Towards Intelligent Adaptive Edge Caching Using Deep Reinforcement Learning

Ting Wang, *Senior Member, IEEE*, Yuxiang Deng, Jiawei Mao, Mingsong Chen, *Senior Member, IEEE*, Gang Liu, *Member, IEEE*, Jieming Di, and Keqin Li, *Fellow, IEEE*

***Abstract*—The tremendous expansion of edge data traffic poses great challenges to network bandwidth and service responsiveness for mobile computing. Edge caching has emerged as a promising method to alleviate these issues by storing a portion of data at the network edge. However, existing caching approaches suffer from either poor caching efficiency with low content-hit ratio or unintelligence of caching policies lacking self-adjustability. In this article, we propose ICE, a novel Intelligent Edge Caching scheme using a deep reinforcement learning (DRL) method to capture specific valuable information from the requested data. With the benefit of our proposed popularity model based on Newton's law of cooling, ICE fully takes into account the popularity of the contents to be cached and leverages the formulated Markov decision model to decide whether or not the contents should be cached. Moreover, to further improve the caching efficiency, we propose a novel distributed multi-node caching framework, named DCCC, assisted by a multi-tiered caching hierarchy. Comprehensive experiments show that the single-node ICE scheme greatly improves the cache hit rate and contents exchanging time in comparison with both DRL-based and legacy approaches, and our distributed multi-node caching scheme DCCC further significantly improves the overall utilization of caching space.**

***Index Terms*—Edge caching, deep reinforcement learning, quality of experience.**

I. Introduction

ALONG with the prosperity of 5G, the Internet of Things (IoT), and mobile cloud computing, the edge network is experiencing a dramatic increase in mobile traffic, as well as booming demand for high Quality of Experience (QoE) from end users on account of the high quality of service (QoS) requirements of various emerging services (e.g., autonomous driving, ultra-high-definition video, IoT-oriented services) [2], [3], [4]. This evidently impels a growing demand for network resources and storage resources to facilitate these services. It is predicted that the total amount of global data is expected to increase from 33 Zettabytes in 2018 to 175 Zettabytes by 2025 [5]. Such a huge amount of continuously generated data will inevitably exacerbate network congestion, increase operating and maintenance expenditure, and deteriorate the network QoS, which in turn impedes data delivery. Although cloud computing can mitigate the resource shortage problem of service requests initiated by the low-end terminal devices [6], [7], end users still suffer from high transmission latency with poor QoE due to their extremely long distance from the cloud data center. In order to deal with these problems, it will be greatly beneficial to push the data service from the remote cloud to the network edge to provide responsive services.

In practice, shifting some core network computation and storage resources to the network edge that is closer to end users would provide significant benefits [8], such as lowering the time it takes to feedback contents to users [9], improving energy efficiency, and enabling the network structure more flexible [10]. Various types of contents, such as videos, photos, music, and texts, can be stored in edge nodes, where such contents are usually frequently visited. The edge nodes storing the cached contents can be nano edge data centers, base stations (BSs), or even terminal devices around the users [11]. However, in view of the high deployment cost of distributed edge nodes, these edge nodes are usually low-end equipment with constrained computation and storage resources. Consequently, the restricted resource capacity of edge nodes brings significant challenges for offloading services and contents from the central cloud to the distributed edge nodes. In case the users' requested contents do not exist in edge nodes, the requests will be redirected to the remote cloud to retrieve contents. In this situation, these edge nodes will lose their intrinsic benefits, failing to provide responsive services. As a result, how to increase the hit rates of the contents cached on edge nodes is deemed a crucial and challenging problem. To this end, extensive studies have been carried out aiming to explore efficient edge caching schemes.

