Customer Centric Service Caching for Intelligent Cyber-Physical Transportation Systems with Cloud-Edge Computing Leveraging Digital Twins

Hanzhi Yan, Xiaolong Xu*, *Senior Member, IEEE* , Muhammad Bilal, *Senior Member, IEEE* , Xiaoyu Xia, Wanchun Dou, Huihui Wang

***Abstract*—To provide various high-quality intelligent transportation services to customers, Intelligent Cyber-Physical Transportation Systems (ICTS) with cloud-edge computing are widely commissioned. In such ICTS, service requests processed by edge servers (ES) usually have a low response latency, thus leading to a high quality of service (QoS). As a prerequisite for requests processing in the ES, service cache provides requests with storage and computing resources. But the limited resources of each ES make it impossible to cache all services, so how to generate high-performance caching strategies for ICTS is a major challenge. Besides, how to evaluate the effectiveness of the application of these strategies is also a challenge. Fortunately, thanks to the constructed digital twins (DT) for ICTS, the strategies have a digital platform to be simulated into application. With the assistance of DT, a solution to service caching problem in ICTS, named SFT-SCAR, is proposed. Firstly, a DT supported service providing framework for ICTS is designed. Then, a graph attention network (GAT) based service request flow prediction scheme and an asynchronous advantage actor-critic (A3C) based service caching scheme are presented. Besides, the generated caching strategies are simulated in the DT of ICTS to evaluate the performance of these strategies. Experimental results demonstrate that the proposed SFT-SCAR approach improves the hit rate by 1.11% - 13.27%.**

***Index Terms*—Intelligent Cyber-Physical Transportation Systems, cloud-edge computing, digital twins, graph attention network, asynchronous advantage actor-critic**

I. Introduction

BENEFITING from advancements of the Artificial Intelligence (AI) and Internet of Things (IoT), the development of Intelligent Cyber-Physical Transportation Systems (ICTS) has drawn widespread concern in industry and academia. As AI and IoT both need data support, the communication in ICTS is a critical point that is worth attention. Currently, the data transmission in ICTS mainly relies on 5G. With the advantages of 5G, such as ultra-high data rate, low latency, and massive connections, vehicles, pedestrians, smart sensors, and service providers are connected to mobile networks (MNs), enabling the Internet of Everything [1]. MNs have become an important part of ICTS, and have supported many ICTS services, such as route planning, collision warning, and autonomous driving. Some of these ICTS services have critical demands on the storage and computing resources [2]. Typically, customers have little resource on their own devices, which makes it a challenge to process service requests locally, especially for some compute-intensive services. Mobile cloud computing (MCC) is a common paradigm for solving this problem. With MCC, the service requests of customers are sent through the MN to the cloud server has massive computing and storage resources in ICTS. Then the requests are processed in the cloud, and finally the processed results are returned to customers.

The proliferation of service requests from customers pose transmission bandwidth and latency challenges for the quality of latency-sensitive services in MCC. For example, autonomous driving requires real-time analysis of the surrounding environment data collected by sensors and then making decisions [3]. While MCC often needs long-distance signal propagation, so it is hard to ensure that the latency-sensitive service requests are processed within the specified time, thus impairing the customers' quality of service (QoS). Timeouts in processing service requests, such as the collision warning request, may even result in life-threatening situations. In response to these challenges, the Mobile Edge Computing (MEC) paradigm is introduced to reduce latency and enhance service capabilities through configurating edge devices with storage and computing resources near customers [4], [5]. Compared with MCC, MEC provides services by edge servers (ES) closer to customers, which reduces service latency, improving the QoS [6]. The introduction of MEC into ICTS to assist MCC in providing services (ICTS in cloud-edge computing) significantly alleviates the network burden caused by large-scale data transmission, and better supports compute-intensive and latency-sensitive ICTS services, it also provides space for better customer privacy protection mechanisms [7], [8]. As a prerequisite for requests processing in the ES, service cache provides requests from customers with the necessary resources. The ES only processes the request which is corresponding to a cached service. If not cached, the requests for this service will be handled by the cloud computing platform (CCP).

Since the difficulty of directly observing the cache condition

Hanzhi Yan is with the School of Computer Science, Nanjing University of Information Science and Technology, Nanjing 210044, China. E-mail: yanhz.nuist@gmail.com.

Xiaolong Xu is with the School of Software, and Jiangsu Collaborative Innovation Center of Atmospheric Environment and Equipment Technology (CICAEET), Nanjing University of Information Science and Technology, Nanjing 210044, China. E-mail: xlxu@ieee.org.

Muhammad Bilal is with the School of Computing and Communications, Lancaster University, Lancaster LA1 4YW, United Kingdom. E-mail: m.bilal@ieee.org.

Xiaoyu Xia is with the School of Computing Technologies, RMIT University, Melbourne, Victoria, Australia, e-mail: xiaoyu.xia@rmit.edu.au.

Wanchun Dou is with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. E-mail: douwc@nju.edu.cn.

Huihui Wang is with St. Bonaventure University, St. Bonaventure, NY 14778, USA. E-mail: hwang@sbu.edu

* Xiaolong Xu is the corresponding author of this paper.

Manuscript received; revised.

and updating the service caching strategies of ESs in the real world, and the potential for losses (especially to customers) incurred from deploying low performance service caching strategies, the digital twins (DT) of ICTS are constructed to assist in observing and controlling. DT, as an emerging digital technology, by modeling the physical world with a network model, not only digitizes scenarios in real time, but also predicts and optimizes the behavior of production systems and its components [9]. Nowadays, MNs are being deployed on a large scale and are more intensively, so the small changes in the MNs have a cascading effect in a short period of time [10]. Constructing DT can simulate different and complex scenarios and test solutions, contributing to the success of MNs in 6G and beyond [11]. Simulating real scenarios using DT generally relies on a wealth of information from the physical world. With the help of MN, the real world and DT are able to transfer data to each other, enabling the interaction between the physical and virtual worlds [12], [13]. Therefore, DT and MN are used in collaboration to provide powerful support for different applications in various sectors [14]. For example, DT assists in predicting and evaluating network events, conducting tests and providing network updates. Besides, DT facilitates the implementation of more powerful functions and services, including the embedding of AI in the MNs. In the ICTS, DT is used to simulate the solution and observe the influence on each road section and area by collecting real data such as traffic flow and service request flow. For example, integrating traffic infrastructure into the DT to analyze the efficiency of traffic infrastructure by predicting the traffic flow based on AI models, thus enabling better resource scheduling [15]. Therefore, the performance of the service caching can be evaluated with the assistance of DT to avoid losses to consumers in real scenarios caused by deploying low performance service caching strategies.

However, with the increasing types of the ICTS service, the resource requirements (e.g., storage space, CPU, RAM) and real-time requirements of services are different. In addition, the limited resources of ESs makes it impossible to cache all the services, and in ICTS, the service request flow is constantly changing caused by the dynamic customer location, which brings about the service requests hard to predict. Therefore, it is difficult to adjust the service caching strategy flexibly in such a multi-constrained and dynamic environment [16]. Responding to this challenge, a solution to the service caching problem in ICTS, named SFT-SCAR, is proposed. SFT-SCAR consists of two main modules, the graph attention network (GAT) based service request flow prediction scheme and the asynchronous advantage actor-critic (A3C) based service caching scheme. The contributions are listed as follows.

- Design a service providing framework for the ICTS with DT. Adopt the data collected by ESs to build the DT of ICTS in the CCP. DT is used to observe the influence on each road section and area by collecting real data such as traffic flow and service request flow.
- Construct a graph neural network structure, and design a customer service request flow prediction scheme based on GAT (SFT). In SFT, by calculating the attention coefficient between the current node and its neighbors, the attention coefficients are weighted for calculating the output features of the current node.
- To update the cache state in real time, a customer centric service caching scheme based on A3C (SCAR) is proposed. The service caching strategy for ICTS in each time interval is generated by SCAR. In SCAR, multiple agents are adopted to interact with the environment, increasing the speed of data acquisition and improve training efficiency.
- Simulate caching strategies in DT to verify the effectiveness of SFT-SCAR in improving cache hit rate and reducing request response latency.

