IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON GREEN INTERNET OF THINGS

Received June 29, 2020, accepted July 15, 2020, date of publication July 20, 2020, date of current version July 30, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3010329

Cooperative Edge Caching: A Multi-Agent Deep Learning Based Approach

YUMING ZHANG[1], BOHAO FENG[1], (Member, IEEE), WEI QUAN[1], (Member, IEEE), ALETENG TIAN[1], KESHAV SOOD[2], (Member, IEEE), YOUFANG LIN[3], AND HONGKE ZHANG[1], (Senior Member, IEEE)

[1]School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China
[2]School of Information Technology, Deakin University, Melbourne, VIC 3125, Australia
[3]School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China

Corresponding author: Bohao Feng (bhfeng@bjtu.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61802014.

ABSTRACT Ubiquitous Internet of Things (IoT) devices have fueled plenty of innovations in the emerging network paradigms. Among them, IoT edge caching has emerged as a promising technique to cope with the explosive growth in network data traffic, with Quality of Service (QoS) improved and energy saved. However, the intrinsic storage limitations of the edge servers poses a critical challenge for the IoT edge caching system. Enabling edge servers to cooperate with each other can provide a potential perspective to improve the edge storage utilization widely discussed. Nevertheless, it also incurs an additional communication overhead, eventually making the caching system more complex. As a result, how to perform an efficient cooperative caching becomes a critical issue. Thus, in this paper, we propose a deep reinforcement learning-based cooperative edge caching approach, which allows the distributed edge servers to learn to cooperate with each other. Specifically, edge servers determine their cache actions based on the local caching state. After that, the centralized remote server evaluates these actions and feeds back the evaluation results to edge servers for subsequent caching actions optimization. We show that, by designing an appropriate reward function, our approach promotes cooperation between edge servers as well as improving the system hit rate. On this basis, we consider a practical and reasonable scenario with inconsistent data item size and propose a novel multi-agent actor-critic caching algorithm. Extensive simulation results demonstrate the performance improvement using our proposed solution over three other caching algorithms.

INDEX TERMS Cooperative edge caching, Internet of Things, multi-agent deep learning, actor-critic, multi-agent deep deterministic policy gradient.

I. INTRODUCTION

Fueled by the ubiquity of the Internet of Things (IoT) devices, IoT data has been growing exponentially. According to the related reports, the number of devices connected to IP networks will be more than three times the global population by 2023 [1] and the total data volume of connected IoT devices is projected to reach 79.4 zettabytes by 2025 [2]. Further, researchers have noted that, it is challenging for the current network paradigm to accommodate these data especially in terms of storage and transmission. This mainly results from the fundamental design of the Internet [3], [4].

A promising technique to address this challenge is the emerging edge caching [5], which provides cloud-like storage resources for IoT data at the edge of the network. As a consequence, IoT applications can retrieve these data without going through the backhaul link, resulting in a significant reduction in data transmission [6]. Furthermore, it also saves the energy consumption of edge servers by reducing the non-essential utilization of backhaul connections [7], and further decreases the energy cost of IoT devices by minimizing the time to obtain IoT data [8].

In fact, edge caching has been investigated exhaustively [9]. Most of them focus on caching points in isolated domains, either at the network edge or the base station. Nevertheless, in practical scenarios, the caching performance is always constrained by the limited storage of an individual edge server [10]. An effective solution is to enable edge servers to cooperate with each other, by sharing data items through their internal connections. Consequently, the set of

The associate editor coordinating the review of this manuscript and approving it for publication was Nan Cheng.

cached data can be enlarged and IoT devices can be served by multiple edge servers to exploit caching diversity [11].

Moreover, designing a caching mechanism for each server independently may also result in an insufficient utilization of caches. This happens, either when the cached data in neighboring servers are overlapped or when the caching loads of edge servers are imbalance [5]. To this end, a well-tuned cooperative caching mechanism is needed, with the purpose of fully utilizing vacant servers and avoiding caching too many redundant data duplicates with data retrieving time guaranteed.

Exploiting the envisioned cooperative edge caching poses several technical challenges. First, in order to actualize cooperation, edge servers should be aware of the caching status of their neighboring servers by an information exchange mechanism, which may induce significant overheads [5]. Second, a dynamic and adaptive caching framework is required for efficient cooperation management [12]. Third, the caching algorithm should be designed with the capability of tackling large-scale information, incurred by information interaction and huge data items. To the best of our knowledge, techniques, such as integer linear programming or dynamical programming, are qualified in addressing the first two issues [13], [14]. However, many projections of them may become prohibitive in problems involving high-dimensional parameters [15]. Recently, with the significant exploration of edge computing [16] and deep reinforcement learning (DRL) based approaches [17], [18] in coping with system dynamics and complexity, we are motivated to use these approaches in our problem domain. Specifically, a DRL agent can be implemented at the edge server to seek for its optimal caching decisions and learn to dynamically adjust its actions based on the other servers' information. Then, with an appropriate reward function, all servers can work cooperatively with overall performance improved.

In this work, we investigate the cooperative caching mechanism in the IoT edge caching scenario, where IoT applications send the request to its assigned edge servers and the edge servers perform cooperatively to satisfy the request. For practicality and generality, we consider that IoT data items are of different sizes, which is reasonable in most actual IoT environments but are not sufficiently investigated. Our main objective is to maximize the overall hit rate, further, minimize the backhaul traffic cost. Inspired by [19], [20], we first propose a deep reinforcement learning-based cooperative edge caching approach by leveraging the actor-critic reinforcement learning scheme [21]. In the proposed approach, each edge server makes their caching decisions based on the local state, and its model is updated by the Temporal-Difference error (TD-error), which is calculated depending on other servers' caching states and actions in a remote centralized server. Besides, we propose a multi-agent actor-critic cooperative caching algorithm and explore the impact of cooperation between edge servers on the system hit rate. We validate the performance improvements of our algorithm over two conventional caching algorithms and one state-of-the-art learning-based caching algorithm [22]. Simulation results also point out reasons to indicate how the cooperative edge caching could achieve a higher system hit rate and why the proposed algorithm outperforms the others.

Our main contributions of this work are as follows.

- We propose a deep reinforcement learning-based cooperative edge caching approach by combining the characteristics of the deep reinforcement learning approach and the IoT edge caching system. We define the feature and action spaces to cope with the obstacles caused by the inconsistent IoT data item size, and design an efficient reward function to promote cooperation between edge servers.
- We develop a multi-agent actor-critic algorithm to conduct the proposed approach. In particular, each edge server makes caching decisions locally. After that, the remote centralized server evaluates the actions based on the global caching information and feeds back the evaluation results to edge severs to optimize their subsequent caching decisions. We treat TD-errors as the evaluation results and use them to update the model in the edge server.
- We demonstrate the performance improvements of the proposed algorithm over two conventional caching algorithms and one state-of-the-art learning-based algorithm. The simulation results also provide a view of how the edge servers cooperating with each other and how the cooperative edge caching achieving a higher hit rate.

The organization of this paper is as follows. We present related works about edge caching in Section II and illustrate the IoT edge caching system model in Section III. In Section IV, we describe the proposed framework and provide the definition of caching state, action, and reward. Based on the proposed framework, we develop a cooperative edge caching algorithm in Section V. Simulation results and performance analysis are shown in Section VI. And finally, we conclude this paper in Section VII.

II. RELATED WORK

A. DEEP REINFORCEMENT LEARNING BASED EDGE CACHING

Due to the efficiency of reinforcement learning approaches in making caching decisions, many excellent works have been done to cope with the key issues in edge caching systems.

In [23], authors have discussed key issues in mobile edge caching and propose a learning-based mobile edge caching schemes to achieve context awareness and intelligence in mobile edge caching. By combining the DDPG (Deep Deterministic Policy Gradient) [24] with the Wolpertinger architecture [25], which can narrow the size of the action space and avoid missing the optimal policy, the authors of [26] propose a DRL-based framework aimed at maximizing the hit rate. To jointly consider data-transiency and dynamic context characteristics, the authors use DRL to solve the problem of caching IoT data at the edge [27], which aims

to strike a balance between the communication cost and the loss of data freshness. Besides, in [28], the authors have leveraged the actor-critic learning framework to solve the joint decision-making problem in the cloud-based IoT scenario. In this framework, caching strategy, computation offloading policy, and radio resource allocation are simultaneously considered. Using edge caching as one of the key enablers in the services of smart cities, the authors in [29] integrate networking, caching, and computing resources to improve the performance of applications for smart cities and then develop a novel big data deep reinforcement learning approach. Also, an integrated framework is proposed in [30], which enables dynamic orchestration of networking, caching, and computing resources in the vehicular networking [31], and the DRL approach in this work is aimed at solving the resource allocation problem.