Conventional caching approaches are usually designed on the basis of the content visit frequency (e.g., Least Frequently Used (LFU)) [12], the content visit order (e.g. Fist In First Out

Manuscript received 8 December 2022; revised 24 January 2024; accepted 27 January 2024. Date of publication 1 February 2024; date of current version 3 September 2024. This work was supported in part by the National Key Research and Development Program of China under Grants 2022ZD0119102 and 2021ZD0114600, and in part by the Shenzhen Science and Technology Plan Project under Grant CJGJZD20210408092400001. An earlier version of this work was presented in 2021 IEEE Global Communications Conference [DOI: 10.1109/GLOBECOM46510.2021.9685196]. Recommended for acceptance by J. Xu. *(Corresponding author: Mingsong Chen.)*

Ting Wang, Yuxiang Deng, Jiawei Mao, and Mingsong Chen are with the MoE Engineering Research Center of Software/Hardware Co-design Technology and Application, Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai 200062, China (e-mail: twang@sei.ecnu.edu.cn; 51255902150@stu.ecnu.edu.cn; 71194501143@stu.ecnu.edu.cn; mschen@sei.ecnu.edu.cn).

Gang Liu is with Bell Labs, Nokia Shanghai Bell Corp., Shanghai 201206, China (e-mail: gang.i.liu@nokia-sbell.com).

Jieming Di is with Meta, Seatle, WA 98109 USA (e-mail: jieming.di@outlook.com).

Keqin Li is with the Department of Computer Science, State University of New York, New York, NY 10018 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/TMC.2024.3361083

(FIFO)) [13], or the content visit times (e.g., Least Recently Used (LRU)) [14]. However, these traditional caching schemes fail to take into account the content attributes, such as the sizes and types of content. As a result, even with high cache hit rates, these legacy caching approaches may still cause significant network latency since they may misguide the edge servers to cache smaller-size contents while neglecting the large-size contents that are more time-consuming. Hence, intuitively it would be beneficial to effectively recognize and classify the content requests in order to improve the overall caching efficiency.

To solve the problems confronted with traditional caching approaches, machine learning (ML), especially deep learning, has emerged as an effective technique for edge caching [15], [16], [17], which exhibits more advantages in dealing with the high dynamics of the edge environment and greatly outperforms traditional methods in cache optimization. Specifically, the diversity of content requests, the high complexity of the edge environment, the restricted resources, and the networking constraints, in combination, make ML-based approaches more effective than legacy methods in handling the tractable edge caching issue [18]. Empirical studies indicate that deep learning has a distinct advantage in classification. However, the caching quality is determined not just by reasonable classification of content requests but also by the caching decision-making policy that plays a more critical role. Therefore, we should not only leverage deep learning to classify content requests but also make full use of its excellent decision-making ability to effectively increase users' QoE and decrease the system's energy cost.

In view of the above observations, in our previous conference paper [1], we presented a novel deep reinforcement learning (DRL)-based single-node Intelligent Caching framework at the Edge, called ICE. In ICE, the edge caching problem is formulated as a Markov Decision Process (MDP), taking into account the popularity of requested contents as well as their energy cost on edge servers. Experimental results prove that benefiting from our novel content popularity model, the efficiency of caching strategies is significantly improved in terms of hit rate. Furthermore, with the learned knowledge of content attributes, ICE's caching policy prioritizes the most time-consuming and energy-consuming content. However, although the caching space of one single edge node is greatly optimized and fully utilized through our ICE scheme, it still dwarfs with the increasing number of resources.

To this end, in this article, we further propose a new distributed computing-based multi-node cooperative caching scheme named DCCC. In the DCCC scheme, the ICE strategy is leveraged to optimize the cache hit rate for each single node, and the Linux Virtual Server (LVS) is adopted to achieve load balancing among all involved edge nodes, where edge nodes are classified as proxy nodes and two-tiered caching nodes according to respective responsibilities. Proxy nodes, as the front-end load balancer, are responsible for distributing content requests to the caching nodes with lower network load to achieve good load balancing. Two-tiered caching nodes, constituting the server pool, manage the contents with unique identification information, whereas the tier-1 nodes are responsible for locating the tier-2 nodes that manage the requested contents and merging the results that will be fed back to users. With the benefit of ICE and DCCC, the caching space composed of different caching nodes obtains high utilization with a high cache hit rate. As a consequence, not only is the quality of experience of users substantially increased, but the system's energy cost is significantly lowered. The main contributions of this article are summarized as follows.