Other content of the paper includes: In Section II, we discuss the related work. In Section III, we design the system model, and define the problem. In Section IV, we introduce the service request flow prediction scheme SFT. In Section V, we describe the service caching scheme SCAR. In Section VI, we analyze the effectiveness of SFT-SCAR. Finally, in Section VII, conclusion of this research is illustrated, which involves the possible future works.

II. Related Work

The related work is summarised in terms of three key points, including A. applications of edge computing in ICTS, B. researches on service caching in MEC and C. functions of digital twins in ICTS.

A. Applications of Edge Computing in ICTS

In recent years, Telematics is widely researched, as Telematics has high network requirements and needs to achieve low latency, good stability and other requirements. Since traditional cloud computing causes long data transmission time in the link and thus high latency, edge computing chooses to place computing resources closer to the vehicle, which largely reduces latency and improve system stability.

The application of edge computing in ICTS has been extensively researched by many scholars. Lv et al. [17] introduced edge computing to Telematics and combined it with techniques such as artificial intelligence to construct a model for task offloading, effectually reducing the time and energy costs caused by task offloading and improving task offloading efficiency. Ning et al. [18] empower edge intelligence in Telematics, propose a framework for optimizing computational offloading as well as content caching within Telematics, and use a lightweight real-time processing framework based on simulation learning, which can effectively reduce network latency and improve offloading and caching efficiency. Li et al. [19] have built a framework with software-defined networking and MEC as a solution to solve the problem of placing ESs with controllers, which can effectively reduce service latency and improve load balance. Wang et al. [20] design an architecture for ICTS and propose a strategy which can realize offload computation and allocate resource jointly. The tasks of vehicles are directly offloaded to edge devices in MEC for executing, thus reducing the data transmission latency.

B. Researches on Service Caching in MEC

Service caching has attracted a lot of attention as it has shown great potential to reduce service latency in MEC. There are already some researches on service caching. Given the limited storage space on ESs, the question of which tasks are cached to the server is particularly important.

To solve the complex problem of which programs in the service caching require long-term caching, Su et al. [21] have designed a single ES to help customers in performing computational tasks. Also, they have transformed this problem into a mixed-integer nonlinear programming problem that simplifies this caching problem and decreases computational time and consumption. Zhao et al. [22] proposed an online algorithm that dynamically computes whether a service needs to be cached in the server, rather than predicting in advance. Ren et al. [23] have transformed the problem of service caching into a Markov-game and designed an deep reinforcement learning based algorithm which is online and is able to optimize the process of determining whether a service needs to be cached. Some recent work has focused on reducing the average service response time in service caching. Pham et al. [24] have described the utility problem in service caching as an integer nonlinear programming problem and proposed a QOE-based utility optimization method to solve the joint service caching problem in MEC systems, which can effectively utilize computational resources to reduce the request response latency. Ma et al. [25] have investigated cooperation among edge nodes and workload scheduling in MEC. Besides an algorithm for cooperative service caching is proposed to reduce the request response latency.

C. Functions of Digital Twins in ICTS

DT can be used within ICTS to map physical reality such as vehicles into virtual space by collecting historical as well as current information from physical devices such as vehicles and roadside units, thus making it easier to track and evaluate devices.

In recent years, there has been extensive research on how DT can be applied to ICTS with MEC. Sun et al. [26] introduced the DT in a UAV-assisted vehicular network to collect data from vehicles as well as RSUs within the ICTS in real-time, using a two-stage incentive mechanism, and thus achieved efficient unified resource scheduling in ICTS, further improving resource utilization efficiency and reducing energy consumption. Zheng et al. [27] introduced the DT in the study of service offloading within the ICTS, using sensor data to establish DTs corresponding to real vehicles, sharing information among them so that vehicles can easily obtain global information, which can better predict the possible arrival of services in the future period and leave sufficient computational resources for them. Vehicles within traditional Telematics can only interact with each other with information from neighboring vehicles, and the degree of resource sharing is limited. To address this drawback, Tan et al. [28] use DTs in the ICTS to enable the virtual vehicles to communicate with each other regardless of distance, and real vehicles can communicate with their DTs thus enabling real vehicles to interact with each other over long distances. Considering the contradiction between the strong computational power required for resource allocation and the insufficient computational power of ESs in ICTS, Liu et al. [29] used DT technology to rationalize the global resource allocation and thus optimize the quality of service. Zhang et al. [30] proposed a new in-vehicle edge network, in which proposed a new in-vehicle edge network, in which the the vehicles are aggregated according to the gravity in DT, divide the complex in-vehicle network into simpler parts, and then use the multi-intelligence learning method to allocate the edge resource based on vehicle aggregation. In addition the system was verified to effectively improve the task offloading efficiency.

III. System Model

This section contains three parts. Firstly, a framework of service providing in ICTS with DT is introduced. Then, the caching model and the cost model are described. Finally, we formulate the service caching problem with multiple constraints.

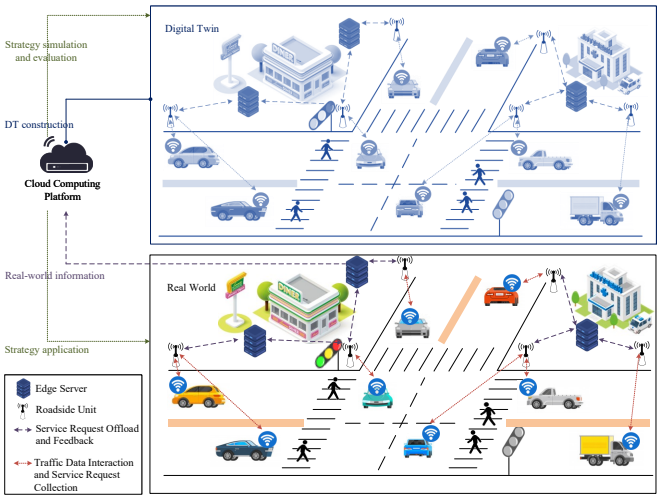

Fig. 1. A service providing framework for ICTS with DT

A. System Framework

The service providing framework with RSUs and ESs for ICTS is shown in Fig.1. The RSUs denoted by $U = \{u_1, u_2, ..., u_M\}$ are placed close to the roads to interact with vehicles. In the ICTS, RSUs collect the service requests which are generated by vehicles (i.e., requests from on-board customers). u_i has three parameters, latitude, longitude and the collected service requests, so it is represented as $u_i(lat_i, lot_i, R_i), u_i \in U$. R_i is a set of service requests, $R_i = \{r_{i,1}, r_{i,2}, \ldots, r_{i,K_i}\}$, where $r_{i,j}$ is a service request collected by u_i, K_i is the amount of service requests in u_i. Typically, most of the services in ICTS are time sensitive (e.g., crash warning and congestion detection). Hence the ESs denoted by $E = \{e_1, e_2, ..., e_N\}$ are arranged in traffic-intensive areas to process service requests. e_i has four parameters, latitude, longitude, the set of cached services and the set of service requests from the RSUs in the zone of e_i, so

it is represented as $e_i(lat_i, lot_i, c_i, \mathcal{R}_i)$. The service requests that are not processed on ESs will be offloaded to the CCP for centralized processing. The CCP has enough storage and computing power to perform all services and manages the resources of ESs. In addition, the DT of the ICTS is built on the CCP to realize real-time data visualization. With the help of DT, the request flow prediction and the caching strategy simulation are implemented more easily.

