Deep Reinforcement Learning-based Joint Caching and Computing Edge Service Placement for Sensing-Data-Driven IIoT Applications

Yan Chen*, Yanjing Sun*, Bin Yang† and Tarik Taleb‡
*School of Information and Control Engineering, China University of Mining and Technology, Xuzhou, China
†School of Computer and Information Engineering, Chuzhou University, Chuzhou, China.
‡Faculty of Information Technology and Electrical Engineering, University of Oulu, Oulu, Finland
Email: chyan@cumt.edu.cn, yjsun@cumt.edu.cn, yangbinchi@gmail.com, tarik.taleb@oulu.fi

***Abstract*—Edge computing (EC) is a promising technology to support a variety of performance-sensitive intelligent applications, especially in the Industrial Internet of Things (IIoT). The sensing-data-driven applications whose task processing requires sensing data from various sensors are typical applications in IIoT systems. The placement of caching and computing edge service functions for such applications is vital to ensure system performance and resource utilization in EC-enabled IIoT systems. Therefore, this paper investigates the joint caching and computing edge service placement (JCCESP) for multiple sensing-data-driven IIoT applications in an EC-enabled IIoT system. The JCCESP problem is formulated as a Markov Decision Process (MDP). Then, a deep reinforcement learning (DRL)-based approach is proposed to address the challenges like limited prior knowledge and the heterogeneity of such IIoT systems. Under such an approach, the policy network of the DRL agent is constructed based on an encoder-decoder model to tackle various applications requiring different numbers of service functions. A REINFORCE-based method is further employed to train the policy network. Simulation results indicate that the performances achieved by our proposed approach can converge after training and are significantly superior to benchmarks.**

***Index Terms*—Edge computing, IIoT, sensing-data-driven application, joint service placement, and DRL**

I. Introduction

Edge computing (EC) is emerging technology in Industrial Internet of Things (IIoT) systems to enable various applications for intelligent fabrication by extending the ability of current cloud-based systems to the proximity of applications, which can simultaneously satisfy requirements of both resources and performances [1]. Unlike general IoT applications, many IIoT applications are cooperatively performed by multiple facilities, even several vertical sub-manufacturing systems. Thus, before making operating decisions, the operator of an application must acquire the working conditions of related facilities/sub-systems via mining the sensing data captured by deployed sensors. Meanwhile, these sensing-data-driven IIoT applications usually need to draw inferences from sensing data over specific historical periods, which requires the computing service function (SF) to fetch the necessary sensing data from servers caching the required sensing data before processing an arrived task. Therefore, in EC-enabled IIoT systems, multiple caching service functions (CFs) for caching necessary sensing data are also required by such a sensing-data-driven IIoT application in addition to one SF [2].

IIoT applications generally with strict requirements on the quality of service (QoS) like reliability and latency, which can be satisfied by utilizing EC to cache necessary data and process tasks in the edge system [3]. However, orchestrating the service placement is necessary and vital for ensuring QoS and resource utilization since edge servers (ESs) are resource-limited [4]–[6]. Besides, in an EC-enabled IIoT system, the controller needs further investigate the joint caching and computing edge service placement (JCCESP) for sensing-data-driven IIoT applications. However, the problem has not been studied in current works, and it is challenging when considering the system heterogeneity in ES resource capacity, bandwidth, and application features. Besides, traditional approaches requiring prior global system knowledge may be infeasible since the EC manager may not be authorized to access the knowledge of underlay communication networks. In recent years, deep reinforcement learning (DRL) has been explored to address similar complex problems since the agent can learn from interactions with an unknown system [6]–[8]. Nevertheless, the number of CFs required by each sensing-data-driven IIoT application is heterogeneous, making neural networks with fixed input and output sizes may be unable to work. The encoder-decoder model prevalent in the natural language processing community can handle problems with different length input sequences, where each output size depends on the input. This network model has been combined with DRL in some works to address combination optimization problems like TSP [9] and placement problem [10]. In [11], the service placement for a single easy-to-represent application was investigated by employing this method. However, each sensing-data-driven application needs to be represented by a multidimensional state. Thus, when considering the JCCESP for multiple heterogeneous sensing-data-driven IIoT applications, representing all applications requires a high dimension system state, and the complexity of the problem significantly increases. Besides, resource congestion among applications should be considered when considering multiple applications.

Therefore, this work formulates the problem of JCCESP for multiple heterogeneous sensing-data-driven IIoT applications as a Markov Decision Process (MDP), and a DRL-based approach is introduced. The policy network of the DRL agent is constructed based on the encoder-decoder model. The system state is constructed as a sequence composed of real-time resource consumption conditions of ESs and service functions required by an application to be placed. Then, a REINFORCE-based algorithm is used to train the policy network. Simulation results show that the performance obtained by the DRL-based approach can converge to significantly improved values after training and is superior to benchmarks.