- We present a new content popularity model suiting the edge caching scenario based on Newton's cooling law. With this model, the system can calculate a reasonable popularity for each requested content, which further supports the formulation of caching policies.
- We propose ICE, a novel DRL-based single-node non-cooperative edge caching framework that outperforms its competitors in terms of QoE and energy efficiency.
- We further propose a novel distributed computing-based multi-node cooperative caching scheme, named DCCC, which aims to maximize the overall caching efficiency of multiple edge nodes, including overall cache space utilization, cache hit ratio, response time, and energy efficiency.
- We conduct comprehensive experiments to evaluate the performance of ICE and DCCC schemes. Experimental results prove that ICE is superior to both conventional caching approaches and DRL-based approaches, and DCCC further significantly increases the overall caching efficiency from various aspects.

The rest of this article is organized as follows. Related works are briefly summarized in Section II. The popularity model design and the problem formulation are presented in Section III. Section IV elaborates on the design of our proposed single-node caching scheme ICE. Section V details the design of the multi-node cooperative caching scheme DCCC. The experimental results are presented in Section VI. Finally, Section VII concludes the article.

II. Background and Related Work

With the dramatic increase in the number of terminal devices in recent years, the volume of data generated and transmitted on the network has increased exponentially. This dilemma impels the emergence and evolution of a new network service architecture, named edge caching, where the service provider chooses to store a copy of the data at the edge in order to ensure the user's QoE. However, storing data at the network edge comes with many new technical challenges and scientific issues, such as what/where/how to cache. In general, existing caching approaches can be divided into two groups: traditional caching approaches and ML-based caching approaches.

A. Traditional Caching Approaches

The content replacement policies commonly employed in traditional caching schemes include LFU, FIFO, and LRU. Specifically, in the LRU scheme, the least recently used contents are preferred to be removed and replaced with newly requested contents. Comparatively, when the size of the content to be cached exceeds the system's storage capacity, LFU will choose to delete the contents that are least frequently used. The FIFO

strategy simply replaces the contents according to the order of being cached, which follows the principle of first in first out. These conventional caching strategies are straightforward but truly efficient for same-sized contents. However, their performance degrades when dealing with contents whose sizes differ significantly and ignoring the introduced differentiated impact of the transmission delay. Besides, the caching policies of these approaches are static and are incapable of self-learning and self-adjustment, which thus cannot adapt to the dynamic edge environment.

B. ML-Based Caching Approaches

According to our investigation, deep learning (DL) and reinforcement learning (RL) are the two most commonly used learning paradigms in the edge caching scenario.

1) DL-Based Caching Approaches: The work [19] studies the caching problem of the information-centric network, then designs a DL-assisted content popularity prediction scheme leveraging software defined networking techniques, and finally presents a lightweight caching method based on storage capacity and popularity prediction. The work [20] considers the inter-relationships between sequential content requests and proposes a stimulable neural network-based caching approach by utilizing such time-varying inter-relationships in caching policies. The work [21] investigates the context-aware content caching problem confronted by the heterogeneous small cell networks and presents a convolutional neural network (CNN)-based caching scheme, which caches the predicted contents in advance to decrease the service latency. Similarly, the work [22] also proposes a DL-based context-aware caching scheme aiming to decrease the service latency and the backhaul congestion. Comparatively, the work [23] takes the sequential features of contents into consideration and proposes a proactive sequence-aware caching approach based on CNN with an attention mechanism.

2) RL-Based Caching Approaches: Reinforcement learning allows the agent to automatically choose the best action to take in order to maximize its performance. The work [24] puts forward a distributed Q-learning algorithm-based localized caching scheme, where a new strategy for increasing the cache hit rate is designed by incorporating Bayesian learning into predicting request preferences. Taking privacy preservation into account, the work [25] presents a distributed RL-based dynamic caching method aiming to improve the caching efficiency, where the authors formulate the distributed caching optimization as a model-free Markov decision process. Comparatively, the work [26] studies the energy conservation problem in mobile edge networks by leveraging caching methods. The authors then propose an RL-based approach to optimize the caching efficiency while aiming to minimize the system's energy cost. The work [27] designs an RL-based collaborative caching scheme for the Internet of Vehicles, which aims to minimize the content access delay by pre-caching the requested contents utilizing the long short-term memory method and optimizing the caching efficiency using an RL approach. The work [28] proposes a deep actor-critic (AC) RL-based caching strategy for edge caching in wireless networks by taking both cache hit rate and network latency into consideration. Aiming at the caching problem in-vehicle networks, the authors of the work [29] propose a multi-time scale framework based on deep reinforcement learning and design a mobile perception reward estimation method under the multi-time scale model. The agent in the deep Q-network (DQN) is responsible for receiving the system states and determining the best action. Each action includes roadside unit information and vehicle collection information, as well as the cache and contents of the requesting vehicle. Although DQN can well solve the problem that limitless states cannot be stored in limited tables, there arises another problem that how to make agents better understand the content popularity. The work [30] proposes a new approach to caching in CDNs. They propose a CDN caching design called Learning Relaxed Belady (LRB) to mimic the Relaxed Belady algorithm. The experimental results show that LRB outperforms traditional CDN algorithms in terms of overhead and performance.