B. Service Caching Model

For ESs, the ICTS services are divided into two categories, which are services that *must be cached* (e.g., crash warnings) and services that *need to be cached* (e.g., path planning). The caching model here is for the services *need to be cached*. Even if these services are not cached in ESs, the QoS will not be seriously affected. At the time interval t, a service request collected by u_i (the request is directed to a service that need to be cached) is represented as $r_{i,j}^t(r_{i,j}^t \in R_i)$. $Q(r_{i,j}^t, e_q)$ indicates whether the service request $r_{i,j}^t$ should be sent to the ES e_q, measured by

$$Q(r_{i,j}^t, e_q) = \begin{cases} 1, & r_{i,j}^t \in \mathcal{R}_q \\ 0, & otherwise \end{cases}. \tag{1}$$

The service requests are directed to various services which are represented as $S = \{s_1, s_2, ..., s_G\}$. s_i has four parameters, service type s_i^{tp}, service priority s_i^{pr}, the required computing resource s_i^{cr} and the required storage resource s_i^{sr}. $W(s_i, e_q)$ indicates whether the service has been cached in the ES e_q, measured by

$$W(s_i, e_q) = \begin{cases} 1, & s_i \in c_q \\ 0, & otherwise \end{cases}. \tag{2}$$

Since the composition of the service request flow is changed over time, and the frequency of requesting a certain service is different in each time interval, the services cached in ESs should be dynamically changed. Only when $Q(r_{i,j}^t, e_q) = 1$ and $W(s_z, e_q) = 1$ (i.e., the customer requests a service cached in e_q) does e_q process $r_{i,j}^t$, where s_z is the service corresponding to $r_{i,j}^t$. Or else, $r_{i,j}^t$ is offloaded to the CCP and then processed by CCP. The cache hit rate (i.e., among all service requests, the proportion of the requests for cached services) of e_q at t is calculated by

$$\mathcal{S}hrate_q^t = \frac{1}{|\mathcal{R}_q^t|} \sum_{i=1}^{|U|} \sum_{j=1}^{|K_i^t|} Q(r_{i,j}^t, e_q) W(s_z, e_q), \tag{3}$$

where $|\mathcal{R}_q^t|$ represents the overall service request scale at time interval t, $|U|$ is the of RSU amount, $|K_i^t|$ represents the number of requests collected by u_i at time interval t.

C. Cost Models

To provide customers with a series of ICTS services, two kinds of costs should be mainly considered, the ES resource occupied by the service caching, and the time required for service request processing. The calculation methods of these two kinds of costs are discussed respectively in the resource model and latency model.

1) Resource Model: Traditional caching focuses on the storage space (i.e., the storage resource) allocation. For example, LRU (least-recently-used) and LFU (least-frequently-used) are designed to improve cache hit rate through cache replacement. While service caching requires not only storage space but also CPU and RAM resources (i.e., the computing resource). The computing and storage resources discussed in this model are provided for the services *need to be cached* (excluding the resources provided for the services *must be cached*).

The amount of computing resource required for each service varies depending on the type of service and the priority of the service. In this model, computing resource is abstracted as the number of computing units. Generally, the service with higher priority occupies more computing units (s_i^{cr} introduced in the framework indicates that the minimum number of computing units needed for s_i). When there are free computing units in a certain time unit, these units are allocated to the services that need to be computed according to the service priority. Assuming that the placed ESs have the same specifications and the computing resource of each ES is e^c, the storage resource of each ES is e^s, and s_i is requested at the time T_j, the additional computing resource allocated to s_i at the time T_j are calculated by

$$\begin{aligned} Add^{T_j}(s_i) &= \frac{1}{e^c}((e^c - \sum_{p=1}^{|c_q|} s_p^{cr}) \frac{s_i^{cr}}{\sum_{p=1}^{|c_q|} s_p^{cr}}) = \frac{1}{e^c}(\frac{e^c s_i^{cr}}{\sum_{p=1}^{|c_q|} s_p^{cr}} - s_i^{cr}) \\ &= s_i^{cr}(\frac{1}{\sum_{p=1}^{|c_q|} s_p^{cr}} - \frac{1}{e^c}) \end{aligned} \tag{4}$$

where $|c_q|$ represents the amount of services that e_q has cached. The computing resource occupancy of s_i at the time T_j are calculated by

$$Occ^{T_j}(s_i) = \frac{s_i^{cr}}{e^c} + Add^{T_j}(s_i) = \frac{s_i^{cr}}{\sum_{p=1}^{|c_q|} s_p^{cr}}. \tag{5}$$

Since the execution of the service requires some regular data and runs a series of pre-defined algorithms, the storage resource for the service caching is relatively fixed compared to the computing resource. The storage resource usage of e_q at the time T_j is expressed as

$$Sru^{T_j}(e_q) = \frac{1}{e^s} \sum_{p=1}^{|c_q|} s_p^{sr} \tag{6}$$

2) Latency Model: The time cost of service requests has three main components, transmission latency, propagation latency and execution latency. Due to the transmission latency and propagation latency between the customer and RSU are not affected by the service request processing way, they are not discussed.

The data upload and download delays constitute the transmission latency. This latency is only influenced by the speed of transmission and the amount of data that need to be

transmitted. The data of service request $r_{i,j}^t$ uploaded at the time interval t is denoted as $d(r_{i,j}^t)$ (this data is also the data downloaded by e_q or the CCP). After the service request $r_{i,j}^t$ is processed, the data uploaded is denoted as $d'(r_{i,j}^t)$ (this data is also the data downloaded by u_i). Generally, $d'(r_{i,j}^t)$ is approximately equal to $d(r_{i,j}^t)$. If s_i is cached in e_q, the latency of $r_{i,j}^t$ transmission is calculated by

$$L_{r_{i,j}^t}^{tm} = d(r_{i,j}^t)(\frac{1}{\theta_u} + \frac{1}{\partial_d} + \frac{1}{\partial_u} + \frac{1}{\theta_d}), \tag{7}$$

otherwise

$$L_{r_{i,j}^t}^{tm} = d(r_{i,j}^t)(\frac{1}{\theta_u} + \frac{2}{\partial_d} + \frac{2}{\partial_u} + \frac{1}{\theta_d}), \tag{8}$$

where θ_u and θ_{td} are respectively the upload and the download speed of RSU. ∂_u and ∂_d represent the upload and download speed of the ES and the CCP. Since the service request not executed on the ES is transmitted to the CCP for processing, the transmission of this service request has two additional interactions between the ES, CCP and RSU.

The service request delivery time and the result feedback time constitute the propagation latency. This latency is only affected by the propagation speed and the propagation distance. If s_i is cached in e_q, the latency of $r_{i,j}^t$ ropagation is calculated as

$$L_{r_{i,j}^t}^{pg} = 2\frac{dist(u_i, e_q)}{v_{u,e}}, \tag{9}$$

otherwise

$$L_{r_{i,j}^t}^{pg} = \frac{dist(u_i, e_q)}{v_{u,e}} + \frac{dist(e_q, CCP)}{v_{e,CCP}} + \frac{dist(CCP, u_i)}{v_{CCP,u}}, \tag{10}$$

where $dist$ is the distance (with euclidean metric) between two computing nodes (including RSUs, ESs and CCP), v represents the propagation speed between two nodes. Since the service request not executed on the ES is transmitted to the CCP for processing, the propagation of this service request has two additional interactions between the ES, CCP and RSU.