The rest of this work is organized as follows: Section II details the system model and problem formulation. The DRL-based approach is illustrated in Section III. Simulation results are exhibited in Section IV. Section V provides conclusion of this work and discussion of future work.

II. System Model and Problem Formulation

We consider an EC-enabled IIoT system, as shown in Fig.1 where some radio access networks (RANs) are deployed with one ES (h) to support smart IIoT applications, and various sensors (m) are distributed in RANs for monitoring states of the environment and facilities. To support smart manufacturing, multiple sensing-data-driven IIoT applications $\mathbb{A}$ are executed by facilities. Each application $\mathcal{A}_i$ can be represented by a set of required service function including one SF and multiple CFs, i.e., $\mathcal{A}_i = \{\text{SF}, \text{CF}_1, \cdots, \text{CF}_{K_i-1}\}$, where K_i is the number of required services. Moreover, each CF caches and manages the sensing data from one sensor required by $\mathcal{A}_i$. The JCCESP of an application refers to placing its SF and CFs on ESs to satisfy their resource requirements and QoS of the application. We use a binary indicator $\varphi^h_{i,k}=1$ to represent the k^{th} service of $\mathcal{A}_i$ is placed on ES h, and $\varphi^h_{i,k}=0$ otherwise, i.e.,

$$\varphi^h_{i,k} = \{0,1\}, 1 \le \forall i \le |\mathbb{A}|, 1 \le \forall k \le |\mathcal{A}_i|, \forall h \in \mathcal{H}. \quad (1)$$

where $\mathcal{H}$ is the set of ESs in the system. Meanwhile, each service function can only be placed on at most one ES, i.e.,

$$\sum_{h\in\mathcal{H}} \varphi^h_{i,k} \le 1, 1 \le \forall i \le |\mathbb{A}|, 1 \le \forall k \le |\mathcal{A}_i|. \quad (2)$$

After placing a service on an ES, resources are consumed by performing the service. We use $c_{i,k}$ and $s_{i,k}$ to represent the computing and storage resources consumed by the k^{th} service of $\mathcal{A}_i$. The computing and storage resource consumption of all services placed on an ES can be calculated, and they cannot exceed the corresponding resource capacities $(\mathcal{C}^h, \mathcal{S}^h)$, i.e.,

$$\text{C1:} \quad \sum_{i=1}^{|\mathbb{A}|}\sum_{k=1}^{|\mathcal{A}_i|} \varphi^h_{i,k} c_{i,k} \le \mathcal{C}^h, \forall h \in \mathcal{H}, \quad (3)$$

and

$$\text{C2:} \quad \sum_{i=1}^{|\mathbb{A}|}\sum_{k=1}^{|\mathcal{A}_i|} \varphi^h_{i,k} s_{i,k} \le \mathcal{S}^h, \forall h \in \mathcal{H}. \quad (4)$$

After implementing the JCCESP of an application, sensors whose data are required by the application will actively send data to corresponding CFs via routing configured by underlay communication networks. Then, each CF processes the cached data following a pre-configured caching policy and provides requested data when receiving a requisition. As shown in Fig. 1, a computing task is initiated from an initiating facility and then forwarded to the SF placed on an ES. Before task processing, the SF needs to request necessary cached sensing data from related CFs placed on nearby ESs. The task is processed after acquiring all required data. Last, a result is generated and forwarded to the executing facility. We assume that the data transmission routing is configured by the underlay networks and inaccessible by the EC service controller since they may be maintained by different providers. The communications among RANs are established by a set of communication links $\mathcal{L}$. We use $x^l_{i,k}=1$ to indicate that caching the data from a sensor to $(k{-}1)^{th}$ CF of $\mathcal{A}_i$ is transmitted via link l, otherwise $x^l_{i,k}=0$, where $k \in \{2,\cdots,|\mathcal{A}_i|\}$ (i.e., only CFs). We use $y^l_i=1$ and $z^l_i=1$ to respectively indicate the computing tasks and results are transmitted via link l, and values are 0 otherwise. Besides, $\alpha^l_{i,k}=1$ indicates fetching sensing data between SF and the $(k{-}1)^{th}$ CF of $\mathcal{A}_i$ is transmitted via link l. Then, we can obtain the bandwidth consumption of any link l, which cannot exceed its maximum bandwidth capacity $\mathcal{B}_l$, i.e.,

$$\text{C3:} \quad \sum_{i=1}^{|\mathbb{A}|}(y^l_i b^T_i + z^l_i b^R_i + \sum_{k=2}^{|\mathcal{A}_i|}(x^l_{i,k} b^M_{i,k} + \alpha^l_{i,k} b^C_{i,k})) \le \mathcal{B}_l, \forall l \in \mathcal{L}, \quad (5)$$

where b^T_i and b^R_i are the bandwidth consumed by transmitting computing tasks and results of $\mathcal{A}_i$, separately. $b^M_{i,k}$ is the bandwidth consumed by caching data from a sensor to the $(k{-}1)^{th}$ CF of $\mathcal{A}_i$, and $b^C_{i,k}$ is the bandwidth consumed by data acquisition from $(k{-}1)^{th}$ CF to the SF before task processing.