C. Multi-Node Distributed Caching

The unprecedented high data traffic at the edge inevitably brings forward higher requirements for caching efficiency, while single-node caching schemes usually cannot meet such requirements in many cases. Therefore, numerous research has been conducted to explore effective multi-node cooperative caching schemes to further improve overall caching efficiency. The work [31] proposes a distributed edge caching scheme for dynamic contents of wireless service. The authors of this work convert the joint cache and recommendation strategy into a "single" cache strategy, avoiding the high training complexity of joint optimization. Authors in [32] present a collaborative caching method to minimize the file transmission time and content placement rate in user-centric mobile networks. The work [33] proposes a proactive cooperative caching mechanism to reduce transmission delay by predicting the content demand. Authors of the work [34] propose a multi-agent AC-based collaborative caching method to reduce data exchanges. The work [35] explores a Lyapunov optimization-based online collaborative edge caching algorithm, which targets minimizing the costs of content caching, content migration, and QoS penalties. To deal with the low task offloading efficiency caused by the consequences of having few services cached, the work [36] investigates efficient ways of dependent task offloading with restricted service caching and proposes a convex programming-based approach. To improve the efficiency of live video delivery, authors of the work [37] design a proactive content push technique that can mitigate the cache-miss issues in a distributed environment. Although these existing works mentioned above make full use of multiple edge nodes in cooperative caching, some critical problems are ignored, such as how to balance the loads, how to improve the computational efficiency of each caching node, and how to improve the overall caching space utilization of different caching nodes.

In this research, we develop ICE, a novel DRL-based caching method that addresses the drawbacks of existing traditional and ML-based approaches. Taking into account the content popularity and some specific properties of contents, we first present

a single-node caching scheme utilizing the DQN algorithm tailored for the formulated caching problem with better QoE and higher energy efficiency. Then, based on ICE, we further design an efficient multi-node distributed cooperative caching system called DCCC, which further improves the overall efficiency of edge caching.

III. Model Design and Problem Formulation

This section first presents the design of our novel popularity model based on Newton's law of cooling and then details the problem formulation of the caching problem at the network edge.

A. Content Popularity Model

To optimize the hit rates of the cached contents and maximize the probability of the requested contents being cached, the popularity of the content is often considered an essential, even decisive, metric in making caching decisions. In this article, we present a novel content popularity model based on Newton's law of cooling, which states that the heat loss rate of an object is proportional to the difference in temperature between the object and its surroundings. When an object and its surrounding environment have different temperatures, the heat loss per time unit is in proportion to the temperature difference. The selection of Newton's cooling law as the foundation for our cache popularity model is grounded in its aptitude for capturing the temporal dynamics of cache content popularity. Originally devised for object cooling, Newton's cooling law can be repurposed to effectively model the decay in popularity of cache contents. This modeling approach proves particularly pertinent in the domain of caching systems for a multitude of reasons: i) Newton's cooling law provides an intuitive framework for representing the innate waning of interest or popularity in cached content over time, enabling us to replicate the natural cooling down of users' interest. ii) The cooling constant in Newton's cooling law serves as a tunable parameter, facilitating alignment with real-world caching scenarios. Through adjustments to this constant, we gain control over the rate at which content popularity diminishes, enhancing adaptability across different caching systems. iii) The mathematical elegance of Newton's cooling law streamlines the modeling process, empowering us to make predictions about cache content popularity through a straightforward equation. This simplicity proves advantageous for both theoretical analysis and practical implementation. iv) Newton's cooling law has demonstrated successful applications across various domains beyond its original context, including information diffusion and social network dynamics. Its inherent adaptability and versatility render it a promising choice for modeling content popularity in cache systems.