The execution latency is dependent on the computing units of ESs and the data of service requests. CCP has massive computing resource (i.e., the computing units), so the execution latency of the service requests processed in CCP is negligible. At the interval t, the sum of execution latency of the service request executed in e_q is calculated as

$$L_{e_q,t}^{ex} = \sum_{i=1}^{M}\sum_{j=1}^{K_i} Q(r_{i,j}^t, e_q)W(s_z, e_q)\frac{d(r_{i,j}^t)}{\eta e^c}, \tag{11}$$

where η is computing power of an computing unit. Due to the minimum number of computing units for each server is already set up, and there is a reallocation process on computing resource, the average execution is more informative. At the interval t, the average latency of the service request execution in e_q is calculated as

$$L_{r_{i,j}^t}^{ex} = \frac{L_{e_q,t}^{ex}}{\sum_{i=1}^{M}\sum_{j=1}^{K_i} Q(r_{i,j}^t, e_q)W(s_z, e_q)}. \tag{12}$$

D. Problem Definition

To obtain a high-performance strategy for service caching in ICTS, the cache hit rate and the time cost of request processing should be considered. We formulate the service caching problem as

$$max(\mathcal{S}hrate_q^t),\ min(L_{r_{i,j}^t}^{tm} + L_{r_{i,j}^t}^{pg} + L_{r_{i,j}^t}^{ex}), \tag{13}$$

$$\text{s.t.}\quad Sru^{T_j}(e_q) \le 1,\ \forall T_j, \forall q \in [1, N], \tag{14}$$

$$\sum_{s_i \in c_q} s_i^{cr} < e^c,\ \forall T_j, \forall q \in [1, N], \tag{15}$$

$$Occ^{T_j}(s_i) \ge \frac{s_i^{cr}}{e^c},\ \forall T_j. \tag{16}$$

The services cached in an ES cannot occupy more resource than the available resource of the ES. When a service cached is requested, the service must immediately occupy enough computing resource to ensure that the corresponding service request can be processed in time. Higher cache hit rate and lower latency are pursued while satisfying these constraints.

IV. GAT Based Service Request Flow Prediction

Owing to the strength of GAT in prediction, it has been mostly applied to traffic flow prediction. In this section, GAT is used to predict the service requests collected by each RSU in ICTS to achieve service request flow prediction (similar to traffic flow prediction). Based on the prediction results, the caching strategy of the ES is adjusted to prepare for the service request processing in next time interval. The scheme of service request flow prediction with GAT (SFT) is shown in Fig.2. The scheme consists of 3 main phases. firstly the graph structure is constructed based on the correlation between RSUs. Then GAT, a graph neural network (GNN) that introduces an attention mechanism, is used for service request flow prediction. Finally, the prediction results are mapped into graph node information and the data are extracted as input for the next phase of the task (SCAR).

A. Graph Neural Network Structure Construction

In the system framework, the service requests collected by u_i is R_i. Therefore, the service requests sent to the ES of the selected area are denoted as the set $\mathcal{R}_i = \{R_1^i, R_2^i, ..., R_Y^i\}$. To predict the service request flow in the future time period, the RSUs in the selected area are used as the nodes for service request flow collection and the ESs are the platform for prediction. According to the correlation degree of two nodes, each RSU is connected to its neighboring nodes by different edge weights. Since the relationship between nodes is undirected and all nodes are connected, the whole region is constructed as an undirected connected graph.

In each time interval, the record of a node is denoted as the set $Rec\{Ams, Srf, Pse\}$. In the set Rec, Ams is the average movement speed of customers on that road segment, which will affect the sending location of service requests in the next time interval. Srf is the service request flow, which reflects all services requested and their corresponding quantities in that time interval. Pse is the percentage of service requests

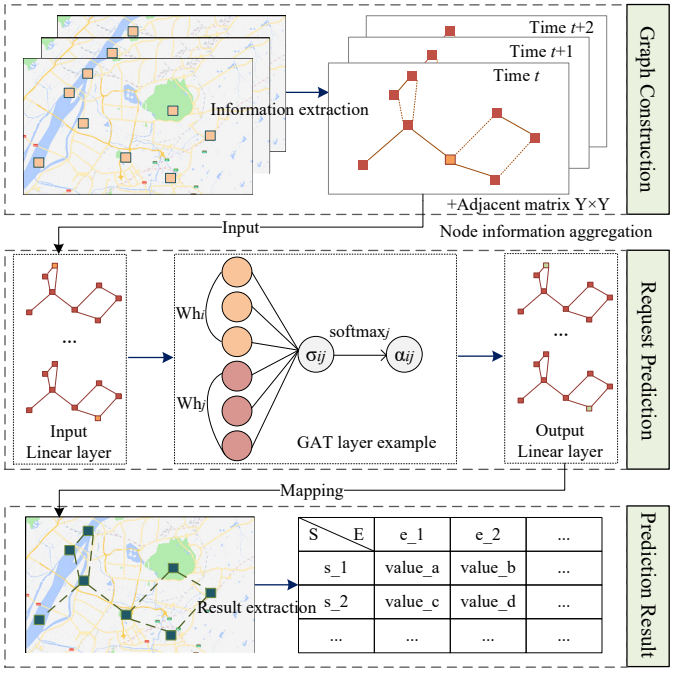

Fig. 2. The GAT-based service request flow prediction scheme for ICTS

collected by the RSU in the number of requests received by the ES in that time interval. These three indicators are considered as the features of each node. A series of node features are input to the building block layer of the GNN. The series of node features are represented as

$$h = \{\vec{h}_1, \vec{h}_2, ..., \vec{h}_Y\}, \vec{h}_i \in \mathbb{R}^F, \tag{17}$$

where Y and F are respectively the amount of nodes and the feature dimensions of nodes. Finally the layer outputs a new series of features (i.e., the objective function), which is are represented as

$$h^{'} = \{\vec{h^{'}}_1, \vec{h^{'}}_2, ..., \vec{h^{'}}_Y\}, \vec{h^{'}}_i \in \mathbb{R}^{F^{'}}, \tag{18}$$

where $\vec{h^{'}}_i$ is the prediction result at next t of node i. According to this output, the service requests sent to the ES at next t are updated to $\mathcal{R}_i^{'} = \{R_1^{i^{'}}, R_2^{i^{'}}, ..., R_Y^{i^{'}}\}$.

B. Prediction of Service Request Flow

The core point of GAT is to quantify the importance of the feature (i.e., attention), and to derive it through network training. The importance of the features of node j to node i is expressed as

$$\sigma_{ij} = a(W\vec{h}_i, W\vec{h}_j), \tag{19}$$

where W is a learnable linear transformation, $W \in \mathbb{R}^{F \times F^{'}}$, $\vec{h}_i$ is the series features of node i. To make the attention coefficients easily comparable across nodes, the softmax function is adopted to achieve normalization of the importance of all neighbor nodes. The normalized coefficient of attention is expressed as

$$\alpha_{ij} = softmax_j(\sigma_{ij}) = \frac{\exp(\sigma_{ij})}{\sum_{k \in \mathfrak{A}_i} \exp(\sigma_{ik})}, \tag{20}$$

where represents the neighbor nodes of node i. After obtaining the normalized attention coefficients, the linear combination of features corresponding to them can be calculated as the final output features of each node. The output series of features of node i are calculated as

$$\vec{h^{'}}_i = \mu(\sum_{j \in \mathfrak{A}_i} \alpha_{ij} W \vec{h}_j) \tag{21}$$

where μ is a nonlinear activation function. The larger the α_{ij}, the closer the relationship between two nodes (i.e., the greater the influence of node j on node i). It is essential to note that α_{ji} is usually $\neq \alpha_{ij}$, which shows that node i and node j have different degrees of influence on each other.

Based on GAT, a new service flow prediction algorithm is designed. By calculating the attention coefficient between the current node and its neighbors, the attention coefficients are weighted for calculating the output features of the current node. GAT focuses more on the key nodes and reduces the influence of noisy nodes. The details of SFT are given in Algorithm 1.

