{\bar{\varpi}}(\mathbf{s}^{t+1})\right) \tag{11}$$

where the $\bar{\varpi}$, from a target value network $V_{\bar{\varpi}}$, is used to stabilize the training process by exponentially moving average of the soft Q-function weights. The soft value function $V_{\varpi}(\mathbf{s}^t)$ is to trained by minimizing the mean squared error (MSE) with the unbiased estimator as:

$$\begin{aligned}\hat{\nabla}_{\varpi} J_V(\varpi) =& \nabla_{\varpi} V_{\varpi}(\mathbf{s}^t)(V_{\varpi}(\mathbf{s}^t) - Q_{\varrho}(\mathbf{s}^t, \mathbf{a}^t) \\ &+ log\pi_{\varpi}(\mathbf{a}^t \mid \mathbf{s}^t)])^2].\end{aligned} \tag{12}$$

In this way, it is observed that the actions are selected according to the current policy, rather than from the replay buffer $\mathcal{D}$. Similarly, the policy parameter can be trained by directly minimizing the KL-divergence from (10) as:

$$\begin{aligned}J_{\pi}(\varkappa) =& \mathbb{E}_{\mathbf{s}^t \sim \mathcal{D}, \mathring{a}^t \sim \mathring{A}}[log\pi_{\varpi}(f_{\varpi}(\mathring{a}^t; \mathbf{s}^t) \mid \mathbf{s}^t) \\ &- Q_{\varrho}(\mathbf{s}^t, f_{\varpi}(\mathring{a}^t; \mathbf{s}^t))],\end{aligned} \tag{13}$$

where $\mathring{a}$ is an input noise vector, sampled from a fixed spherical Gaussian distribution, holding $f_{\varpi}(\mathring{a}^t; \mathbf{s}^t) = \mathbf{a}^t$. In this way, we can approximate the gradient as follows:

$$\begin{aligned}\hat{\nabla}_{\varkappa} J_{\pi}(\varkappa) =& \nabla_{\varkappa} log\pi_{\varkappa}(\mathbf{a}^t \mid \mathbf{s}^t) + \nabla_{\varkappa} f_{\varkappa}(\mathring{a}^t; \mathbf{s}^t) \\ &\left(\nabla_{\mathbf{a}^t} log\pi_{\varkappa}(\mathbf{a}^t \mid \mathbf{s}^t) - \nabla_{\mathbf{a}^t} Q(\mathbf{s}^t, \mathbf{a}^t)\right).\end{aligned} \tag{14}$$

The process of the proposed ASAC is shown in Algorithm.1. Input the system state $\{\mathbf{s}^t\}_{\times(x,y)}$ and action $\{\mathbf{a}^t\}_{\times \mathbb{S}_n}$ obtained from AMR layer (Line 3), compute the current reward and next system state at each episode (Line 4-9), then update the soft Q-function, value function, and soft policy parameters until convergence (Line 10-14). The complexity is derived hereafter, there are K iterations in the outer loop, and a batch of $\mathcal{N}$ samples in one episode in the inner loop, thus the complexity of ASAC is $O(K\mathcal{N})$.

IV. Experiment

A. Simulation Settings

In this section, we conduct the experimental simulations in the python environment, the main parameter values of computation ability of UE layer c^m, edge layer c_m^k, and cloud layer c^C are uniformly set as 1.0-1.2GHz, 2.4-2.5GHz and 3.0GHz. The discount rate γ is 0.85, the learning rate is $3e-4$, and the batch size is 128. Other parameters of simulations in this work are similar to [15].

B. Simulation Results

We demonstrate the simulation results in terms of the performance of average system cost and the system reward. All the results are obtained by the average values of 20 times.

1) Performance of average system cost: We deploy 20 UEs and 30 microservices in the system for the demonstration. From Fig. 3(a), it is observed that ASAC reaches the maximum average system cost when the number of Edge servers is 10, while the optimal cost occurs when the number is 30. With the increase in the number of Edge servers, the system cost becomes smaller until it converges. This is mainly because fewer edge servers mean less disposable network resources, so as the number of edge servers increases, system cost becomes smaller. Fig. 3(b) shows the average system cost performance with different UEs, 10 edge servers and 30 microservices are deployed in this case. It can be seen that when the number of UE is 10, ASAC has the worst performance in the beginning but achieves the lowest cost in the end, while UE is 30, the performance is the worst. This is because the AMR algorithm occupies many network resources at the beginning of the system representation, and finally, the resources are optimally allocated with the system operation. We deploy 10 edge serves, 20 UEs, and 30 microservices in the following case. Fig. 3(c) shows that the proposed ASAC reduces 5% average system cost compared to the original SAC [18]. The main reason is that ASAC extracts the critical features from the system, accelerating the deployment speed at a lower cost.

2) Performance of system reward: For the same settings as the above demonstration, respectively, we carry out the performance of system reward, shown as Fig. 4. It can be observed that when the edge server is 30, ASAC achieves the best performance in Fig. 4(a), the performance improvement is positively correlated with the number of edge servers while there is a negative correlation with the increase of users in Fig. 4(b). The reason is that when UE is 10, it occupies many network resources, which leads to the high system cost in the beginning, while when UE is 30, the average system cost is over-consumed, resulting in the low QoS for each UE. Fig. 4(c) shows that the priorities of the proposed ASAC outperform the SAC algorithm for almost 30%.