When there is a temperature difference between the object and its surrounding environment, the heat lost per time unit is in proportion to the temperature difference, where the proportional coefficient is known as the heat transfer coefficient. We performed a sequence of formula derivations on the basis of Newton's Law of Cooling, as stated in (1), and ultimately obtained the popularity model calculated as (8). Table I lists the symbols with relevant descriptions, and the derivation process is given below:

TABLE I
Summary of Notations

Symbols	Description
t	The target time to compute temperature
t_0	Initial time
$T(t)$	Temperature of the content at time t
T_{se}	Temperature of surrounding environment
λ	Proportionality coefficient
Θ_{req}	Score of user's request to contents
Θ_{others}	Score of user's other actions to contents
μ	The size of population
γ	Initial content popularity
φ	Changes of the content popularity due to users' behaviours
$P_c(t)$	The content popularity at time t
B	Bandwidth of the network
E	Energy consumption of transmission
v_j	Content size j
d_j	Total length of transmission route of accessing to content j without edge caching
$d_j^{'}$	Total length of transmission route of accessing to content j with edge caching

$$\frac{dT(t)}{dt} = -\lambda(T(t) - T_{se}), \tag{1}$$

where $T(t)$ indicates the temperature of the content at time t, T_{se} refers to the temperature of the surrounding environment, and λ is the proportionality coefficient. Then, we can obtain that

$$\int \frac{dT(t)}{T(t) - T_{se}} = \int -\lambda dt, \tag{2}$$

$$\ln(T(t) - T_{se}) = -\lambda t + C. \tag{3}$$

We use elementary variations to obtain

$$T(t) = T_{se} + We^{-\lambda t}. \tag{4}$$

Then, the variable t is substituted by the initial time t_0, and we obtain

$$T(t_0) = T_{se} + We^{-\lambda t_0}. \tag{5}$$

Thus, W can be computed as

$$W = (T(t_0) - T_{se})e^{\lambda t_0}. \tag{6}$$

Combined with (6), (4) can be further derived as follows

$$T(t) = T_{se} + (T(t_0) - T_{se})e^{-\lambda(t_0 - t)}. \tag{7}$$

In essence, the changes in content popularity could be interpreted as a natural cooling process of the content. As a result, if no intervention is involved, the content popularity will ultimately fall to zero, which is the same temperature as the surroundings. Thus, we can deduce that T_{se} in (7) will turn out to be zero. Then, we have

$$T(t) = T(t_0)e^{\lambda(t - t_0)}. \tag{8}$$

Based on the above derivations, taking into account the impact of users' behaviors on the content popularity, the increase in

the number of requests for the content implies that the content popularity is resisting the natural cooling brought by time. That is to say, the content popularity will increase and the rate of content cooling will decrease. As thus, the impact of users' behaviors on the content popularity can be computed as (9).

$$\varphi = (\Theta_{req} + \Theta_{others}) \div \mu. \tag{9}$$

The above analysis demonstrates how content popularity changes over time, dependent on its initial popularity. It can be concluded that the popularity of the content is determined not only by external influences but also by its initial popularity, which depends on its content type. Empirically, different types of contents, such as sports-related content and political news, may have different initial popularities.

To summarize, in our caching scheme, we can combine the changing process of the content's popularity over time with its initial states to dynamically represent the content popularity at any given time, as computed in (10).

$$P_c(t) = (\gamma + \varphi) \div T(t). \tag{10}$$

B. Problem Formulation

In the scenario of edge computing, cache units such as BSs are expected to cache as many contents as possible from the core network [38] so as to decrease the transmission delay and energy cost within the backhaul. The total transmission time Rt_i and transmission energy Re_i of request i are calculated as the sum of all contents' transmission time and energy, respectively. The transmission time of each content rt_j can be calculated as (11) and the transmission energy of each content re_j can be computed as (12). The calculation of total transmission time Rt_i and energy Re_i can be defined as (13) and (14), respectively.