Algorithm 1: Service Request Flow Prediction

Input: the records set $Rec\{Ams, Srf, Pse\}$, the series of node features $h = \{\vec{h}_1, \vec{h}_2, ..., \vec{h}_Y\}$

Output: the service request flow prediction result $h^{'} = \{\vec{h^{'}}_1, \vec{h^{'}}_2, ..., \vec{h^{'}}_Y\}$

```
Construct an undirected graph G(V, E) based on node data;
Initialize the edge weights;
for each episode do
    Achieve the novel W;
    for (i = 1; i <= Y; i++) do
        Calculate the importance of the features of node j to node i → σ_ij ;
        Normalized the attention coefficient with α_ij = softmax_j(σ_ij) = exp(σ_ij) / Σ_{k∈A_i} exp(σ_ik);
        Differentiated information aggregation of neighbor nodes with attention coefficients;
        Complete the graph convolution operation.
    end
    Obtain h'.
end
return the prediction results;
```

V. SCAR for ICTS Service Caching

The section contains two parts. A. The definition of markov decision process (MDP). B. The detailed description of the proposed SCAR.

A. MDP Definition

The goal of reinforcement learning is to construct a MDP and find the optimal strategy. The strategy is the mapping of states to actions, and it makes the final cumulative reward maximum. MDP is a markov process that considers action strategies, i.e., each state under the system is related not only

to the current state but also to the action currently taken. Since reinforcement learning learns based on the rewards given by the environment, the corresponding MDP should also include the reward values.

For the the problem defined in Section III, the agents are set on ESs, and the service providing condition in ICTS is considered as environment. The MDP is defined as $M = (St, Ac, Po, Re)$, where St is the state, Ac is the action, Po is the state transition probability, and Re is the reward value. Specific definitions are given:

1) State Space: Since service request flow changes dynamically, the caching strategy for each ES should also change over time. The state at t is represented as

$$state(t) = \{[sc(t)], [tp(t)], [am(t)]\}, \tag{22}$$

where sc is the cache state of services, tp is the types of service requests , am is the amounts of different service requests. In the MDP, the next state is only affected by the current state. The previous state has no effect on the next state (i.e., $state(t+1)$ is only influenced by $state(t)$).

2) Action Space: The action (i.e., an update of service caching condition) that the agent takes is is dependent on the current state. Taken an action at t is recorded as a list $action(t)$, assuming that there are three services, and $[0, 1, 1]$ indicates that the second and third services are cached.

3) State Transition Probabilities: The state transition probability is the probability of changing from the current state to the next state which is belonging to the next time interval. The state transition probability of changing from $state(t)$ to $state(t+1)$ with $action(t)$ is represented as

$$Pb = (state(t+1)|state(t), action(t)). \tag{23}$$

Taking $action(t)$ at t is noted as a policy, denoted as

$$\pi(state(t), action(t)). \tag{24}$$

4) Rewards Mechanism: The service request processing latency is negatively correlated with the cache hit rate. When the cache hit rate of the ES e_q rises, the latency of service request processing falls. Therefore, the cache hit rate greatly determines the reward value. However, service caching has to consider resource constraints, and the cached services take up more computing or storage resources than the limit is not allowed. In conclusion, the reward function is represented as

$$Rwd = \sum_{i=0}^{N} (\mathcal{S}hrate_i + B_1(\sum_{s_j \in c_i} s_j^{cr} - e^c) + B_2(Sru(e_i))), \tag{25}$$

where B_1 and B_2 are punishment weights, they are able to make the reward value extremely low.

B. The Service Caching Scheme Based on A3C

Fig.3 shows an service caching scheme based on A3C (SCAR) for ICTS. The output of SFT (i.e., the result of service request flow prediction) is adopted as part of the input to the SCAR. There are two commonly used classical deep reinforcement learning methods. They are respectively entitled deep Q-network (DQN) and deep deterministic policy gradient (DDPG). DQN uses two key techniques, a replay buffer to break the correlation between samples, and a fixed target network for better training stability and convergence. DQN copes well with high-dimensional inputs, while it is helpless for high-dimensional action outputs. DDPG has the advantage of solving high-dimensional or continuous action spaces. It consists of an actor network and a critic network, and these two networks are responsible for generating actions and judging actions respectively. Similar to DQN, DDPG also uses a replay buffer and a fixed target network, and it is an Actor-Critic method combined with the deep network. The A3C-based service caching in ICTS has made engineering improvements to DDPG, using multiple agents to interact with the environment, increasing the speed of data acquisition and improve training efficiency.

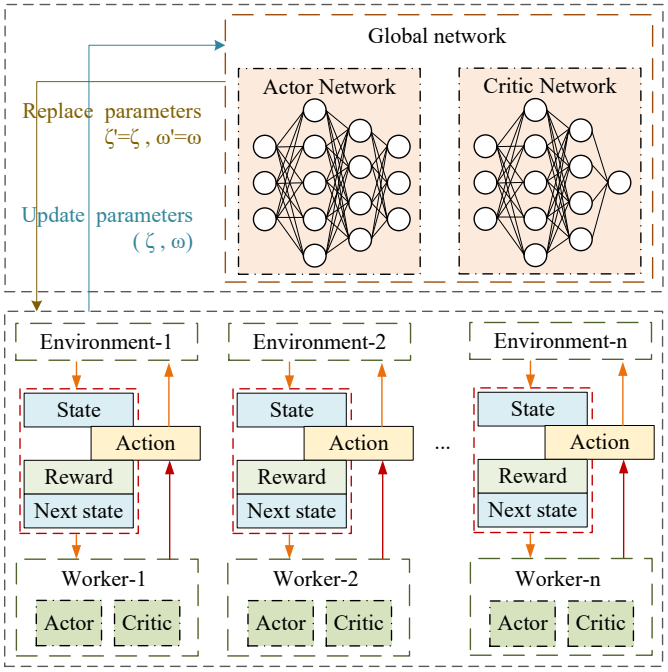

Fig. 3. The A3C-based service caching scheme for each ES in ICTS

1) Cache Update: The cache state of the services in ES and the number of requests received by ES for each type of service are considered as the primary state which is input to the agent. Subsequently, an action is performed by the agent for the state in this time interval. The cache state is updated according to the application of this action in the environment. The reward for action execution is calculated by equation (25). During training, the experience (a piece of experience is consist of a state, an action and a next state) is put into a replay buffer, which serves to overcome the experience data dependency. The global network in A3C consists of two core networks which are the same as that in DDPG. Multiple workers have the same configuration are set for interacting with environment in their own thread. When a worker has received the result after interacting, the gradient is calculated. The gradient is used for updating the global network. That is, multiple threads update the parameters in global network using the cumulative gradients separately. At regular intervals, the threads update

the parameters of their own networks to those of the global network, which in turn guide the later interactions.

2) The Network training: In the global network, the parameter of the actor network is ζ, and the parameter of the critic network network is ω. Correspondingly, in each worker, the parameter of the actor network is ζ', and the parameter of the critic network network is ω'. First, set the relative updated value $d\zeta$ of ζ and the relative updated value $d\omega$ of ω in the global network to 0. Meanwhile, assign the ζ' and ω' of the thread-local network to the ζ and ω of the global network. Then let the agent in the current thread keep exploring until the step limit is reached. During this process, the action is output using policy $\pi(state(t), action(t))$ (i.e., using the local actor network to take actions).

The gradient update equation is as follows.

$$d\zeta \leftarrow d\zeta + \nabla_{\zeta'} log Pb_t(\mathbb{X} - V(state(t); \omega')), \tag{26}$$

and

$$d\omega \leftarrow d\omega + \frac{\partial(\mathbb{X} - V(state(t); \omega'))^2}{\partial \omega'}, \tag{27}$$

where $\mathbb{X}$ represents the reward valuation ($\mathbb{X} = \mathcal{S}hrate^t + \gamma\mathbb{X}$), while $V(state(t); \omega'))$ is the reward valuation in $state(t)$. Algorithm 2 shows the details of SCAR for ICTS.