Meanwhile, the EC service latency of an application $\mathcal{A}_i \in \mathbb{A}$ is obtained, which includes the latency used for transmitting computing task and result, latency in requesting and acquiring data from CFs, and task processing latency (d^P_i), i,e.,

$$T_i = \sum_{l\in\mathcal{L}}(y^l_i d^T_i + z^l_i d^R_i) + \max_{k\in\mathcal{K}}\left\{\sum_{l\in\mathcal{L}} \alpha^l_{i,k} d^C_{i,k}\right\} + d^P_i, \quad (6)$$
$$\mathcal{K} = \{2,3,\cdots,|\mathcal{A}_i|\},$$

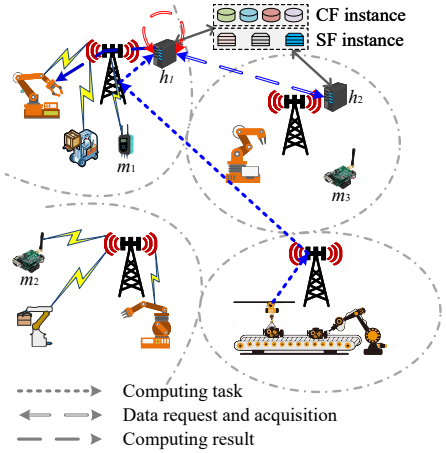

Fig. 1. IIoT system running sensing-data-driven applications

where d_i^T and d_i^T are the one-hop latency for transmitting computing tasks and results of $\mathcal{A}_i$ under required bandwidths. The symbol $d_{i,k}^C$ represents the one-hop latency for acquiring sensing-data from $(k-1)^{th}$ CF of $\mathcal{A}_i$ to the SF. We assume that RANs are close to each other, so the propagation delay is ignored. Meanwhile, we assume that SFs request data with short notification frames and CFs can provide the requested data quickly. To ensure the QoS, the EC service latency of an application ($\mathcal{A}_i$) should satisfy a constraint (τ_i), i.e.,

$$\text{C4:}\ \ T_i \le \tau_i, \forall i \in \{1, 2, \cdots, |\mathbb{A}|\}. \tag{7}$$

Due to the limited resource capacity of ESs and communication links, some applications cannot be satisfied by the EC system. Meanwhile, unsuitable service placement may break an application's latency constraint. We define an application as an accepted application if its all services are placed on ESs without breaking any constraint, indicated by a binary indicator $\psi_i = 1$. Otherwise, the application is rejected by the system and represented by $\psi_i = 0$. Then, ψ_i is determined by the JCCESP result of $\mathcal{A}_i$, i.e.,

$$\psi_i = \begin{cases} 1, & \text{if } \sum_{k=1}^{|\mathcal{A}_i|} \sum_{h \in \mathcal{H}} \varphi_{i,k}^h = |\mathcal{A}_i| \\ 0, & \text{otherwise} \end{cases}. \tag{8}$$

Moreover, ψ_i relies on the JCCESP action (a_i) and the result of implementation the action in the IIoT system. If a_i is implemented without breaking any constraint, the JCCESP policy is maintained, i.e., $\sum_{h \in \mathcal{H}} \varphi_{i,k}^h = 1, \forall k \in \{1, \cdots, |\mathcal{A}_i|\}$. Otherwise, the application is rejected and the policy is cleared, i.e., $\sum_{h \in \mathcal{H}} \varphi_{i,k}^h = 0, \forall k \in \{1, \cdots, |\mathcal{A}_i|\}$.

From the perspective of edge service providers, their primary objective is maximizing the reward from providing high-performance services to accepted applications. Besides, they generally want to take full utilization of resources to accept more applications. Thus, we define the objective as

$$\begin{aligned} \text{P1:}\ \ & \max \sum_{i=1}^{|\mathbb{A}|} (\psi_i \beta W_i \frac{\tau_i}{T_i} + \psi_i) \\ & subject\ to: \text{C1, C2, C3, C4,} \end{aligned} \tag{9}$$

where W_i is a weight factor that can represent the importance or value of an application $\mathcal{A}_i$ in actual systems. The left part of the reward is obtained by providing a low latency service, which is increased when the service latency is reduced. The right side is an incentive reward obtained from accepting an application. β is a factor to balance the two parts flexibly.