$$rt_j = v_j \times B \times d_j, \tag{11}$$

$$re_j = v_j \times E \times d_j, \tag{12}$$

$$Rt_i = \sum_{j=1}^{N} rt_j, \tag{13}$$

$$Re_i = \sum_{j=1}^{N} re_j. \tag{14}$$

In general, if one requested content has been cached on some edge nodes, then the reduction ratio of transmission time pt_i and energy consumption pe_i can be formulated as (15) and (16), respectively.

$$pt_i = 1 - \frac{\sum_{j=1}^{N} v_j \times B \times d'_j}{\sum_{j=1}^{N} v_j \times B \times d_j}, \tag{15}$$

$$pe_i = 1 - \frac{\sum_{j=1}^{N} v_j \times E \times d'_j}{\sum_{j=1}^{N} v_j \times E \times d_j}. \tag{16}$$

The goal is thus to maximize the average reduction ratio of transmission time and energy consumption of the system, and the optimization objective can be formulated as

$$Objective: \max_{d_j>0,\, d'_j>0} (pt_i + pe_i), \tag{17}$$

subject to the following constraints:

$$d_j > 0, \quad d'_j > 0, \quad B > 0, \quad E > 0 \tag{18}$$

$$\sum_{j=1}^{N} v_j \times B \times d_j \geq \sum_{j=1}^{N} v_j \times B \times d'_j, \tag{19}$$

$$\sum_{j=1}^{N} v_j \times E \times d_j \geq \sum_{j=1}^{N} v_j \times E \times d'_j. \tag{20}$$

Thus, the problem to be addressed is how to achieve better performance by maximizing the reduction rate of transmission time and energy consumption. Note that the system proposed in this article holds for resources whose content usually does not change, such as videos and music.

IV. DRL-Based Single-Node Non-Cooperative Caching Scheme

This section first introduces the RL model tailored for the edge caching problem. Then, we elaborate on the design of our proposed DRL-based ICE caching approach.

A. RL Model

In RL, through interacting with the environment, the agent learns the best behavior so as to obtain the maximum reward. The goal of RL can be characterized as a Markov Decision Process (MDP), which necessitates the use of the state space and reward function. Here, we define the MDP model by the tuple $\{S, A, R(s, a)\}$.

- S $= \{s_1, s_2, \ldots, s_n\}$ represents the set of possible states constructed from all requests.
- A $= \{a_1, a_2, \ldots, a_n\}$ denotes the set of actions. The agent performs an action a_t on the environment based on the current state s_t, and the environment subsequently transits to a new state s_{t+1} based on the transition probability.
- R $= \{r_1, r_2, \ldots, r_n\}$ stands for the set of rewards, where r_t indicates the reward that the agent receives after completing action a_t in state s_t.

The reward function is the most crucial part of an RL algorithm because it ultimately determines the actions to be taken, which in turn exerts a considerable impact on the algorithm's performance. Empirically, to maximize the ultimate reward, the agent should take not just the current but also the future reward into consideration. The discount rate γ, $0 < \gamma < 1$, is used to indicate the reduction proportion of future rewards. Here, we denote R_t as the discounted future reward, which is computed as follows.

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{j=0}^{\infty} \gamma^j r_{t+j+1}. \tag{21}$$

The objective of an RL agent is to find the best strategy that can generate the highest reward in each state in order to maximize

the cumulative reward over the long term. When the policy pi is performed by the agent in the current state, the state-value function that calculates the expected value of the discounted cumulative reward is defined as (22), where t denotes the time step.

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{j=0}^{\infty}\gamma^{j} r_{t+j+1} | s_t = s\right]. \quad (22)$$

Equation (23) gives the optimal state-value function in accordance with the Bellman Equation, where $R(s,\pi(s))$ denotes the average of the immediate rewards $r(s,\pi(s))$, and $P_{ss'}(\pi(s))$ refers to the probability of transiting from current state s to next state s' when policy $\pi(s)$ is executed.

$$V^{\pi^*}(s) = \max_{a\in A}\left[R(s,a) + \gamma\sum_{s'\in S} P_{ss'}(a) V^{\pi^*}(s')\right]. \quad (23)$$

Likewise, the action-value function, which determines the value of taking action x in state s under the policy π, is defined as $Q^{\pi}(s,a)$ and it is calculated as (24).