Unlike most of the aforementioned works, size of the content items is assumed to be the same, but in [30], authors have taken the content item's size as a feature, based on which they have developed a model-free reinforcement learning algorithm to address the cache admission problem with the goal of improving the hit rate. In [33], authors have proposed a DRL algorithm to solve the resource allocation problem, with an objective to jointly optimize tasks offloading, cache allocation, computation allocation, and dynamic power distribution. Besides, the authors of [34] leverage a reinforcement learning approach to deal the edge caching problems in the vehicular networks, aiming to enhance the download rate of vehicles. Additionally, a DRL with the multi-time scales framework is developed in [35], which is used to tackle the joint communication, caching, and computing design problem in the vehicular network, taking into account the vehicles' mobility and the hard service deadline constraint.

B. COOPERATIVE EDGE CACHING

Note that, in cooperative edge caching scenarios, a request may be served either by the local edge server or the neighbor edge server. And this implies that the cached content can be shared between edge servers. Therefore, cooperation has become another research direction in the edge caching literature in order to enhance performance.

In [36], the authors have considered an integrated mobile content distribution framework with universal in-network caching and collaboration across domains and develop a context-aware solution to provide collaborative video content distribution. By investigating the impact of collaborative cache constraints, the authors of [37] propose a collaborative cache scheme for the 5G wireless network, which improves the overall performance of the wireless networks in terms of transmission delay, local availability of contents and utilization of caches. By leveraging information-centric networking, the authors of [38] have proposed a social-aware fog network and develop a social-aware content caching and distribution scheme allowing nodes to collaboratively cache content locally. In [39], the authors formulate an optimization problem to cope with the issues of collaborative cache allocation and computing offloading, aiming at maximizing resource utilization. By considering content sharing between user devices, the authors of [40] construct a content sharing framework in the D2D scenario and propose a distributed collaborative cache management scheme to make caching decisions. In [41], the authors have developed an analytical framework to evaluate the data volume that can be downloaded using a cooperative drive-thru Internet scheme.

Further, in our literature review we have also noticed some works using learning-based approaches to address the cooperation problems. In [42], two novel proactive cooperative caching approaches are proposed to predict users' content demand in mobile edge caching network, by leveraging deep learning algorithms. Similarly, a proactive caching mechanism, named Learning-based Cooperative Caching strategy, is proposed to reduce transmission cost while improving QoE for mobile edge computing. Authors in [41] have used the Q-learning method to design the cache mechanism and develop an action selection strategy for the cache problem. Through the learning-based method, the appropriate caching state can be found and the cache performance can be effectively improved. In [44] and [45], firstly authors have modeled the cooperative content caching problem as a multi-agent multi-armed bandit problem, and then propose a multi-agent reinforcement learning-based algorithm to solve the problem. Very recently, the authors of [22] develop a deep actor-critic reinforcement learning-based multi-agent framework to make caching decisions in the cooperative edge caching scenario, aiming to minimize the overall transmission delay. We have used this algorithm in our research modified it and further re-evaluated it. The proposed modified algorithm is compared with the original one.

III. SYSTEM MODEL

In this section, the IoT edge caching model is introduced and basic assumptions are given first. Besides, the caching decision issue is formulated as an optimization problem with the objective of maximizing backhaul traffic reduction.

A. MODEL DESCRIPTION

The proposed IoT edge caching is illustrated in Fig. 1. IoT applications have access to Edge Servers within their service region. All Edge Servers are connected with each other and can retrieve the IoT data from the Internet through a reliable backhaul link. There also exits a Central Server, that acts as a manager of all edge servers.

In this work, Edge Servers are the cache entity with limited storage and work cooperatively to share IoT data items. For example, if a data request cannot be fulfilled locally by Edge Server 1, it can be satisfied by Edge Server 2 or Edge Server 3 through the internal connections depending on whether Edge Server 2 or Edge Server 3 has the requested data item. The Central Server does not store any IoT data item. Instead, it has a global view of caching states. Besides, when all Edge Servers cannot fulfill an IoT data request, the Central Serve is responsible for fetching the requested data item from the Internet through the backhaul link.

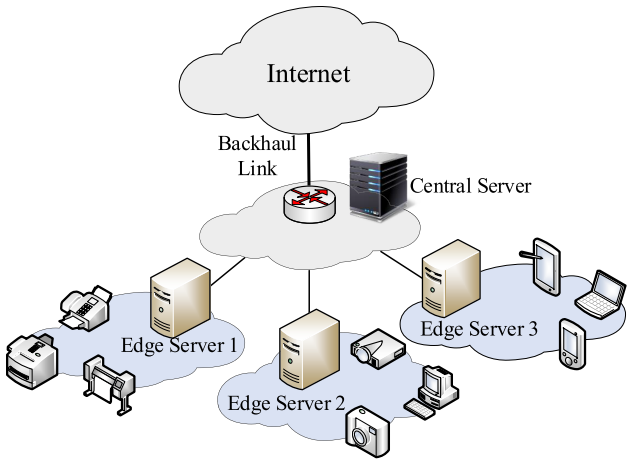


FIGURE 1. Cooperative IoT edge caching system model.

The workflow of the proposed system is described as follows. 1) An IoT data request is first sent to its assigned edge servers. 2) If the local cache has the relevant data item, it returns the requested item immediately; otherwise, the request is reported to the central server. 3) The central server queries whether other edge servers have the requested data item. If it exists, it will be sent back through the internal connections between edge servers; otherwise, the central server fetched the data item from the Internet through the backhaul link.

Usually, the primary goal of a caching server is to maximize its hit rate to minimize the traffic cost incurred by data transmission. For the proposed system, maximizing the hit rate of an individual edge server would be inferior to maximizing the total hit rate of all servers, as the latter could enable edge servers to cache more data items and thus achieve a larger backhaul traffic reduction. Nevertheless, improving the total hit rate requires the edge server to be enabled to adjust its caching decisions based on other servers' caching states. To this end, there is a need to efficiently exchange the caching information, including request feature and caching states, between edge servers. Although cooperative edge caching may cause additional traffic overhead due to caching information interaction, still it can significantly improve the total hit rate and reduce the overall backhaul traffic load. Furthermore, the traffic overhead incurred is negligible compared with fetching a data item from the Internet.

Towards making a caching decision, there are usually two methods to develop a caching decision: Centralized and Decentralized. For the centralized method, caching decisions are determined in the central server and then sent to edge servers. In this mode, edge servers are only responsible for data storage and caching decision execution, and the central server provides optimal caching decisions based on the global view of caching states. However, for the decentralized method, it is the edge servers that determine the caching decisions. In this mode, an edge server receives other servers' caching information and then determines its own caching decisions based on local data requests. The center server works for caching information interaction and synchronization.

In the rest of this paper, $E = \{e_1, e_2, \ldots, e_N\}$ is used to denote the set of Edge Server, where e_n represents the cache capacity of Server n. Here N represents the number of Edge Servers. The set of IoT data item is denoted by $C = \{c_1, c_2, \ldots, c_M\}$, where c_m represents the size of the data item m, here M is the total number of IoT data items. In addition, time is assumed to be slotted into periods, denoted by $\mathcal{T} = \{1, 2, \ldots, T\}$. Then, caching decisions is determined and updated periodically as well as caching information.

In the domain served by e_n, the number of IoT application is assumed to be L_n and requests generated by l at time t is denoted by a binary vector $U^t_{n,l} = \{u^t_{n,l,1}, u^t_{n,l,2}, \ldots, u^t_{n,l,M}\}$, where $u^t_{n,l,m} = 1$ means application l requests data item m at time t while $u^t_{n,l,m} = 0$ otherwise. Here, the popularity of the IoT data item is assumed to be unknown. Instead, $F^t_{n,l} = \{f^t_{n,l,1}, f^t_{n,l,2}, \ldots, f^t_{n,l,M}\}$ is introduced to record the request frequency of data items. Specifically, $f^t_{n,l,m}$ represents the cumulative number of requests for data item m by application l at time t. In addition, data requests are assumed to be independent.

B. PROBLEM FORMULATION

Based on the above assumptions, the cooperative caching problem is formulated as an optimization problem with a goal of maximizing the long time averaged backhaul traffic reduction.