III. DRL-BASED JCCESP APPROACH

This section details the employed DRL-based JCCESP approach for multiple sensing-data-driven IIoT applications.

A. Framework of DRL-based approach

The number of CFs required by applications is heterogeneous, and application features like resource requirements are also heterogeneous, making it hard to represent the service requirement of all applications in a simple format. Meanwhile, the JCCESP of applications interferes with each other, making it hard to decide which applications should be rejected when

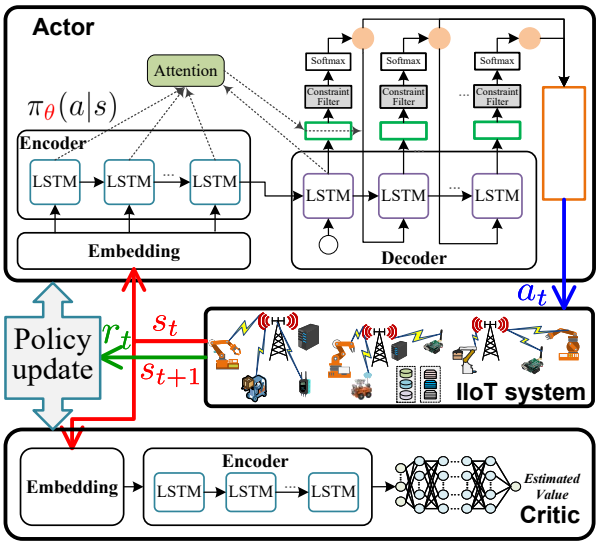

Fig. 2. The training architecture of the DRL-based JCCESP approach for multiple sensing-data-driven IIoT applications

breaking constraints after implementing JCCESP for all applications at once. For example, the resources freed by rejected applications can be re-utilized to satisfy other applications, requiring re-designing the JCCESP action, and the problem may repeat. Therefore, we set multiple IIoT applications to be processed one by one, and the order is based on its arrival to maintain fairness. At each time, the edge service controller makes a JCCESP a_t for an application $\mathcal{A}_t$ based on observed system state s_t. Then, the IIoT system implements a_t, and a reward r_t can be obtained. Meanwhile, the system evolves into the next state s_{t+1}. Since the action at each step is made only based on the current state, and the state transition depends on the current state and employed action, the JCCESP process of multiple IIoT applications is an MDP. Then, DRL can be employed for such problems in heterogeneous IIoT systems with limited prior knowledge because it aims at learning a policy that can maximize the expected accumulated reward of an MDP. Fig. 2 shows the training framework of the DRL-based approach. Besides, only the actor works when the agent is fully trained and implemented in the system (i.e., execution phase). Moreover, the training process can be re-triggered when system requirements change or observing a significant system QoS degradation. The state, action, and reward are defined as follows.

State: The system state consists of two parts: the resource capacities and real-time observed consumption condition of all ESs ($s_t^e = [\mathcal{C}^h, \mathcal{S}^h, C_{h,t}^o, S_{h,t}^o]_{4 \times |\mathcal{H}|}$), and the service sequence ($s_t^a = [\text{SF}_t, \text{CF}_{1,t}, \text{CF}_{2,t}, \cdots]_{6 \times (|\mathcal{A}_t|+1)}$) of the application to be placed ($\mathcal{A}_t$). The state of each SF_t is represented by two columns, which include the b_t^T, b_t^R, $d_t^T, d_t^R, d_t^P, W_t, \tau_t, s_{t,1}, c_{t,1}$ as well as the indexes of RANs where the initiator and executor of the application are located. The state of $\text{CF}_{k-1,t}$ is represented by one column including $b_{t,k}^M, b_{t,k}^C, d_{t,k}^C, c_{t,k}, s_{t,k}$ and the index of each related sensor.

$$s_t = \{s_t^e, s_t^a\}. \tag{10}$$

Action: Action is an one-dimensional sequence of the same

length as the input state, where each element is the index of an ES that the corresponding service will be placed on it. The resource condition of ESs in the input state sequence is treated as virtual services. Thus, the action generated by the DRL agent can be represented by

$$a_t = [a_t^k]_{1\times(|\mathcal{H}|+|\mathcal{A}_t|+1)}. \tag{11}$$

Then, the system will select the part that indicates the JCCESP of the application's actual services and implement it. The SF is represented by two columns since it contains more features.