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{n} r_{t+n+1} | s_t = s, a_t = a\right]. \quad (24)$$

The optimal action-value function is computed as (25).

$$Q^{\pi^*}(s,a) = R(s,a) + \gamma\sum_{s'\in S} P_{ss'}(a)\max_{a'}\left[Q^{\pi^*}(s',a')\right] \quad (25)$$

Algorithm 1: DQN-Based Caching Scheme.

Input: Requests $s_1, s_2 ... s_t$
Output: Action $a_1, , a_2 ... a_t$
Initialize Q-network, replay memory, and action-value function Q with random weights;
for $episode = 1, ..., M$ **do**
  Initialize environ. and analyze content req. s_t
  while *true* **do**
    set $step = step + 1$;
    Choose a random action a_t with probability ε, otherwise choose $a_t = max(Q(s_t, a))$;
    Perform action a_t;
    if $t > 1$ **then**
      observe reward r_t;
      set content request = s_{t+1};
    end
    if *break* **then**
      Break;
    end
    Store $\{r_t, a_t, s_t, s_{t+1}\}$ to replay memory;
    if $step > 2000$ *and* $step \,\%\, 10 == 0$ **then**
      Sample random minibatch of transitions $\{r_t, a_t, s_t, s_{t+1}\}$ from replay memory;
      Learn Q-network;
    end
    set $s_t = s_{t+1}$
  end
end

B. ICE: Deep Q-Learning-Based Caching Scheme

In an unknown environment, deep Q-learning provides an effective technique enabling agents to effectively acquire the best rewards in case the rewards R and the transition probabilities P are unknowable. To estimate the action-value function in DQN, a function approximator is typically utilized. A nonlinear approximator, for instance a neural network $Q(s,a,\theta) \approx Q(s,a)$, is often leveraged to approximate a function. Here, θ represents the weights of the neural network and in each iteration it is adjusted to train the network to decrease the mean square error. In accordance with the work [39], derived from Q-learning we define the evaluation of $Q(s,a)$ as in (26), where $\zeta \in (0,1)$ denotes the learning rate.

$$Q(s_t,a_t) = Q(s_t,a_t) + \zeta\left[r_t + \gamma * maxQ(s_{t+1},a_{t+1}) - Q(s_t,a_t)\right] \quad (26)$$

The training of the Q-network will be executed until it converges by minimizing the loss function, where the loss function is given as (27).

$$L(\theta^{+}) = \mathbb{E}[(r_t + \zeta * maxQ(s_{t+1},a_{t+1},\theta) - Q(s,a,\theta^{+}))^2] \quad (27)$$

The evaluation network and the target network are two distinct neural networks in the DQN algorithm, which have the same structure but different parameters. θ^{+} and θ represent the evaluation network and target network parameters, respectively. Specifically, the current action-value Q values $Q(s,a,\theta^{+})$ are computed by the evaluation network using the latest parameter θ^{+}, and θ^{+} will be updated in each cycle. Comparatively, the the next action-value Q value $Q(s',a',\theta)$ is computed by the target network using the parameter θ, which is updated on a regular basis. Note that the correlation between the Q value and the Q target value can be decreased via the target network, enabling DQN to converge more easily.

Furthermore, to decrease the data correlation and improve learning efficiency, the experience replay mechanism is introduced in DQN to store the valuable experiences obtained through interacting with the environment. As experiences, the current state, action, reward, and next state are stored in the memory in the format $\{s, x, r, s'\}$. Then, during the training stage, the training data can be selected at random from the experience replay space. In this way, the neural network's overfitting problem can be well solved by such kind of random selection, which breaks the correlation of experiences. In the next three subsections, we will present the design of states, actions, and reward functions, which are pivotal for a DQN policy to obtain optimal rewards.

1) States: In our approach, the current state s is jointly determined by the popularity and size of the contents. Here, we define the state at time t as a set $\{P(t), v_t\}$, where $P(t)$ represents the content popularity and v_t denotes the content size.

2) Actions: We design three different kinds of actions for the agent in our ICE, namely, *adding cache*, *deleting cache*, and *keeping cache unchanged*. Using the action-value function, the agent outputs the one-hot value for each action and chooses the most significant value to perform the corresponding action.