Algorithm 2: Service caching based on A3C

Input: space of state $state$, time interval number $\mathcal{T}$, replay buffer, workers, global network
Output: strategies for caching

Initialize replay buffer, agents and global network;
$t = 1$;
$d\zeta \leftarrow 0$, $d\omega \leftarrow 0$;
for *each episode* **do**
  Synchronize parameters to this thread's neural network $\zeta' = \zeta$, $\omega' = \omega$;
  Get $state(t)$;
  for $(t_{start}; t <= \mathcal{T}; t++)$ **do**
    if *the initial state is* $state(t)$ **then**
      Random an $action$;
    else
      $action(t)$ is taken with ζ according to formula (24);
    end
    Computing reward by formula (25);
  end
  Calculate the reaward valuation $\mathbb{X}$;
  for $j \in (t-1, t-2, ..., t_{start})$ **do**
    Calculate the reward valuation $\mathbb{X}_j$ for the time interval j;
    The local accumulative gradient update of ζ': using formula (26);
    The local accumulative gradient update of ω': using formula (27);
  end
  Update the ζ with $d\zeta$ and update ω with $d\omega$ asynchronously;
end
return ζ and ω of global network;

VI. Experiment and Analysis

The setup of experiments and the comparative schemes are firstly introduced. With the implementation of SFT-SCAR and the experimental results, the caching strategies are simulated in the DT of ICTS and its performance is evaluated. The effectiveness of SFT-SCAR will be analyzed with regard to prediction accuracy, cache hit rate and time cost of requests.

A. Experimental Setup

The dataset used to assess the accuracy of SFT predictions is downloaded from the Caltrans Performance Measurement System (https://pems.dot.ca.gov). The data in the dataset (including traffic, user location) is collected in real time by more than 39,000 individual detectors. These sensors span the freeway system across all major metropolitan areas of the State of California. Besides, the dataset used to evaluate the performance of SFT-SCAR in solving the service caching problem in ICTS was generated with reference to the simulation in [31] (Generate service request datasets in proportion, e.g. Request A: 0.6, Request B: 0.3, Request C: 0.1). Regarding the parameter configuration of A3C, the network consisted of 3 fully connected layers with 200 neurons and 2 fully connected layers with 100 neurons, with a learning rate of 0.0001 and a reward discount of 0.95. The Adam optimizer was used to adaptively adjust the learning rate, with the value set to (0.95, 0.999).

To analyze the performance of SFT-SCAR, we implement three schemes for performance comparison, and they are described as follows:

- Static caching scheme (SCS)
  The caching decision of this scheme is based on the historical number of requests for each service (without relying on the request flow prediction), while ensuring that resource usage does not exceed the limit. Once services with high request rate have been cached, the caching strategies are not automatically updated.
- DQN-based service caching scheme (DQN)
  DQN is a powerful reinforcement learning method in solving problems such as task offloading [32]. DQN usually outputs discrete actions.
- DDPG-based service caching scheme (DDPG)
  This service caching scheme makes strategies based on DDPG [33]. DDPG is a method that can solve the problem of continuity control. The output of DDPG is continuous.

B. Analysis of the Prediction Performance

GAT introduces an attention mechanism that allows for adaptive weight allocation to the relationships between nodes, enabling the model to focus on nodes with higher relevance during the learning process, improving the efficiency and accuracy of representation learning. Due to the high accuracy of existing studies using GAT for traffic flow prediction, we decided to involve a service request flow prediction method based on GAT, SFT, to provide an initial input to SCAR. Figure 4 demonstrates a similar prediction accuracy as [1].

According to the Pareto principle (i.e. for any set of things, the most important is only a small part of it, about 20%, and the remaining 80%, although the majority, is less important), user requests are concentrated on 20% services, so that degree of prediction accuracy is sufficient to reflect user requirements and achieve the objectives of the scheme. As the service requests originate from customers in the ICTS, the service request flow has the similar temporal and spatial characteristics as the traffic flow. Therefore it is reasonable for this experiment to generate service request flows based on traffic flows. The predicted and actual values of the number of requests for a service over 200 time intervals are shown in Fig.4. The data show the service request flow prediction scheme has a high accuracy rate, and the error values between predicted and actual values are small and only become relatively large when there is a measurable change in the actual values. According to Pareto principle, only a small part of the services are requested by customers in large numbers at a certain time interval. Hence, the acceptable error in the predicted result has a little impact on the proportion of this type of service request to all service requests. In addition, the figure demonstrates the stability of the GAT prediction performance, maintaining essentially a similar and high accuracy rate across all time intervals. The loss of each Epoch is shown in Fig.5. In this

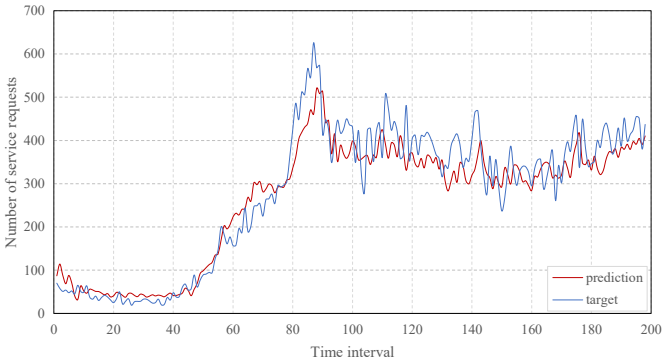

Fig. 4. Service request flow predicted value and actual value

figure, the loss eventually drops to around 0.2 at the 20th epoch, and the loss fluctuates slightly at this value for the following epochs. This shows that the algorithm has good convergence.

C. Analysis of Service Caching Performance

The service caching strategies generated by SFT-SCAR should not be applied directly until their validity has been verified. Due to the massive service requests in ICTS and their varying sensitivity to latency, direct application of low performance caching strategies will significantly degrade QoS. So the caching strategies are simulated in the DT of ICTS to evaluate their performance. To the service caching problem, the cache hit rate is a significant indicator. The higher cache hit rate means that more services are processed on the ESs, thus reducing the overall response latency of service requests.

In this experiment, the service caching strategies for the different ESs are recorded for analysis. Fig.6 and Fig.7 show the

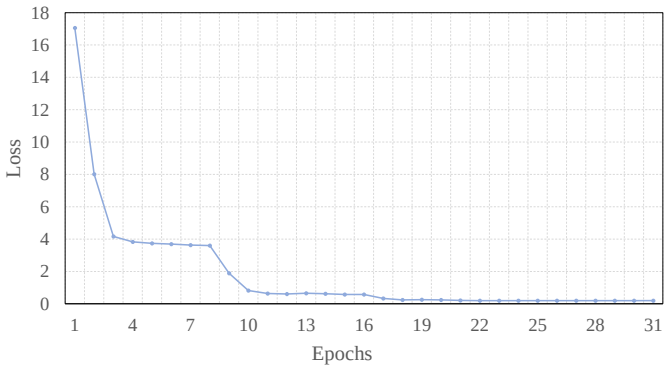

Fig. 5. The test-loss value of prediction (measuring the distortion rate of prediction results)

cache hit rates corresponding to the service caching strategies generated by SFT-SCAR, DDPG, DQN and SCS. In Fig.6, only in 4 time intervals does SFT-SCAR perform slightly worse than DDPG, in the other 12 time intervals SFT-SCAR has a different degree of advantage. In Fig.7, only in 2 time intervals does DDPG performs better. In contrast to DDPG, workers in A3C interact with the environment independently through multiple threads, while asynchronous training breaks the correlation of data. As a result, A3C usually performs better than DDPG on problems with large scale (including service caching problems, which are complicated by various constraints). DQN is inferior to the other two reinforcement learning schemes because of its poor performance on problems with complex action spaces. SCS performs extremely well or poorly at certain time intervals because it is non-dynamic, which brings low utility. Fig.8 visually shows the cache hit