Reward: An immediate reward r_t is obtained based on if the system accepts the application after executing the JCCESP action selected from a_t in the system. We calculate the r_t as

$$r_t = \begin{cases} \beta w_i \frac{\tau_t}{T_t} + \frac{t}{|\mathbb{A}|}, & \text{if } \mathcal{A}_t \text{ is accepted} \\ 0, & \text{otherwise} \end{cases}, \tag{12}$$

where $w_i = \frac{W_i}{\sum_{j=1}^{|\mathbb{A}|} W_j}$ is the normalized weight value.

B. DRL-based JCCESP approach

The JCCESP action for each application is the combination of actions for all of its services, resulting in a vast discrete action space for every application. Thus, the value-based DRL methods (e.g., deep Q-learning) in which action is selected by comparing values of all candidate actions are unsuitable for such problems as estimating and comparing values of all JCCESP candidate actions is difficult. Thus, policy-based RL algorithms have been explored to address continuous and large-scale state and action space problems.

As shown in Fig. 2, the actor network is responsible for generating the JCCESP action for each input application. The process is detailed in Algorithm 1. Each system state sequence is composed of an application service sequence and the real-time resource condition of ESs, which is first passed by an embedding layer and the encoder. Then, a final hidden state ($\mathbb{Y}_0$) is generated, and hidden states ($\mathbb{H}_t$) at each encoding step (i.e., each element in the sequence) are also obtained (step 5). Then, the outputs will be used in decoding steps recurrently processed by a long short-term memory (LSTM) layer. The LSTM layer can be seen as a function of two inputs: the LSTM input and the hidden state obtained from the previous step, i.e.,

$$\mathbb{Y}_j = \text{LSTM}(\mathbb{Y}_{j-1}, (\rho_{j-1}, \mathbb{C}_{j-1})) \tag{13}$$

The LSTM input tuple $(\rho_{j-1}, \mathbb{C}_{j-1})$ is the contacted array of action selected in previous decoding step and a context vector generated by attention mechanism [12], i.e.,

$$\mathbb{C}_j = \text{Attention}(\mathbb{Y}_j, \mathbb{H}_t). \tag{14}$$

The input of the first decoding step (i.e., $(\rho_0, \mathbb{C}_0)$) is filled with trainable parameters (step 6). A linear layer (green rectangle in Fig. 2) whose output size is equal to $|\mathcal{H}|$ is employed to get the action of placing a service. The linear layer processes the output hidden state ($\mathbb{Y}_j$) and the obtained context ($\mathbb{C}_j$) to get a set of score values ($\mathbb{P}_j$). For simplification, we use a map function f^D to represent the above steps, i.e.,

$$(\mathbb{Y}_j, \mathbb{P}_j) = f^D(\mathbb{Y}_{j-1}, \rho_{j-1}, \mathbb{H}_t). \tag{15}$$

Besides, a constraint filter is added after the linear layer to avoid selecting actions that will break the resource constraint of ESs (step 8∼step 14). If the estimated resource consumption exceeds the capacity of an ES after adding the resource requirements of the service, the corresponding location value of $\mathbb{P}_j$ is set to be $-\infty$. Then, passing the modified $\mathbb{P}_j$ through a softmax function, the probabilities of selecting each ES $\mathbb{P}_j$ to place the service are obtained, in which the probability of selecting an ES that will break resource constraint is 0. Then, sampling from these probabilities gives the action of placing the service (step 15).