To describe a caching decision, this work introduces a binary matrix $X^t = \{x^t_{m,n} | c_m \in C, e_n \in E\}$, where $x^t_{m,n} = 1$ indicates that the data item m is cached in the edge server e_n at time t, while $x^t_{m,n} = 0$, otherwise. Since the total size of the cached data items cannot exceed the storage capacity of the edge server, the capacity constraints is

$$\sum_{m=1}^{M} c_m \cdot x^t_{m,n} \leq e_n \tag{1}$$

In general, the backhaul traffic reduction is proportional to the frequency of requests for the cached data items. It is calculated as the accumulated weighted sum of the requested data items. When considering the proposed model, it is divided into two parts: local caching traffic reduction and cooperation caching traffic reduction. For the first part, the backhaul traffic reduction is achieved by the cached data items in local edge server.

$$r^t_{n,0} = \sum_{m=1}^{M} \sum_{l=1}^{L_n} f^t_{n,l,m} \cdot u^t_{n,l,m} \cdot c_m \cdot x^t_{m,n} \tag{2}$$

For the second part, it is achieved by other servers sharing their data items, which is expressed by

$$r^t_{n,1} = \sum_{m=1}^{M} \sum_{l=1}^{L_n} f^t_{n,l,m} \cdot u^t_{n,l,m} \cdot c_m \cdot (1 - x^t_{m,n}) \cdot y^t_n \tag{3}$$

where $y^t_n = (1 - \prod_{n`=1, n` \neq n}^{N} (1 - x^t_{m,n`}))$

Although sharing data items reduces the backhaul traffic load, it causes an additional traffic overhead. Therefore, a discount factor, $\beta_n \in [0, 1]$, is introduced. Now, the expected traffic reduction is

$$R^t = \sum_{n=1}^{N} (r^t_{n,0} + \beta_n \cdot r^t_{n,1}) \tag{4}$$

Consider Eq. (4), $\beta_n = 0$ means that internal traffic is equivalent to the backhaul traffic. Under this circumstance, cooperation between servers is meaningless. On the contrary, $\beta_n = 1$ implies that the internal connection traffic is ignored. Finally, there is no difference between caching a data item locally and on other servers.

Based on the above discussion, the caching problem with the objective of maximizing backhaul traffic reduction is formulated as follows:

$$\begin{aligned} \max \ & \frac{1}{T} \sum_{t=1}^{T} R^t \\ s.t. \ & \sum_{m=1}^{M} c_m \cdot x^t_{m,n} \le e_n, \quad \forall c_m \in C, \ \forall e_n \in E \\ & \sum_{m=1}^{M} x^t_{m,n} \le 1, \quad \sum_{n=1}^{N} x^t_{m,n} \le N, \ x^t_{m,n} \in \{0, 1\} \\ & 1 \le m \le M, \quad 1 \le n \le N, \ 0 \le \beta_n \le 1 \end{aligned} \tag{5}$$

$\sum_{m=1}^{M} x^t_{m,n} \le 1$ and $\sum_{n=1}^{N} x^t_{m,n} \le N$ constrains that the edge server is not allowed to cache duplicate data items.

Although the above optimization problem could provide an optimal caching decision, it requires a lot of computing resources to obtain the exact solution of X^t. This is because the above problem is NP-hard. To prove this, a simple case of problem (5) is considered. Let $\beta_n = 0$, means there is no cooperation between edge servers. Then, the reduced problem is divided into n independent sub-problems. For one sub-problem, it can be understood as follows. There are M data items of different sizes and the objective of an edge server is to select a portion of the data items for storage to maximize the weighted sum of the cached data without exceeding its capacity. Obviously, the sub-problem is converted to a knapsack problem [46]. Since the knapsack problem is known to be NP-hard, the problem (5) is also NP-hard.

In the following section, a deep reinforcement learning approach is proposed to cope with the cooperative caching problem, in which edge servers could learn to adjust caching decisions according to other servers' states and actions.

IV. DEEP REINFORCEMENT LEARNING-BASED COOPERATIVE CACHING APPROACH

In this section, a deep reinforcement learning-based cooperative caching framework is first proposed by combining the characteristics of the deep learning approach and the cooperative edge caching system. Then, the feature and action spaces are defined to cope with the obstacles brought by inconsistent IoT data item size, and a reward function is designed to promote cooperation between edge servers.

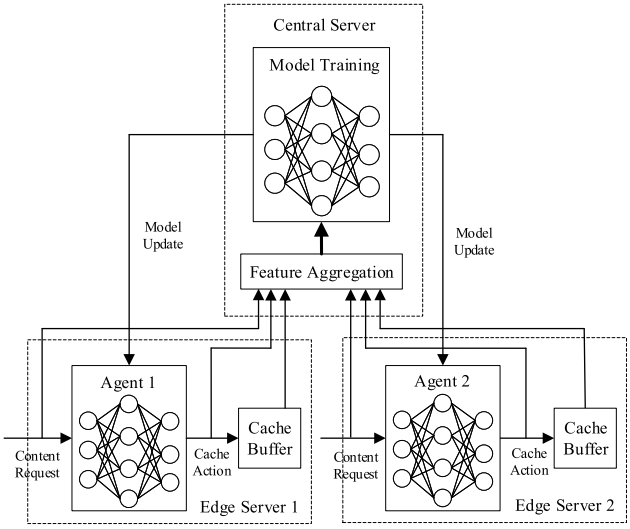


FIGURE 2. Deep Reinforcement Learning-Based Cooperative Caching Framework.

A. COOPERATIVE CACHING FRAMEWORK

As discussed in Section III. A, there are usually two methods to develop a caching decision: Centralized and Decentralized. However, both methods will incur new problems. For the centralized method, the deployment of the caching decision would cause an additional delay, as caching decisions are not determined locally; for the decentralized method, the issue of how to exchange the caching information between edge servers needs to be addressed, as it has a severe impact on executing cooperation between edge servers.

To alleviate the above problems, this work proposes a cooperative caching framework with centralized training and distributed operations by leveraging the deep reinforcement learning based approaches, as shown in Fig. 2.

In the proposed framework, each edge server has an agent and a cache buffer. The agent extracts the features of data requests and determines the cache action locally. The cache buffer performs the cache action and stores the requested data items. Request features, cache actions, and buffer states of all edge servers are periodically reported to the central server, and the model in each agent is updated depending on the parameters sent by the central server.

With this framework, cooperation between edge servers can be achieved with information exchanging overheads significantly reduced. This is mainly because the parameters, send by the central server, are treated as the indicators of other servers' caching information and these parameters would cost lower transmission overheads. Also, the central server could include the cooperation reward in the parameters with cooperation promoted.

The workflow is discussed as follows. When data requests arrive at edge servers, they make caching decisions based on their current states. Then, they report their

caching information, including actions, states, and requests, to the central server. The central server evaluates the actions and trains models. Following this, the central server sends the parameters to each edge server to update their models.

We believe that, the proposed caching framework is naturally compatible with the commonly used DRL algorithms, such as DQN (Deep Q-Network) [47], Actor-Critic [21], and DDPG (Deep Deterministic Policy Gradient) [24]. These DRL algorithms have one thing in common. That is, they hold multiple neural networks. For example, the basic Actor-Critic algorithm has two neural networks, one determines acts, and the other evaluates these acts. By leveraging the above framework, the actor network can be deployed in the edge server and the critic network can be trained in the central server. In this way, the central server could send TD-errors to the edge servers for actor network update. Other DRL-based approaches with the same characteristics can also be similarly implemented in this framework.

B. FEATURE SELECTION, ACTION ANALYSIS, AND REWARD DESIGN

The proposed framework provides a clear vision to perform cooperative edge caching, however, there is still a key challenge that needs to address. That is how to define the feature, the action, and the reward.

1) FEATURE SELECTION

This work considers a complex but practical scenario, where the size of the IoT data item is not constant. This poses an obstacle in defining the feature space. The essential reason lies in the difficulty to describe the caching state with a fixed-length vector.

A simple solution is to use the pre-defined binary matrix $X^t = \{x^t_{m,n} | c_m \in C, e_n \in E\}$. However, this approach brings two problems. First, the total number of data items needs to be known in advance, which is almost impractical. Second, using X^t would make DRL algorithms inefficient in making caching decisions, due to its large-scale space.