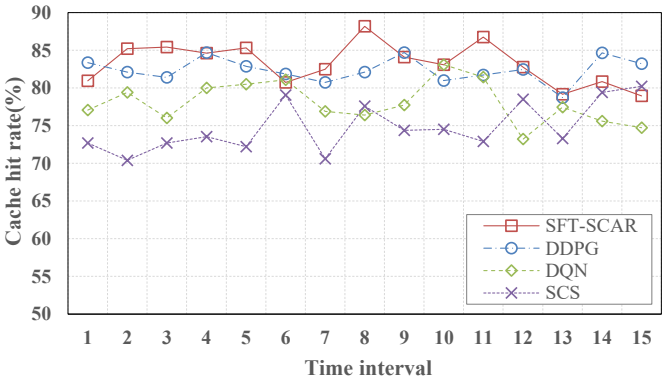

Fig. 6. Cache hit rate of ES-1 with SFT-SCAR, SCS, DQN and DDPG

condition of the strategies generated by SFT-SCAR, DDPG, DQN and SCS through the values of their minimum hit rate, average hit rate and maximum hit rate. The maximum hit rate of SFT-SCAR is about 4.14% higher than the maximum hit rate of DDPG, 6.17% higher than the maximum hit rate of DQN, and 9.92% higher than the maximum hit rate of SCS. The average hit rate of SFT-SCAR is about 1.11% higher than that of DDPG, 6.81% higher than that of DQN, and 13.27% higher than that of SCS. The maximum hit rate and the average hit rate of SFT-SCAR are both the highest, which demonstrates

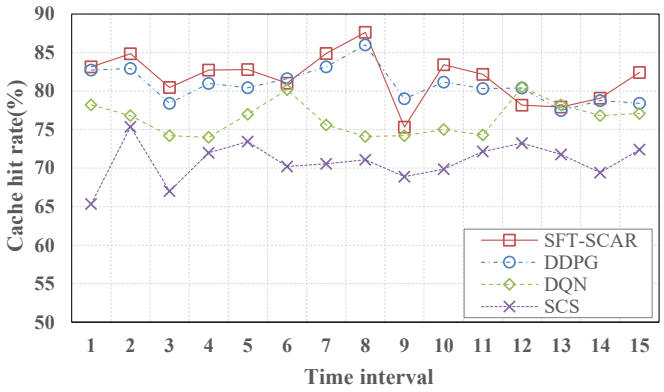

Fig. 7. Cache hit rate of ES-2 with SFT-SCAR, SCS, DQN and DDPG

the effectiveness of SFT-SCAR in making caching decisions. Regarding the minimum hit rate of SFT-SCAR, it is lower than DDPG, but higher than DQN and SCS. Having the highest average cache hit rate, but not the highest minimum hit rate, indicates that this strategy is generally effective only in a few time intervals (but still has a hit rate higher than 3/4), which also illustrates the superiority of SFT-SCAR.

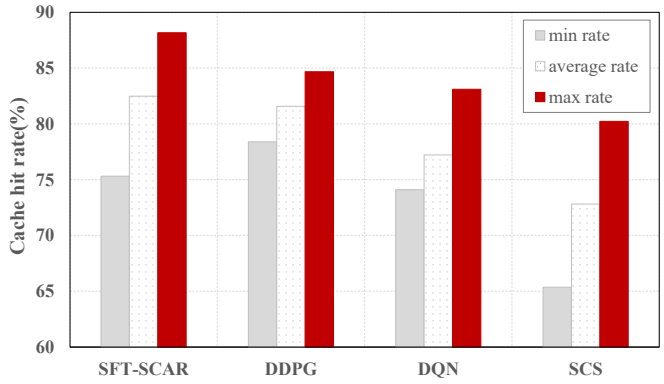

Fig. 8. Comparison on the minimum, average and maximum cache hit rate with SFT-SCAR, SCS, DQN and DDPG

Since the ES is closer to the customer, customers get feedback more quickly when requesting services cached in the ES. The response latency of the requests for strategy corresponding to Fig.6 is shown in Fig.9. Similar to the cache hit rate, only in 4 time intervals does SFT-SCAR have higher response latency than DDPG. In addition, the request response latency of SFT-SCAR is lower than the request response latency of DQN and SCS in almost all time interval. The result of the experiment shows that SFT-SCAR effectively reduces time cost of service requests processing in ICTS with cloud-edge computing. After testing strategies generated by SFT-SCAR in DT, the strategies are validated to be applied to real scenarios.

In Fig.7 and Fig.9, the reason that SCAR is only slightly higher compared to DDPG is that most of the service requests in this group have more distinct distribution characteristics, and thus both schemes obtain better decisions. It is possible that there is only difference in caching decisions on few services and there is little difference between the number of

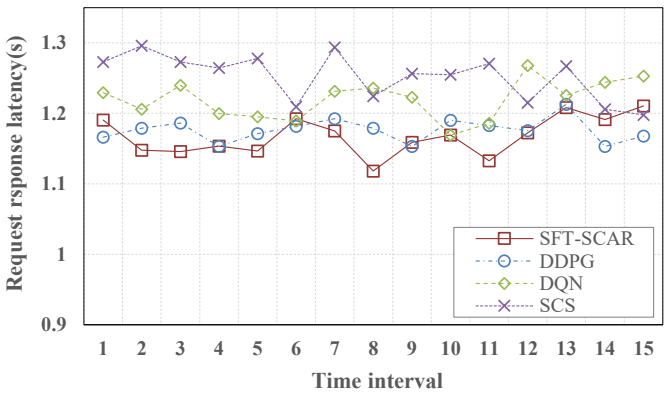

Fig. 9. Average request response latency with SFT-SCAR, SCS, DQN and DDPG

requests for these services, leading to a similarity in the performance of the final strategies. A3C has good parallelization capabilities, it uses multiple asynchronous actors to collect samples and update the strategies, and it learns in multiple environments at the same time, and therefore may be able to explore some better solutions.

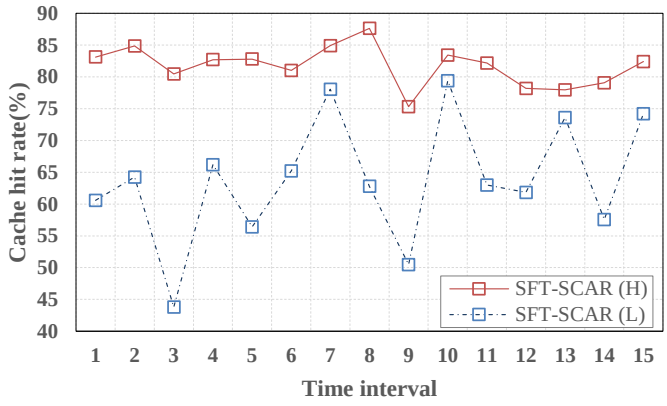

Fig. 10. Performance gap obtained by DT simulation

Fig.10 illustrates the normal output high-performance strategy (SFT-SCAR (H)) of SFT-SCAR and the low-performance strategy (SFT-SCAR (L)) due to an attack or an error in the training process. The major advantage of using DT to simulate service caching strategies is its ability to accurately model the real-world caching decision-making process in virtual environments, based on real-time data and physical models, so as to optimize the effect of caching strategies as well as the performance of service request flow prediction, and to provide highly credible predictions and guidance for practical decision-making. The data of the figures such as Fig.6 and Fig.7 in this section are generated with the assistance of the DT simulation. Without the assistance of DT, it is difficult to assess the performance of a scenario in advance without actually utilizing it. The core application of DT in this paper is putting the scheme into use virtually to avoid losses to consumers in real scenarios caused by deploying low performance service caching strategies.

VII. Conclusion

Responding to the challenge in service caching and the QoS assurance in ICTS, a service providing framework for ICTS is proposed, and a solution for service caching in ICTS named SFT-SCAR is proposed. The solution consists of two main modules, the GAT-based service request flow prediction scheme and the A3C-based service caching scheme. Besides, the strategies are simulated in DT and the efficiency is verified. Commonly, service providers should pay for the occupied ES resources (it should be a long-term occupation). The future work could be devoted to researching the caching problem or the allocation of ES resources in relation to service providers' costs.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant no. 92267104 and no. 62372242, and Natural Science Foundation of Jiangsu Province of China under no. BK20211284.