Algorithm 1 JCCESP for one sensing-data-driven application

Input: $s_t^e \leftarrow$ Observed resource state of ESs
$s_t^a \leftarrow$ Services of the application to be placed
Output: JCCESP action a_t and corresponding probabilities
1: $(\boldsymbol{\mathcal{C}^h}, \boldsymbol{\mathcal{S}^h}) \leftarrow$ Resource capacities of all ESs
2: $(\mathbf{C}^{\mathbf{o}}_{|\mathcal{H}|}, \mathbf{S}^{\mathbf{o}}_{|\mathcal{H}|}) \leftarrow$ Estimated resource consumption
3: $s_t = \{s_t^e, s_t^a\}$ State inputted to the agent
4: $K_t \leftarrow$ length of the s_t sequence
5: $(\mathbb{Y}_0, \mathbb{H}_t)$ = Encoder(Embedding(s_t))
6: $(\rho_0, \mathbb{C}_0) \leftarrow$ Trainable start input of Decoder
7: **for** $j = 1, 2, \cdots, K_t$ **do**
8: $(c_j, s_j) \leftarrow$ Resources required by j^th service
9: $(\mathbb{Y}_j, \mathbb{P}_j) = f^D(\mathbb{Y}_{j-1}, \rho_{j-1}, \mathbb{H}_t)$
10: $\mathbb{P}_j \leftarrow [p_1, p_2, p_{|\mathcal{H}|}]$
11: $\mathbf{g} = [(\mathbf{C}^{\mathbf{o}}_{|\mathcal{H}|} + c_j) > \boldsymbol{\mathcal{C}^h}] + [(\mathbf{S}^{\mathbf{o}}_{|\mathcal{H}|} + s_j) > \boldsymbol{\mathcal{S}^h}]$
12: $\mathbb{P}_j = \mathbb{P}_j(1 - \mathbf{g}) + (\mathbb{P}_j - \infty)\mathbf{g}$
13: $\mathbf{P}_j$ =softmax($\mathbb{P}_j$)
14: $a_t^j =$ action_sample ($\mathbf{P}_j$)
15: Update $(\mathbf{C}^{\mathbf{o}}_{|\mathcal{H}|}, \mathbf{S}^{\mathbf{o}}_{|\mathcal{H}|})$
16: $a_t = [a_t^1, a_t^2, \cdots, a_t^{K_t}]$, $\mathcal{P}_t = [\mathbf{P}_1, \mathbf{P}_2, \cdots, \mathbf{P}_{K_t}]$

Since the JCCESP process for multiple sensing-data-driven IIoT applications is an MDP with finite steps, we employ a REINFORCE-based algorithm to train the policy network. The JCCESP MDP trajectory can be represented by a sequence of state, action and reward tuples, i.e., $(s_1, a_1, r_1), \cdots, (s_t, a_t, r_t), \cdots, (s_d, a_d, r_d)$. Note that (s_d, a_d, r_d) denotes that the system reaches a terminal state. In this work, we set the system to reach the done state once an application is rejected due to breaking any constraint. Because a better controller can place more applications before breaking any constraint. Moreover, we assume the IIoT system will refuse to execute subsequent actions since they may also be irresponsible actions that damage other parts of the system.

The accumulated reward of the JCCESP of multiple sensing-data-driven IIoT applications is the sum of the immediate reward of all steps in the trajectory from an initial step, i.e., $R(s_t) = \sum_{\mu=t}^{d} r_\mu$. For any given state, the object of DRL is to find a policy to maximize the expected accumulated reward.

$$\mathcal{J}_\theta = E_{\{s\sim\mathcal{S},\pi_\theta\}}[R(s)] \tag{16}$$

where $\mathcal{S}$ represents the state space of the system. π_θ is the policy of the edge service controller with parameters θ. Then, according to REINFORCE algorithm, the policy can be updated by the policy gradient method [13], i.e.,

$$\nabla\mathcal{J}_\theta = E_{\{s\sim\mathcal{S},\pi_\theta\}}[(R(s) - \widetilde{v}(s))\nabla_\theta \log(p_\theta(a|s))], \tag{17}$$

where $\widetilde{v}(s)$ represent state values estimated by an auxiliary critic network (as shown in Fig. 2) to improve the stability of the training process. In actual implementations, the accumulated reward generally employs a discounted format, i.e.,

$$G_t = \sum_{\mu=t}^{\infty} \gamma^{\mu-t} r_\mu. \tag{18}$$

The detailed training process is shown in Algorithm 2, in which the probability of selecting an action is calculated by the probability chain rule (step 14).

Algorithm 2 Training of JCCESP based on REINFORCE

Input: Actor network (θ), Critic network (θ_v), IIoT system
Output: Trained JCCESP policy networks.

1: **for** Episode = $1, 2, \cdots$, n_episodes **do**
2: System initialization
3: $\mathbb{A} \leftarrow$ IIoT applications to be served
4: **for** $t = 1, 2, \cdots, |\mathbb{A}|$ **do**
5: $s_t^e \leftarrow$ Observed resource state of ESs
6: $s_t^a \leftarrow$ Services of t^{th} application in $\mathbb{A}$
7: $\widetilde{v}_t =$ Critic(s_t^e, s_t^a)
8: Obtain a_t and $\mathcal{P}_t$ by performing **Algorithm** 1
9: $r_t \leftarrow$ Reward obtained by implementing a_t
10: **if** $r_t == 0$ **then**
11: **break**
12: **else**
13: Update the system state
14: $p_\theta(a_t|s_t) = \prod_{j=1}^{K_t} \mathcal{P}_t(j, a_t^j)$
15: Calculate discount rewards $[G_1, G_2, \cdots]$
16: $\mathcal{J}_\theta = -\mathbb{E}_{t\sim|\mathbb{A}|}[(G_t - \widetilde{v}_t)\log(p_\theta(a_t|s_t))]$
17: $\mathcal{J}_{\theta_v} = \mathbb{E}_{t\sim|\mathbb{A}|}[(G_t - \widetilde{v}_t)^2]$
18: Update θ of actor by Adam$(\nabla_\theta(\mathcal{J}_\theta))$
19: Update θ_v of critic by Adam$(\nabla_{\theta_v}(\mathcal{J}_{\theta_v}))$