To tackle the above problems, this work divides the cache buffer of e_n into K_n pieces, where $K_n = \lceil e_n/(\min(C)) \rceil$. Then, the caching state of e_n is described by $S_n = \{s_1, s_2, \ldots, s_{K_n}\}$, in which $s_k = \{\delta_k, \varphi_k\}$. k denotes the location in the cache buffer, δ_k indicates the cached data item, and φ_k represents its size. When $\delta_k = 0$, it means there is no data item and φ_k equals to zero. In this way, the state of a cache buffer is represented by a fixed-length vector whose length is related to the cache size.

In addition to the caching state, the features of data items also need to be defined. Consider conventional caching heuristics, which usually treat the data item size, request frequency, and request interval as features and use them to make caching decisions. For example, LRU (Least Recently Used) determines whether to cache a data item based on its request interval and LFU (Least Frequently Used) tends to store data items with large request frequency. This work also incorporates these features. Thus, the feature of a data item is denoted by $q_m = \{\delta_m, \varphi_m, \omega_m, \mu_m\}$, where δ_m is the indicator of the data item m and φ_m, ω_m and $\mu_m$represent its size, request frequency, and request interval respectively. Note that, $\varphi_m = c_m$ and $\omega_m = \sum_t \sum_l f^t_{n,l,m}$.

TABLE 1. Notations of data item feature.

Notation	Description
S_n	The cache state of edge server n
s^t_k	The feature of kth data item in the edge server at t
q^t_m	The feature of requested data item m
δ_k	The indicator of data item m
φ_k	The size of data item m
ω_m	The total request frequency of data item m
μ_m	The request interval of data item m
$\rho^t_{m,s}$	Short-term request frequency of data item m at t
$\rho^t_{m,m}$	Medi-term request frequency of data item m at t
$\rho^t_{m,l}$	Long-term request frequency of data item m at t

To better describe a data item, this work also introduces three additional features, similar to [26]: short-term request frequency $\rho^t_{m,s}$, medium-term request frequency $\rho^t_{m,m}$, and long-term request frequency $\rho^t_{m,l}$. These variables are useful for characterizing the popularity and can be updated as requests arrive. They are calculated as follows:

$$\rho^t_{m,s} = \sum_{t-\tau_s}^{t} \sum_l f^t_{n,l,m} \tag{6}$$

$$\rho^t_{m,m} = \sum_{t-\tau_m}^{t} \sum_l f^t_{n,l,m} \tag{7}$$

$$\rho^t_{m,l} = \sum_{t-\tau_l}^{t} \sum_l f^t_{n,l,m} \tag{8}$$

Therefore, the features of a data item can be described as $q^t_m = \{\delta_m, \varphi_m, \omega_m, \mu_m, \rho^t_{m,s}, \rho^t_{m,m}, \rho^t_{m,l}\}$ and the caching state is $s^t_k = \{\delta_k, \varphi_k, \omega_k, \mu_k, \rho^t_{k,s}, \rho^t_{k,m}, \rho^t_{k,l}\}$. The features used in this work are listed in Table 1.

2) ACTION ANALYSIS

In this work, the output of the DRL algorithm is treated as the cache action, and it faces a similar problem with the cache feature.

Specifically, if the cache action is defined by using the caching state, the action space varies w.r.t the size of the requested data item as well as the current caching states. For example, when the requested data item size is $\min(C)$, the action space is described by $A_n = \{0, 1, 2, \ldots, K_n\}$, where $a_n = k$ means that the kth data item in the cache buffer is replaced by the requested data item. However, when the requested data item size is $2 \times \min(C)$, the above action space only works in the case that all cached data items are no smaller than the requested. Otherwise, the action for storing the requested item must be represented by a pair of a_n, which means the edge server has to drop out up to two data items to store the requested one. Under this situation, the size of the action space is extended to $K_n + K_n \cdot (K_n - 1)/2 + 1$.

Furthermore, to store a data item of size c_m, the number of actions can be calculated by

$$\Omega^m_{action} = \sum_{\kappa=1}^{\kappa_m} \binom{K_n}{\kappa} \tag{9}$$

where $\kappa_m = \lceil c_m/(\min(C)) \rceil$.

Thus, the total size of the action space is as follows.

$$\Omega_{action} = \sum_{m} \sum_{\kappa=1}^{\kappa_m} \binom{K_n}{\kappa} + 1 = \sum_{\kappa=0}^{\eta} \binom{K_n}{\kappa} \tag{10}$$

where $\eta = \lceil (\max C)/(\min(C)) \rceil$.

It is obvious that the action space grows exponentially with the cache size. Thus, it is impractical to define the cache action when the cache size is large. Moreover, Ω_{action} only represents the number of all possible actions, including considerable amount of invalid actions, which may increase the difficulty of choosing an optimal cache action.

To cope with the above issues, this work also develops a caching policy selection scheme. In particular, the output of the model specifies a simple caching policy instead of a caching replacement behavior, and then, the actual caching replacement behavior is determined by the selected policy. In other words, the DRL algorithm is used to select a caching policy instead of making caching decisions.

By this means, the action space is greatly reduced, and we denote this by a fixed-length vector $A_n = \{p_1, p_2, \ldots, p_n\}$, where $a_n = p_n$ means that the edge server would execute p_n policy to determine its caching decisions. Besides, the size of the action space also depends on the number of policies deployed in the edge server.

The following example illustrates how the proposed scheme works. When a data request's features are fed into the model, an output is taken, which represents that a certain caching policy is deployed in the edge server. The cache buffer operates this policy to determine whether and how to cache it, further also to update the buffer state.

The proposed scheme have further significant advantages. 1) All actions are valid and model training can be accelerated. 2) The worst-case performance can be guaranteed.

In this work, to simplify, the caching policies are defined by the features $A_n = \{\varphi, \omega, \mu, \rho_s, \rho_m, \rho_l\}$. An ω policy indicates that the cache buffer determines its caching replacement behavior based on the request frequency of a data item. In particular, the buffer sorts the cached and the requested data items based on their cumulative requests and kicks off the least one. Other policies work in a similar method.

3) REWARD DESIGN

In our work the cooperation among edge servers is extremely important and challenging. To this end, we note that giving rewards is one of the good approach but that needs to be carefully designed.

For a caching system, the hit rate is usually selected as the reward to present the primary objective. It is often set to be the number of requests for the cached data items in the next epoch. Nevertheless, for the proposed system, the data items cached in the local server and the neighboring servers both contribute to the system hit rate. Thus, the reward can be set as the weighted sum of the both to promote cooperation between edge servers:

$$g^t_n = \nu \cdot h^t_n + \upsilon \cdot o^t_n \tag{11}$$

where h^t_n denotes the number of requested data items in the local server and o^t_n denotes that in neighboring servers. ν and υ are the weights to balance the rewards gained by the local server and neighboring servers. When υ is zero, it implies that the edge servers works without cooperation.

However, in this work, (11) may not embody the objective well. It is mainly because the backhaul traffic reduction is affected by both hit rate and data item size. Particularly, caching a data item with a large hit rate and small size may not reduce more traffic than caching a data item with a bit lower hit rate and larger size. And this implies that both the hit rate and data item size must be taken into consideration when designing the reward function.

Therefore, this work incorporates the size of a data item into the reward.

$$g^t_n = \nu \cdot \sum_{k} h^t_{n,k} \cdot \varphi_k + \upsilon \cdot \sum_{k`} o^t_{n,k`} \cdot \varphi_{k`} \tag{12}$$

where $h^t_{n,k}$ is the number of requests for the locally cached data item k and $o^t_{n,k`}$ is the number of requests for the data item $k`$ stored in neighboring servers. With this design, cooperation between edge servers can be dynamically adjusted by ν and υ. When ν is large, the edge server would pay more attention to its local performance. And when υ becomes large, it would prefer to assist its neighbors.

V. MULTI-AGENT ACTOR-CRITIC ALGORITHM FOR COOPERATIVE EDGE CACHING

In this section, a multi-agent actor-critic caching algorithm, Algorithm 1, is proposed to tackle the cooperative caching issue, which is inspired by the Actor-Critic algorithm [21].

The Actor-Critic algorithm provides a learning framework with two separate neural networks, where one is for action output, namely the actor network, and the other is for action evaluation, namely the critic network. As illustrated in Section IV. A, it is compatible with the proposed framework, as the actor network can be deployed in the edge server for caching policy selection and the critic network can be in the central server for caching decision evaluation. Further, this work extends the Actor-Critic algorithm to multi-agent scenarios, where other servers' caching information is considered when evaluating caching decisions. Then, the TD-errors used to update the actor network is treated as the indicators of other servers' caching information and calculated with the purpose of promoting cooperation.