V. Conclusion

This paper has investigated the microservice deployment (MSD) problem, formulated as the max-min problem, by jointly considering the overall system cost and the quality of service (QoS). For better extracting the system context, an attention-based microservice representation (AMR) has been carried out, and then the MSD problem has been formulated as a Markov decision process. Moreover, an attention-modified soft actor-critic algorithm (ASAC) has been derived to solve the above problem. The experimental simulation results have shown the superiority of the proposed algorithm.

Acknowledgment

This work was partially supported by the CHARITY project that received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101016509. This work was also supported partially by the National Key Research and Development Program of China under Grant No. 2019YFB2101901; the National Science Foundation of China under Grant No. 62072332, China NSFC (Youth) through grant No. 62002260; and the China Postdoctoral Science Foundation under Grant No. 2020M670654, and the National NSFC (Grant No. 61902044), Key Research Program of Chongqing Science & Technology Commission (Grants No. cstc2021jscx-dxwtBX0019) and the Chinese Government Scholarship (NO. 202006250167) awarded by China Scholarship Council.

References

[1] J. B. Gomes Gilzamir, Creto A and Y. L. Nogueira, "Two level control of non-player characters for navigation in 3d games scenes: A deep reinforcement learning approach," in *SBGames*, 2021, pp. 182–190.

[2] H. Yu, T. Taleb, J. Zhang, and H. Wang, "Deterministic latency bounded network slice deployment in ip-over-wdm based metro-aggregation networks," *IEEE TNSE*, vol. 9, no. 2, pp. 596–607, 2022.

[3] H. Yu, T. Taleb, and J. Zhang, "Deterministic latency/jitter-aware service function chaining over beyond 5g edge fabric," *IEEE TNSM*, 2022.

[4] H. Sun, S. Wang, F. Zhou, L. Yin, and M. Liu, "Dynamic deployment and scheduling strategy for dual-service pooling based hierarchical cloud service system in intelligent buildings," *IEEE T Cloud Comput*, 2021.

[5] Y. He, G. Han, J. Jiang, H. Wang, and M. Martinez-Garcia, "A trust update mechanism based on reinforcement learning in underwater acoustic sensor networks," *IEEE Transactions on Mobile Computing*, 2020.

[6] X. Wang, C. Wang, X. Li, V. C. Leung, and T. Taleb, "Federated deep reinforcement learning for internet of things with decentralized cooperative edge caching," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9441–9455, 2020.

[7] J. Islam, T. Kumar, I. Kovacevic, and E. Harjula, "Resource-aware dynamic service deployment for local iot edge computing: Healthcare use case," *IEEE Access*, vol. 9, pp. 115 868–115 884, 2021.

[8] Z. Fan, W. Yang, F. Wu, J. Cao, and W. Shi, "Serving at the edge: An edge computing service architecture based on icn," *ACM TOIT*, vol. 22, no. 1, pp. 1–27, 2021.

[9] H. Tian, X. Xu, T. Lin, Y. Cheng, C. Qian, L. Ren, and M. Bilal, "Dima: Distributed cooperative microservice caching for internet of things in edge computing by deep reinforcement learning," *WWW*, pp. 1–24, 2021.

[10] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading tasks with dependency and service caching in mobile edge computing," *IEEE TPDS*, vol. 32, no. 11, pp. 2777–2792, 2021.

[11] Y. Zhai, T. Bao, L. Zhu, M. Shen, X. Du, and M. Guizani, "Toward reinforcement-learning-based service deployment of 5g mobile edge computing with request-aware scheduling," *IEEE Wirel Commun.*, vol. 27, no. 1, pp. 84–91, 2020.

[12] C. Li, L. Zhu, W. Li, and Y. Luo, "Joint edge caching and dynamic service migration in sdn based mobile edge computing," *J Netw Comput Appl.*, vol. 177, p. 102966, 2021.

[13] C. Pham, N. H. Nguyen, K. K. Nguyen, and M. Cheriet, "Optimized iot service chain implementation in edge cloud platform: A deep learning framework," *IEEE TNSM*, vol. 18, no. 1, pp. 538–551, 2021.

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

[15] J. Chen, Y. Yang, C. Wang, H. Zhang, C. Qiu, and X. Wang, "Multi-task offloading strategy optimization based on directed acyclic graphs for edge computing," *IEEE Internet Things J.*, 2021.

[16] X. Li, X. Wang, Z. Han, and V. C. Leung, "Hierarchical edge caching in device-to-device aided mobile networks: Modeling, optimization, and design," *IEEE J-SAC*, vol. 36, no. 8, pp. 1768–1785, 2018.

[17] A. Vaswani, N. Parmar, J. Uszkoreit, and I. Polosukhin, "Attention is all you need," in *Adv Neural Inf Process Syst.*, 2017, pp. 5998–6008.

[18] T. Haarnoja, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, 2018, pp. 1861–1870.