IV. Performance Evaluation

Simulations are conducted based on Pytorch 1.8 environment to evaluate the performance of the DRL-based approach, and we compare it with two benchmarks: the random policy and the load-least algorithms. The random approach randomly selects an ES for each service in each application, and the load-least algorithm selects the ES with the least real-time computing load to place a service. The computing load on an ES is defined as the maximum ratio of the computation and storage resource consumption to the corresponding maximum resource capacities of the ES. Meanwhile, the constraint filter is also employed in the random method to postpone breaking the resource constraint of ESs.

We randomly select 7 RANs in the network to deploy an ES, where the resource capacities and locations (i.e., $\mathcal{C}^h, \mathcal{S}^h$, and associated RANs) of ESs are randomly generated. For simplicity, computing and storage resources are normalized values, representing the number of resource blocks that ES can provide or required by each service function. Meanwhile, we set SFs requiring more computing resources but fewer storage resources than CFs because SF mainly undertakes task computing and CF mainly undertakes data caching. Besides, computing tasks are generally large volumes of data and require significant processing time. An online network topology is employed[1], and the routing of the underlay communication network is configured by the Dijkstra method. Detailed information is shown in Table I. The learning rate of actor and critic are separately set to be 0.001 and 0.002 at first and reduces to 0.1 times of initial value after 3000 episodes of training to improve stability. The discount factor (γ) used in (18) is 0.9.

TABLE I
Properties of Edge Servers

ES	0	1	2	3	4	5	6
RAN	22	10	4	19	32	2	20
$\mathcal{C}^h$	970	764	766	843	761	602	791
$\mathcal{S}^h$	521	600	949	573	617	729	559

Besides, 50 sensors are randomly placed in the system. Then, at each training episode, we generate 40 applications. The task processing of every application relies on sensing data from multiple randomly selected sensors (i.e., CFs). Besides, the parameters of every SF and CF are random generated integer values based on parameters detailed in Table II.

TABLE II
Simulation Parameters

Parameter	Value	Parameter	Value
$\lvert\mathcal{K}\rvert$	[3, 10]	W_i	[1, 5]
τ_i	[300, 400] ms	d_i^P	[50, 80] ms
d_i^T	[15, 20] ms	d_i^R	[2, 10] ms
$d_{i,k}^C$	[10, 15] ms	$b_{i,k}^C$	[5, 10] Mbps
b_i^T	[20, 50] Mbps	b_i^R	[2, 8] Mbps
$b_{i,k}^M$	[10, 20] Mbps	LSTM layer	3
$c_{i,1}$	[5,15]×10	$s_{i,1}$	[2, 8]
$c_{i,k}, (k>1)$	[2, 8]	$s_{i,k}, (k>1)$	[2, 8]×3
Embedding size	128	Hidden size	64

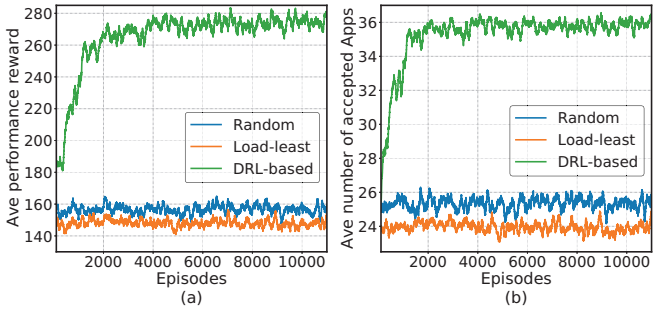

Fig. 3. Training performance. (a) Average latency performance reward. (b) Average number of accepted applications.

First, we test the convergence of the DRL-based approach. We statistic the number of applications (Apps) accepted by the IIoT system before breaking any constraint in each training episode. The performance reward is obtained from accepted applications and is calculated by the left part of (9) in each training episode. The results during the training process are processed by the simple moving average method over every 100 episodes. From Fig. 3, we can observe that DRL-based algorithms can achieve significant performance improvement

[1]http://www.topology-zoo.org/files/Bics.gml

and performance convergence after training. The load-least approach does not perform better than the random approach with constraint filter because it only considers limited knowledge about ESs' resource consumption. After training, the DRL-based approach can accept an average of about 36 applications, much higher than 24 and 26 applications separately obtained by the load-least approach and the random method. Meanwhile, the results of performance reward present the same phenomenon. The results reveal that the DRL-based approach can accept more applications before breaking any constraint, and accepted applications are performed with low latency since higher rewards are achieved.