In the proposed algorithm, the actor network is deployed in the agent of the edge server and the critic network is in the central server.

Algorithm 1 Multi-Agent Actor-Critic Algorithm for Cooperative Edge Caching

Initialize: the cache buffer sate $S_n = \{s_1, s_2, \ldots, s_{K_n}\}$; the actor network $\pi_n(S_n|\theta_n^\pi)$ with weights θ_n^π; the critic network $Q_n(S_1, S_2, \ldots, S_n, a_n|\theta_n^Q)$ with weights θ_n^Q; the replay buffer M_n.
for $t = 1, T$ **do**
  Each edge server observes the caching state S_n^t.
  For edge server e_n, select action $a_n = \pi_n(S_n|\theta_n^\pi)$ w.r.t. the current actor network.
  Execute a_n^t, observer new state S_n^{t+1} and send $(S_n^t, a_n^t, S_n^{t+1})$ to the central server.
  for $n = 1, N$ **do**
    The central server calculates the reward g_n^t for e_n and stores $(S_n^t, a_n^t, g_n^t, S_n^{t+1})$ in M_n.
    Sample a random minibatch of I transitions $(S_n^i, a_n^i, g_n^i, S_n^{i+1})$ from M_n.
    Set $y_n^i = g_n^i + \gamma Q`_n(S_1^{i+1} \ldots S_N^{i+1}, \pi_n(S_n^{i+1})|\theta_n^{Q`})$
    Calculate the TD-error based on the current parameter $\phi_n = \frac{1}{I}\sum_i (y_n^i - Q_n(S_1^i \ldots S_N^i, a_n^i|\theta_n^Q))$
    Update the critic network Q_n by minimizing the loss: $Loss_n = (\phi_n)^2$.
    Send ϕ_n to edge server e_n.
  end for
  Each edge server updates the actor network by maximizing $\nabla_{\theta_n^\pi} J = \nabla_{\theta_n^\pi} \log \pi_n(S_n, a_n)\phi_n$
  Update cache state.
end for

Actor network: The input of the actor network includes the caching state and request features of the local edge server. The actor network is noted by θ_n^π and it is defined as a function to seek the optimal caching policy, which maps the caching state S_n to an action a_n. In each decision epoch, edge servers choose their actions based on the current parameters and the caching state.

$$a_n^t = \pi_n(S_n^t|\theta_n^\pi) \tag{13}$$

Critic network: The critic network is mainly used for estimating the values of the selected actions. The input of the critic network contains all edge servers' caching states $(S_1, S_2, \ldots, S_N)$ and the local server's action a_n. The basic idea of this design lies in the consideration that the action value of an edge server is determined by the overall system state and its action. Then, the critic network calculates the TD-error, ϕ_n by

$$\phi_n = \frac{1}{I}\sum_i (y_n^i - Q_n(S_1^i, \ldots, S_N^i, a_n^i|\theta_n^Q)) \tag{14}$$

$$y_n^i = g_n^i + \gamma Q`_n(S_1^{i+1}, \ldots, S_N^{i+1}, \pi_n(S_n^{i+1})|\theta_n^{Q`}) \tag{15}$$

where γ is the discount factor of the future accumulated reward with the range of (0, 1), I is the size of minibatch, and θ_n^Q denotes the parameters of the critic network.

Update: The critic network is updated by minimizing the loss function.

$$Loss_n = (\phi_n)^2 \tag{16}$$

And the actor network is updated by the policy gradient.

$$\nabla_{\theta_n^\pi} J = \nabla_{\theta_n^\pi} \log \pi_n(S_n, a_n)\phi_n \tag{17}$$

The workflow of the proposed algorithm is discussed as follows. The edge server receives its data requests and extracts the features. Then, it feeds the current caching state and request features into the actor network to obtain the caching action. After executing the action, the edge server sends the current caching state, the next caching state, and the caching actions to the central server. At the central server, the critic network assigned to the edge server calculates the reward and the TD-error of the current caching action. By minimizing the loss function, the critic network is updated. And by sending the TD-error to the edge server, the actor network is updated.

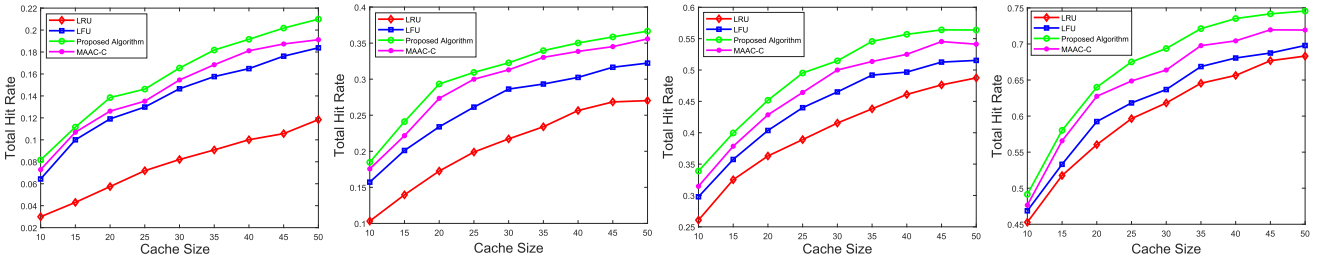
Note that, the proposed algorithm can be implemented in the following way to support online work and offline training. Specifically, all the actor and critic networks of edge servers are deployed in the central server, and each edge server holds a duplicate of its assigned actor network. Then, the training of each model can be operated in the central server with the aggregated caching information in an offline manner. For online work, each edge server can observe its caching states and obtain the selected caching policy from its local actor network. With this implementation, when the model needs to be updated, the local actor network parameters can be replaced by that in the central server.

VI. SIMULATION RESULTS

In this section, the performance of the proposed caching algorithm is validated by comparing it with two conventional algorithms and one state-of-the-art learning-based algorithm, which are described as follows.

- LRU: LRU algorithm determines its cache action based on the arrival time of the request. It keeps track of the most recent requests. When the cache buffer is full, it replaces the least requested data item with the new one.
- LFU: LFU algorithm determines its cache action based on the request frequency. It prefers to store data items with a large request frequency. When the cache buffer is full, it replaces the least requested data item with the newly requested one.
- Multi-Agent Actor-Critic algorithm with Centralized Critic [22]: This algorithm employs a multi-agent framework with a centralized critic network. Caching decisions are determined based on the observation of each agent. All caching states are inputted into the critic network to calculate the TD-error and all actor networks are updated using the same TD-error. In the following text, this algorithm is also referred as MAAC-C.

Note that, although both the MAAC-C and the proposed algorithm employ a multi-agent framework, there still exist



(a) $\alpha = 0.7$ (b) $\alpha = 0.9$ (c) $\alpha = 1.1$ (d) $\alpha = 1.3$

FIGURE 3. Total Hit Rate vs Cache Capacity, where Zipf parameter, α, varies as {0.7, 0.9, 1.1, 1.3} and cache capacity ranges from 10 to 50.

three key differences. First, the MAAC-C contains only one critic network and all actor networks are updated by using the same TD-error. However, in the proposed algorithm, there are multiple critic networks, each corresponds to an actor network. Second, the MAAC-C assumes that the data items are of the same size. However, the proposed algorithm addresses the issue that the size of the data item is not a constant. Third, the feature, the action, and the reward are different with different purposes, where MAAC-C attempts to minimize the transmission delay and the proposed algorithm has an objective of maximizing backhaul traffic reduction.

For comparison, MAAC-C is modified to adapt to the proposed system in this work, and its goal is also changed to be consistent with the proposed algorithm.

A. SIMULATION SETUP AND PARAMETER SETTINS

In this work, the default number of edge servers is set to 3, as shown in Fig. 2. The cache size of each edge server varies from 10 to 50 units. Data requests are generated according to a Zipf distribution with a parameter α, which varies as 0.7, 0.9, 1.1, 1.3. The total number of the data items is assumed to be 2000 and the size of each data item varies from 1 to 5 units randomly. The backhaul traffic cost is assumed to be the accumulated data item size fetched through the backhaul link. The episode length is set to 5000 in each simulation and each data point in the figures is averaged based on 100-times simulation.

For the caching algorithm, both the actor network and the critic network have two hidden layers using ReLu as the activation function. The inputs of the actor network are the features selected in Section IV. B, and the input of the critic network includes all the features and the actions of the three edge servers. The additional short-term, medium-term, and long-term features are set to the request frequency within the most recent 10, 50, 100 requests.