References

[1] X. Xu, Q. Jiang, P. Zhang, X. Cao, M. R. Khosravi, L. T. Alex, L. Qi, and W. Dou, "Game theory for distributed iov task offloading with fuzzy neural network in edge computing," *IEEE Transactions on Fuzzy Systems*, vol. 30, no. 11, pp. 4593–4604, 2022.

[2] Z. Wang, D. Jiang, Z. Lv, and H. Song, "A deep reinforcement learning based intrusion detection strategy for smart vehicular networks," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2022, pp. 1–6.

[3] J. Zhao, X. Sun, Q. Li, and X. Ma, "Edge caching and computation management for real-time internet of vehicles: An online and distributed approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 4, pp. 2183–2197, 2020.

[4] Y. Dai, D. Xu, S. Maharjan, G. Qiao, and Y. Zhang, "Artificial intelligence empowered edge computing and caching for internet of vehicles," *IEEE Wireless Communications*, vol. 26, no. 3, pp. 12–18, 2019.

[5] X. Xu, J. Gu, H. Yan, W. Liu, L. Qi, and X. Zhou, "Reputation-aware supplier assessment for blockchain-enabled supply chain in industry 4.0," *IEEE Transactions on Industrial Informatics*, 2022.

[6] Y. Liu, Q. He, D. Zheng, X. Xia, F. Chen, and B. Zhang, "Data caching optimization in the edge computing environment," *IEEE Transactions on Services Computing*, vol. 15, no. 4, pp. 2074–2085, 2020.

[7] P. Dai, K. Hu, X. Wu, H. Xing, and Z. Yu, "Asynchronous deep reinforcement learning for data-driven task offloading in mec-empowered vehicular networks." in *INFOCOM*, 2021, pp. 1–10.

[8] Z. Li, X. Xu, T. Hang, H. Xiang, Y. Cui, L. Qi, and X. Zhou, "A knowledge-driven anomaly detection framework for social production system," *IEEE Transactions on Computational Social Systems*, 2022.

[9] G. N. Schroeder, C. Steinmetz, R. N. Rodrigues, R. V. B. Henriques, A. Rettberg, and C. E. Pereira, "A methodology for digital twin modeling and deployment for industry 4.0," *Proceedings of the IEEE*, vol. 109, no. 4, pp. 556–567, 2020.

[10] A. Banchs, D. M. Gutierrez-Estevez, M. Fuentes, M. Boldi, and S. Provvedi, "A 5g mobile network architecture to support vertical industries," *IEEE Communications Magazine*, vol. 57, no. 12, pp. 38–44, 2019.

[11] Z. Lv, Y. Li, H. Feng, and H. Lv, "Deep learning for security in digital twins of cooperative intelligent transportation systems," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[12] H. Darvishi, D. Ciuonzo, E. R. Eide, and P. S. Rossi, "Sensor-fault detection, isolation and accommodation for digital twins via modular data-driven architecture," *IEEE Sensors Journal*, vol. 21, no. 4, pp. 4827–4838, 2020.

[13] X. Xu, S. Tang, L. Qi, X. Zhou, F. Dai, and W. Dou, "Cnn partitioning and offloading for vehicular edge networks in web3," *IEEE Communications Magazine*, 2023.

[14] X. Lin, J. Wu, J. Li, W. Yang, and M. Guizani, "Stochastic digital-twin service demand with edge response: An incentive-based congestion control approach," *IEEE Transactions on Mobile Computing*, 2021.

[15] Z. Tu, L. Qiao, R. Nowak, H. Lv, and Z. Lv, "Digital twins-based automated pilot for energy-efficiency assessment of intelligent transportation infrastructure," *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[16] Q. Xu, Z. Su, and R. Lu, "Game theory and reinforcement learning based secure edge caching in mobile social networks," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3415–3429, 2020.

[17] Z. Lv, D. Chen, and Q. Wang, "Diversified technologies in internet of vehicles under intelligent edge computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 4, pp. 2048–2059, 2020.

[18] Z. Ning, K. Zhang, X. Wang, L. Guo, X. Hu, J. Huang, B. Hu, and R. Y. Kwok, "Intelligent edge computing in internet of vehicles: a joint computation offloading and caching solution," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 4, pp. 2212–2225, 2020.

[19] B. Li, X. Deng, and Y. Deng, "Mobile-edge computing-based delay minimization controller placement in sdn-iov," *Computer Networks*, vol. 193, p. 108049, 2021.

[20] K. Wang, X. Wang, and X. Liu, "A high reliable computing offloading strategy using deep reinforcement learning for iovs in edge computing," *Journal of Grid Computing*, vol. 19, no. 2, pp. 1–15, 2021.

[21] S. Bi, L. Huang, and Y.-J. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 19, no. 7, pp. 4947–4963, 2020.

[22] T. Zhao, I.-H. Hou, S. Wang, and K. Chan, "Red/led: An asymptotically optimal and scalable online algorithm for service caching at the edge," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 8, pp. 1857–1870, 2018.

[23] D. Ren, X. Gui, and K. Zhang, "Adaptive request scheduling and service caching for mec-assisted iot networks: An online learning approach," *IEEE Internet of Things Journal*, 2022.

[24] X.-Q. Pham, T.-D. Nguyen, V. Nguyen, and E.-N. Huh, "Joint service caching and task offloading in multi-access edge computing: A qoe-based utility optimization approach," *IEEE Communications Letters*, vol. 25, no. 3, pp. 965–969, 2020.

[25] X. Ma, A. Zhou, S. Zhang, and S. Wang, "Cooperative service caching and workload scheduling in mobile edge computing," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2076–2085.

[26] W. Sun, P. Wang, N. Xu, G. Wang, and Y. Zhang, "Dynamic digital twin and distributed incentives for resource allocation in aerial-assisted internet of vehicles," *IEEE Internet of Things Journal*, vol. 9, no. 8, pp. 5839–5852, 2021.

[27] J. Zheng, T. H. Luan, L. Gao, Y. Zhang, and Y. Wu, "Learning based task offloading in digital twin empowered internet of vehicles," *arXiv preprint arXiv:2201.09076*, 2021.

[28] C. Tan, X. Li, T. H. Luan, B. Gu, Y. Qu, and L. Gao, "Digital twin based remote resource sharing in internet of vehicles using consortium blockchain," in *2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)*. IEEE, 2021, pp. 1–6.

[29] T. Liu, L. Tang, W. Wang, X. He, and Q. Chen, "Resource allocation via edge cooperation in digital twin assisted internet of vehicle," in *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2021, pp. 1–6.

[30] K. Zhang, J. Cao, and Y. Zhang, "Adaptive digital twin and multi-agent deep reinforcement learning for vehicular edge computing and networks," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 2, pp. 1405–1413, 2021.

[31] H. Yan, X. Xu, F. Dai, L. Qi, X. Zhang, and W. Dou, "Service caching for meteorological emergency decision-making in cloud-edge computing," in *2022 IEEE International Conference on Web Services (ICWS)*. IEEE, 2022, pp. 120–128.

[32] C. Yang, X. Xu, X. Zhou, and L. Qi, "Deep q network–driven task offloading for efficient multimedia data analysis in edge computing–assisted iov," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 18, no. 2s, pp. 1–24, 2022.

[33] W. Liu, X. Xu, L. Qi, X. Zhang, and W. Dou, "Godeep: Intelligent iov service deployment and execution with privacy preservation in cloud-edge computing," in *2021 IEEE International Conference on Web Services (ICWS)*. IEEE, 2021, pp. 579–587.