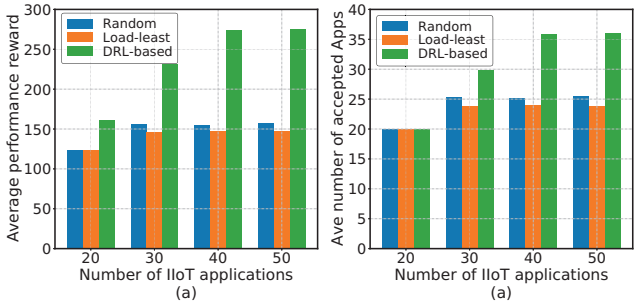

Fig. 4. Execution performance. (a) Average latency performance reward. (b) Average number of accepted applications.

We also test the execution performance of the trained policy after finishing the training phase, and the result is shown in Fig. 4, where we generate different numbers of applications at each time with a different random seed. We can find that the number of accepted applications and achieved performance rewards significantly increased as the number of IIoT applications increased when employing the trained DRL-based policy. When the number of applications is 20, the system can accept almost all applications, while the DRL agent can help achieve a higher performance reward. The benchmarks achieve similar performance results when more than 30 IIoT applications, i.e., cannot achieve higher rewards and accept more applications when the number of applications increases. The trained DRL-based policy can also accept almost all applications when there are 30 IIoT applications. When the number of applications is 50, the DRL agent achieves similar performance as the condition of 40 IIoT applications because the system can only accept finite applications due to limited resource capacities.

V. Conclusion and Discussion

This work investigates the JCCESP problem for multiple sensing-data-driven IIoT applications, where each application requires a sequence of edge service functions, including one SF and multiple CFs. Then, considering the limited prior knowledge of the underlay network and complexity resulting from the heterogeneity of applications, the problem is formulated as an MDP, and a DRL-based approach is proposed. The policy network employs an encoder-decoder architecture to cope with the issue of different lengths of application service sequence and the requirement that the length of an output action depends on the length of the input state. After that, a REINFORCE-based architecture is employed to train the policy network. Simulation results show the effectiveness of the DRL-based approach, where the performance achieved by the DRL-based approach converged to significantly improved results after training compared to benchmarks.

In the future expansion of this work, more advanced off-policy DRL approaches like DDPG, TD3 and SAC can be explored to improve the training stability and exploration. Because REINFORCE-based on-policy DRL algorithms can only update parameters with the latest experiences after completing an episode, which results in low experience utilization and sensitivity to environment and hyperparameters.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (No. 62071472), the Program for 'Industrial IoT and Emergency Collaboration' Innovative Research Team in CUMT (No. 2020ZY002), the Fundamental Research Funds for the Central Universities (No. 2020ZDPY0304), the Chinese Government Scholarship (CSC202006420096), the Academy of Finland Projects: 6Genesis (No. 318927) and IDEA-MILL (No. 335936)

References

[1] L. Chen, C. Shen, P. Zhou, and J. Xu, "Collaborative service placement for edge computing in dense small cell networks," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 377–390, 2021.

[2] A. Ndikumana, N. H. Tran, T. M. Ho, Z. Han, W. Saad, D. Niyato, and C. S. Hong, "Joint communication, computation, caching, and control in big data multi-access edge computing," *IEEE Transactions on Mobile Computing*, vol. 19, no. 6, pp. 1359–1374, 2020.

[3] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, "Survey on multi-access edge computing for internet of things realization," *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, pp. 2961–2991, 2018.

[4] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas,"Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 10–18.

[5] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 514–522.

[6] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1409–1434, 2019.

[7] S. Wang and T. Lv, "Deep reinforcement learning for demand-aware joint vnf placement-and-routing," in *2019 IEEE Globecom Workshops (GC Wkshps)*, 2019, pp. 1–6.

[8] G. Li, H. Zhou, B. Feng, Y. Zhang, and S. Yu, "Efficient provision of service function chains in overlay networks using reinforcement learning," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2019.

[9] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, vol. 28. Curran Associates, Inc., 2015, pp. 2692–2700.

[10] A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, and J. Dean, "Device placement optimization with reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 2430–2439.

[11] R. Solozabal, J. Ceberio, A. Sanchoyerto, L. Zabala, B. Blanco, and F. Liberal, "Virtual network function placement optimization with deep reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 292–303, 2020.

[12] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, United States, May 2015.

[13] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, p. 229–256, 1992.