B. HIT RATE

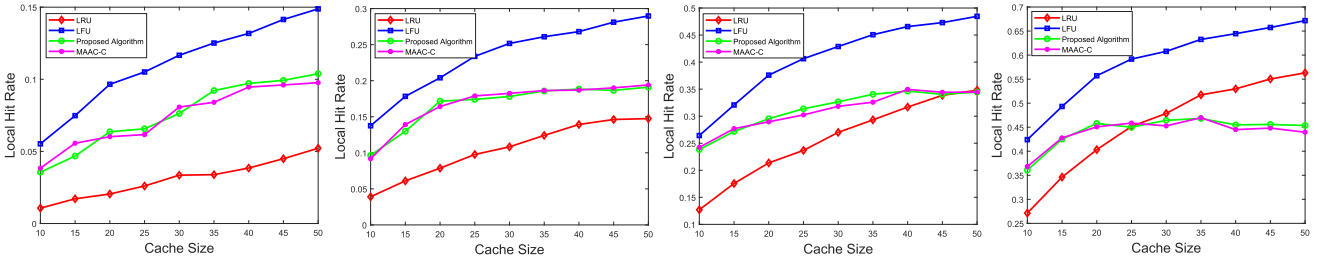
In this subsection, the hit rate of the proposed algorithm is evaluated based on the simulation results. Moreover, the local hit rate and the other hit rate are presented to illustrate how the edge servers cooperate. The local hit rate is achieved by caching data items in the local server and the other hit rate denotes the requests satisfied by neighboring edge servers.

Fig. 3 illustrates how the total hit rate varies with the cache size under different Zipf parameters. As shown in Fig. 3, the hit rate of all algorithms increases as the cache size increases, and the learning-based algorithms, the proposed algorithm, and MAAC-C, outperforms the conventional algorithms. This is mainly because the conventional algorithms only use a single feature for caching decisions, but the learning-based algorithms combine more features, which captures various aspects of a data item to make effective caching decisions. Besides, the performance of MAAC-C is better than LFU and LRU, and this is consistent with [22].

For the learning-based algorithms, the proposed algorithm has better performance, when compared with MAAC-C. The reasons can be explained as follows. The MAAC-C algorithm has only one critic network and all actor networks are updated by the same TD-error. This may promote cooperation between edge servers. However, this may not be conducive to maintaining the independence of each edge server, which could make the edge server to tend to store data items with similar features, and may lead to a reduction in total hit rate.

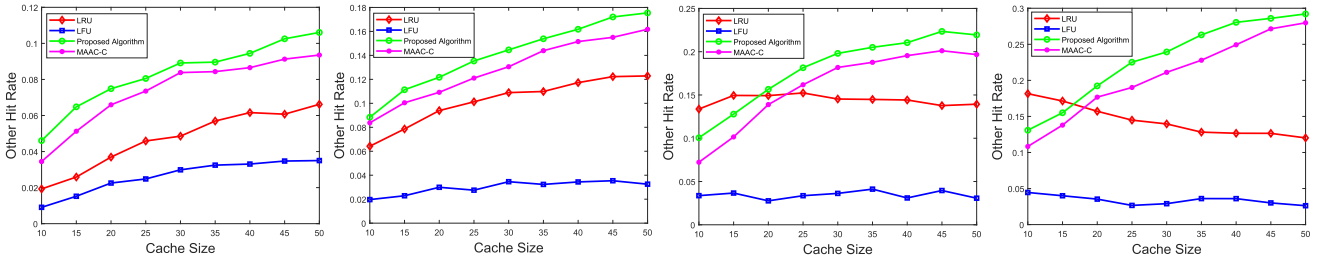
Another observation from Fig. 3 is that the hit rate of all algorithms increases with the increase in the Zipf parameter α. This can be understood easily. When α is large, the requested data items become concentrated. In this situation, all algorithms prefer to store the data item with a large request frequency resulting in an increase in the total hit rate. In addition, the gap between LRU and LFU becomes small as α raises. However, the gap between learning-based algorithms and LFU remains almost constant. This scenario can be further explained as follows. Since both the LRU and LFU prefer to store the most popular data items, they would like to make similar caching decisions. Thus, the gap between them becomes small. As for the learning-based algorithms, both α and cooperation have an impact on caching decisions, which results in a relatively steady improvement in the total hit rate. Besides, cooperation between edge servers implies that each edge server may sacrifice a partition of the local hit rate in exchange for the total hit rate increment. And this can be validated in Fig. 4 and Fig. 5.

As a supplement to Fig. 3, Fig. 4 and Fig. 5 is used to illustrate that the learning-based approaches could achieve a higher hit rate. Fig. 4 shows the local hit rate of the edge server, which represents the number of requests satisfied by the local edge server. In Fig 4. (a), the local hit rates of all



(a) $\alpha = 0.7$ (b) $\alpha = 0.9$ (c) $\alpha = 1.1$ (d) $\alpha = 1.3$

FIGURE 4. Local Hit Rate vs Cache Capacity, where Zipf parameter, α, varies as {0.7, 0.9, 1.1, 1.3} and cache capacity ranges from 10 to 50.



(a) $\alpha = 0.7$ (b) $\alpha = 0.9$ (c) $\alpha = 1.1$ (d) $\alpha = 1.3$

FIGURE 5. Other Hit Rate vs Cache Capacity, where Zipf parameter, α, varies as {0.7, 0.9, 1.1, 1.3} and cache capacity ranges from 10 to 50.

algorithms increase as the cache size increases, and LFU has the highest local hit rate, followed by the learning-based algorithms, and then LRU. However, in Fig. 4 (b), the local hit rate of learning-based algorithms first raises and then remains almost constant. In Fig. 4 (c), the learning-based algorithms show the same trend. But after the cache sizes grow to 45, there is almost no difference between LRU and learning-based algorithms. Moreover, in Fig. 4 (d), the local hit rate of LRU is higher than learning-based algorithms as the cache size grows larger than 25.

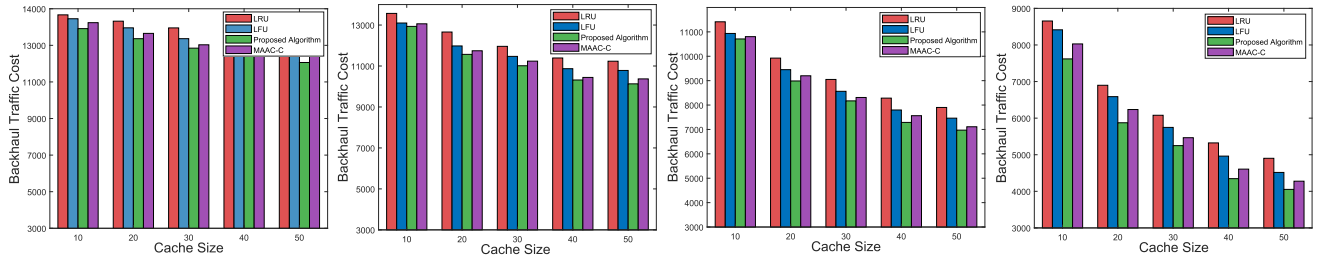
Fig. 5 shows the other hit rate. It can reflect the cooperation between edge servers. From Fig. 5 (a) and (b), the other hit rate increases as the cache size grows. However, in Fig. 5 (c), the other hit rate of LFU and LRU remains almost constant while learning-based approaches continue to grow. Nevertheless, in Fig. 5 (d), the other hit rates of LFU and LRU decrease with the increase of cache size. In addition, both Fig. 5 (c) and (d) display the phenomenon that initially the other hit rates of learning-based algorithms are lower than LRU, but they continue to grow as cache size raises.

Based on the above illustrations, there are some findings that help to understand the reasons why learning-based algorithms outperform LRU and LFU. 1) As shown in Fig. 4 and Fig. 5, the local hit rates of LFU are always the highest, while the other hit rates are the lowest. This is mainly because LFU pays more attention to the efficiency of the edge server itself and ignores the cooperation between edge servers. In contrast, learning-based algorithms learn to work cooperatively. They sacrifice their local hit rate in exchange for the improvement of the total hit rate. 2) The other hit rates of LRU and LFU remain constant in Fig. 5 (c) and decrease in Fig. 5 (d). This is due to the tendency of edge servers to store same data item when *alpha* is large. Once the cache size is large enough, there will be a large overlap between the cached data items. Thus, requests that cannot be satisfied locally are also difficult to be served in neighboring edge servers. However, different from LRU and LFU, the learning-based algorithms leverage the global view to make caching decisions, and this makes them avoid the aforementioned situations. 3) The local hit rates of LFU are always higher than that of LRU in Fig. 4, while the other hit rates of LFU are always lower in Fig. 5. This phenomenon is mainly due to the randomicity of caching decisions. When compared with LFU, LRU has stronger randomicity, which promotes the whole system to store different items leading to an increment in the other hit rate. In addition, this can also be used to explain why learning-based algorithms show better performance.

In addition, in Fig. 4, the local hit rates of MAAC-C and the proposed algorithm are almost the same. The reason is that both algorithms make cache decisions locally, based on the current cache state of each edge server. Moreover, from Fig. 4 (d) and Fig. 5 (d), it can be observed that when the cache size is small, all edge servers choose to maximize the local hit rates. However, as cache size increases, they pay more attention to cooperate with each other. This scenario implies that both algorithms tend to improve system performance by cooperation. Further, the local hit rates of MAAC-C and the proposed algorithm are almost the same in Fig. 4, but the other hit rate of the proposed algorithm is higher than that of MAAC-C in Fig. 5. This phenomenon convinces the reason why the proposed algorithm outperforms MAAC-C.

C. BACKHAUL TRAFFIC COST

In addition to the hit rate, the backhaul traffic cost is also presented to validate the performance of the proposed algorithm in this subsection, as shown in Fig. 6.



(a) $\alpha = 0.7$ (b) $\alpha = 0.9$ (c) $\alpha = 1.1$ (d) $\alpha = 1.3$

FIGURE 6. Other Hit Rate vs Cache Capacity, where Zipf parameter, α, varies as {0.7, 0.9, 1.1, 1.3} and cache capacity ranges from 10 to 50.

From Fig. 6, it is seen that the proposed algorithm consumes the least amount of the backhaul traffic, followed by MAAC-C, then the LFU, and finally LRU. Besides, all subfigures in Fig. 6 show a trend that the backhaul traffic cost decreases as cache size increases. These results imply that the backhaul traffic cost can be reduced by improving the hit rate.

Moreover, Fig. 6 also shows that the backhaul traffic cost of all algorithms decreases as the Zipf parameter, α, increases. This phenomenon is attributed to the improvement of the hit rate. As α increases, more requests could be satisfied, which would result in an increment in the hit rate and a reduction in the backhaul traffic cost.

Another observation noted from Fig. 6 is the gap between learning-based algorithms and the conventional algorithms becomes larger as α grows larger. However, in Fig. 3, the gap between them remains almost constant. It is mainly due to the fact that the requested data items become concentrated as α becomes larger. In this case, the higher the hit rate is, the less the traffic would cost. In addition, this situation also implies that the cooperation between edge servers may bring more benefits when the hit rate is high.

VII. CONCLUSION

In this work, we proposed a deep reinforcement learning based cooperative caching approach for IoT edge caching. The caching states and the caching actions were defined to cope with the obstacles brought by inconsistent data item size. A reward function was designed to promote cooperation between edge servers. On this basis, a multi-agent actor-critic cooperative edge caching algorithm was developed to conduct the proposed approach. Simulation results confirmed the performance improvement using our proposed algorithm over one state-of-the-art learning-based algorithm and two conventional algorithms, and provided a view in explaining the superior performance brought by cooperative edge caching.

In the future, there are still several issues to be investigated. First, it should be discussed how to take other factors, such as user preference, into consideration. Besides, more situations need to be explored, such as how to make edge servers cooperate to perform computing tasks. In addition, the proposed approach will be implemented in a real system and verified by the real-life workload.

REFERENCES

[1] *Cisco Annual Internet Report (2018–2023) White Paper*. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive–perspectives/annual-internet-report/hite-paper-c11-741490.html

[2] [Online]. Available: https://www.statista.com/statistics/1017863/worldwide-iot-connected-devices-data-size/

[3] B. Feng, H. Zhou, H. Zhang, G. Li, H. Li, S. Yu, and H.-C. Chao, "HetNet: A flexible architecture for heterogeneous satellite-terrestrial networks," *IEEE Netw.*, vol. 31, no. 6, pp. 86–92, Nov. 2017.

[4] H. Zhang, W. Quan, H.-C. Chao, and C. Qiao, "Smart identifier network: A collaborative architecture for the future Internet," *IEEE Netw.*, vol. 30, no. 3, pp. 46–51, May 2016.

[5] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[6] W. Quan, K. Wang, Y. Liu, N. Cheng, H. Zhang, and X. Shen, "Software-defined collaborative offloading for heterogeneous vehicular networks," *Wireless Commun. Mobile Comput.*, vol. 2018, pp. 1–9, Apr. 2018.

[7] B. Perabathini, E. Bastug, M. Kountouris, M. Debbah, and A. Conte, "Caching at the edge: A green perspective for 5G networks," in *Proc. IEEE Int. Conf. Commun. Workshop (ICCW)*, London, U.K., Jun. 2015, pp. 2830–2835.

[8] K. Peng, B. Zhao, S. Xue, and Q. Huang, "Energy-and resource-aware computation offloading for complex tasks in edge environment," *Complexity*, vol. 2020, pp. 1–14, Mar. 2020.

[9] J. Yao, T. Han, and N. Ansari, "On mobile edge caching," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2525–2553, 3rd Quart., 2019.

[10] D. Liu and C. Yang, "Energy efficiency of downlink networks with caching at base stations," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 4, pp. 907–922, Apr. 2016.

[11] S. Zhang, P. He, K. Suto, P. Yang, L. Zhao, and X. Shen, "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Trans. Mobile Comput.*, vol. 17, no. 8, pp. 1791–1805, Aug. 2018.

[12] L. Ramaswamy, L. Liu, and A. Iyengar, "Cache clouds: Cooperative caching of dynamic documents in edge networks," in *Proc. 25th IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS05)*, Columbus, OH, USA, Jun. 2005, pp. 229–238.

[13] W. Wang, D. Niyato, P. Wang, and A. Leshem, "Decentralized caching for content delivery based on blockchain: A game theoretic perspective," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Kansas City, MO, USA, May 2018, pp. 1–6.

[14] W. Jiang, G. Feng, and S. Qin, "Optimal cooperative content caching and delivery policy for heterogeneous cellular networks," *IEEE Trans. Mobile Comput.*, vol. 16, no. 5, pp. 1382–1393, May 2017.

[15] M. Zhang, Y. Zhou, W. Quan, J. Zhu, R. Zheng, and Q. Wu, "Online learning for IoT optimization: A Frank-Wolfe Adam based algorithm," *IEEE Internet Things J.*, early access, Mar. 30, 2020, doi: 10.1109/JIOT.2020.2984011.

[16] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Artificial neural networks-based machine learning for wireless networks: A tutorial," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3039–3071, 4th Quart., 2019.

[17] X. Cheng, F. Lyu, W. Quan, C. Zhou, H. He, W. Shi, and X. Shen, "Space/aerial-assisted computing offloading for IoT applications: A learning-based approach," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1117–1129, May 2019.

[18] G. Li, H. Zhou, B. Feng, Y. Zhang, and S. Yu, "Efficient provision of service function chains in overlay networks using reinforcement learning," *IEEE Trans. Cloud Comput.*, early access, Dec. 23, 2019, doi: 10.1109/TCC.2019.2961093.

[19] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6379–6390.

[20] B. Feng, G. Li, G. Li, Y. Zhang, H. Zhou, and S. Yu, "Enabling efficient service function chains at terrestrial-satellite hybrid cloud networks," *IEEE Netw.*, vol. 33, no. 6, pp. 94–99, Nov. 2019.

[21] A. Gruslys, W. Dabney, M. Gheshlaghi Azar, B. Piot, M. Bellemare, and R. Munos, "The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning," 2017, *arXiv:1704.04651*. [Online]. Available: http://arxiv.org/abs/1704.04651

[22] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep multi-agent reinforcement learning based cooperative edge caching in wireless networks," in *Proc. ICC-IEEE Int. Conf. Commun. (ICC)*, Shanghai, China, May 2019, pp. 1–6.

[23] H. Zhu, Y. Cao, W. Wang, T. Jiang, and S. Jin, "Deep reinforcement learning for mobile edge caching: Review, new features, and open issues," *IEEE Netw.*, vol. 32, no. 6, pp. 50–57, Nov. 2018.

[24] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*. [Online]. Available: http://arxiv.org/abs/1509.02971

[25] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, "Deep reinforcement learning in large discrete action spaces," 2015, *arXiv:1512.07679*. [Online]. Available: http://arxiv.org/abs/1512.07679

[26] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in *Proc. 52nd Annu. Conf. Inf. Sci. Syst. (CISS)*, Princeton, NJ, USA, Mar. 2018, pp. 1–6.

[27] H. Zhu, Y. Cao, X. Wei, W. Wang, T. Jiang, and S. Jin, "Caching transient data for Internet of Things: A deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2074–2083, Apr. 2019.

[28] Y. Wei, F. R. Yu, M. Song, and Z. Han, "Joint optimization of caching, computing, and radio resources for fog-enabled IoT using natural actor–critic deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2061–2073, Apr. 2019.

[29] Y. He, F. R. Yu, N. Zhao, V. C. M. Leung, and H. Yin, "Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach," *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 31–37, Dec. 2017.

[30] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 44–55, Jan. 2018.

[31] N. Cheng, N. Zhang, N. Lu, X. Shen, J. W. Mark, and F. Liu, "Opportunistic spectrum access for CR-VANETs: A game-theoretic approach," *IEEE Trans. Veh. Technol.*, vol. 63, no. 1, pp. 237–251, Jan. 2014.

[32] V. Kirilin, A. Sundarrajan, S. Gorinsky, and P. K. Sitaraman, "Rl-cache: Learning-based cache admission for content delivery," in *Proc. Workshop Netw. Meets (AI ML)*, 2019, pp. 57–63.

[33] Z. Yang, Y. Liu, Y. Chen, and G. Tyson, "Deep reinforcement learning in cache-aided MEC networks," in *Proc. ICC-IEEE Int. Conf. Commun. (ICC)*, Shanghai, China, May 2019, pp. 1–6.

[34] J. Chen, W. Xu, N. Cheng, H. Wu, S. Zhang, and X. Shen, "Reinforcement learning policy for adaptive edge caching in heterogeneous vehicular network," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Abu Dhabi, United Arab Emirates, Dec. 2018, pp. 1–6.

[35] L. T. Tan and R. Q. Hu, "Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 10190–10203, Nov. 2018.

[36] H. Xing and W. Song, "Collaborative content distribution in 5G mobile networks with edge caching," in *Proc. ICC-IEEE Int. Conf. Commun. (ICC)*, Shanghai, China, May 2019, pp. 1–6.

[37] M. Furqan, C. Zhang, W. Yan, A. Shahid, M. Wasim, and Y. Huang, "A collaborative hotspot caching design for 5G cellular network," *IEEE Access*, vol. 6, pp. 38161–38170, 2018.

[38] J. A. Khan, C. Westphal, and Y. Ghamri-Doudane, "Information-centric fog network for incentivized collaborative caching in the Internet of everything," *IEEE Commun. Mag.*, vol. 57, no. 7, pp. 27–33, Jul. 2019.

[39] A. Ndikumana, S. Ullah, T. LeAnh, N. H. Tran, and C. S. Hong, "Collaborative cache allocation and computation offloading in mobile edge computing," in *Proc. 19th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Seoul, South Korea, Sep. 2017, pp. 366–369.

[40] D. Wu, L. Zhou, Y. Cai, and Y. Qian, "Collaborative caching and matching for D2D content sharing," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 43–49, Jun. 2018.

[41] H. Zhou, B. Liu, T. H. Luan, F. Hou, L. Gui, Y. Li, Q. Yu, and X. Shen, "ChainCluster: Engineering a cooperative content distribution framework for highway vehicular communications," *IEEE Trans. Intell. Transp. Syst.*, vol. 15, no. 6, pp. 2644–2657, Dec. 2014.

[42] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, D. Niyato, and D. I. Kim, "Distributed deep learning at the edge: A novel proactive and cooperative caching framework for mobile edge networks," *IEEE Wireless Commun. Lett.*, vol. 8, no. 4, pp. 1220–1223, Aug. 2019.

[43] W.-C. Chien, H.-Y. Weng, and C.-F. Lai, "Q-learning based collaborative cache allocation in mobile edge computing," *Future Gener. Comput. Syst.*, vol. 102, pp. 603–610, Jan. 2020.

[44] W. Jiang, G. Feng, S. Qin, and Y.-C. Liang, "Learning-based cooperative content caching policy for mobile edge computing," in *Proc. ICC-IEEE Int. Conf. Commun. (ICC)*, Shanghai, China, May 2019, pp. 1–6.

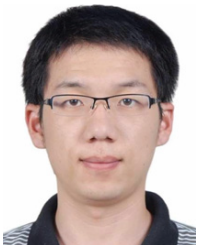
[45] W. Jiang, G. Feng, S. Qin, and Y. Liu, "Multi-agent reinforcement learning based cooperative content caching for mobile edge networks," *IEEE Access*, vol. 7, pp. 61856–61867, 2019.

[46] A. A. Korbut and I. K. Sigal, "Exact and greedy solutions of the knapsack problem: The ratio of values of objective functions," *J. Comput. Syst. Sci. Int.*, vol. 49, no. 5, pp. 757–764, Oct. 2010.

[47] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: http://arxiv.org/abs/1312.5602

YUMING ZHANG received the B.S. degree in communication and information systems from Beijing Jiaotong University, Beijing, China, in 2014, where he is currently pursuing the Ph.D. degree in telecommunications and information system with the National Engineering Laboratory for Next Generation Internet Interconnection Devices. His current research interests include software-defined networking, network function virtualization, mobile edge computing, queue theory, and machine learning.

BOHAO FENG (Member, IEEE) is currently an Associate Professor with the School of Electronic and Information Engineering, Beijing Jiaotong University. His research interests include software-defined networking, network functions virtualization, information-centric networking, service function chaining, 5G, the mobile Internet, and satellite networks. He has been served as a TPC Member of a number of international conferences, such as IEEE ICC and IEEE GLOBECOM, and a Reviewer for many international journals, such as the *IEEE Communications Magazine*, the IEEE COMMUNICATIONS SURVEYS & TUTORIALS, the IEEE NETWORK, the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, the IEEE INTERNET OF THINGS, and the IEEE CLOUD COMPUTING.



WEI QUAN (Member, IEEE) received the Ph.D. degree in communication and information system from the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2014. He is currently an Associate Professor with the School of Electronic and Information Engineering, Beijing Jiaotong University (BJTU). His research interests include key technologies for network analytics, the future Internet, 5G networks, and vehicular networks. He serves as an Associate Editor for the *Journal of Internet Technology* (JIT), *Peer-to-Peer Networking and Applications* (PPNA), and *IET Networks* and as a Technical Reviewer for many important international journals. He is also a member of ACM and a Senior Member of the Chinese Association of Artificial Intelligence (CAAI).

ALETENG TIAN received the B.S. degree in communication and information systems from Beijing Jiaotong University, Beijing, China, in 2018, where he is currently pursuing the master's degree in communication and information systems with the School of Electronic and Information Engineering. His current research interests include areas of software-defined networking, network function virtualization, mobile edge caching, and satellite networks.

KESHAV SOOD (Member, IEEE) is currently a Lecturer with Deakin University, Melbourne, VIC, Australia. His research interests include software-defined networks, the Internet of Things, and cybersecurity. He has served as a TPC Member of various IEEE conferences, including the International Conference on Computer Communications (INFOCOM), Bigdata service, and the International Telecommunications Network and Applications Conference (ITNAC). He is also a Professional Engineer with Engineers Australia Accreditation. He is also a Reviewer of the IEEE journals and TRANSACTIONS, including the *IEEE Network Magazine*, the *IEEE Wireless Communications Magazine*, the IEEE TRANSACTIONS ON COMPUTATIONAL SOCIAL SYSTEMS, the IEEE COMMUNICATION LETTERS, and IEEE ACCESS.

YOUFANG LIN received the Ph.D. degree in computer science and technology from Beijing Jiaotong University, Beijing, China, in 2003. He is currently a Professor with the School of Computer and Information Technology, Beijing Jiaotong University. His main fields of expertise and current research interests include big data technology, intelligent systems, complex networks, and traffic data mining.

HONGKE ZHANG (Senior Member, IEEE) is currently a Full Professor with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, China. He also directs the National Engineering Laboratory for Next Generation Internet Technology, China. His research has resulted in many academic articles, books, patents, equipment, and systems in the areas of communications and computer networks. His current research interests include network architecture, the Internet of Things, the Internet of Vehicles, and wireless communications. He is the Chief Scientist of the National Basic Research Program of China (973 Program) and has also served on the editorial boards of several international journals.