3) Reward Function: Based on extensive trial experiments and mathematical derivation, we define the reward function as (28).

$$R = \begin{cases} \dfrac{1}{(1+e^{-P_t - v_t})} & hitcache = true \\ -\dfrac{1}{(1+e^{-P_t - v_t})} & hitcache = false \end{cases} \tag{28}$$

The reward function, denoted as R in (28), serves as a crucial component in our RL framework. It is designed to quantify the desirability of a given state-action pair by considering both the content popularity (P_t) and the content size (v_t) at a specific time t.

- Positive Reward for Cache Hits: In the case where a cache hit occurs (i.e., $hitcache = true$), the reward is a positive value. We use the sigmoid function $1/(1+e^{-(P_t - v_t)})$ to ensure that the reward increases as both content popularity and size contribute positively to the desirability of the state-action pair.
- Negative Reward for Cache Misses: Conversely, when a cache miss occurs (i.e., $hitcache = false$), the reward is a negative value. This penalizes cache misses, discouraging the caching system from evicting potentially popular content.

By incorporating both popularity and content size into the reward function, we aim to strike a balance between favoring popular content and considering the available cache space, ultimately improving the overall performance of the caching system.

The working procedure of our ICE approach is described in Algorithm 1. At the very beginning, the stochastic policy and neural network are initialized. Subsequently, the agent generates an action depending on current states and policy. The agent then observes the next state in the environment and obtains rewards based on the prior strategy's outcomes. Afterwards, the tuple $\{r_t, a_t, s_t, s_{t+1}\}$ is stored in the experience replay memory using the observation. A small batch of transitions is then randomly sampled from the replay buffer. Next, the Q-network is trained based on each batch through the minimization of the loss function. Lastly, the Q-network adjusts the adaptive learning rate to maintain the training's stochastic stability.

C. System Workflow of ICE

Fig. 1 depicts the workflow of the ICE system. In the initial phase, the Q-network and replay memory of DQN are initialized. The system then computes the content popularity using our proposed content popularity model. Then, the ICE system updates the reward returned by the previous action to reflect the current content. Afterward, the system selects the most significant value from each action's one-hot value to perform the corresponding action if a random number generated with a random seed based on the current time is greater than the parameters; otherwise, the system randomly chooses an action from the action space to execute. Before caching the current content, the systems will first judge the availability of storage space. If the size of the content to be cached is greater than the available storage capacity of the server, then the system will delete the content whose reward is the least and replace it with the current requested content. If the system has sufficient storage space or it does not need to cache the current content, then the system will perform corresponding actions. Then, ICE updates the Q-network with the aim of minimizing the loss function. In this way, the system will gradually optimize the caching decisions over time by repeating the above procedures and eventually achieve optimal policies.

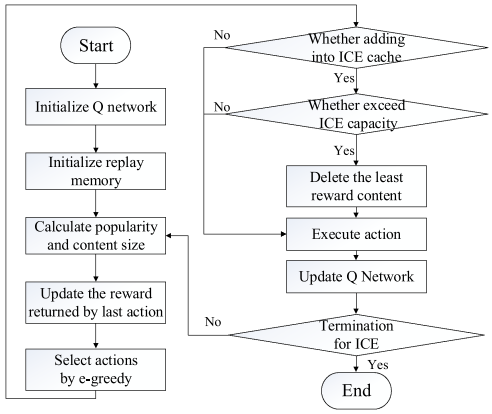

Fig. 1. ICE system workflow.

V. Distributed Computing-Based Cooperative Caching Scheme

The proposed ICE algorithm mainly focuses on optimizing the caching efficiency of a single node. To further increase the overall caching efficiency, in this section, we introduce a new distributed computing-based multi-node cooperative caching scheme, named DCCC. The relationship between ICE and DCCC can be briefly described as DCCC offloading requests to each edge node of the domain and then each edge node taking advantage of ICE to determine whether to cache the requested contents.

The primary purpose of DCCC design is to improve the utilization of storage space as much as possible, which is quite different from the mainstream distributed systems. The mainstream distributed systems aim to improve the availability of services by adopting the form of master-slave backup in data storage, which will also bring the problem of data redundancy. For the reason that some key data, such as orders and amounts, do not allow data inconsistency among multiple nodes in the distributed system, distributed systems tend to improve the reliability and stability of system services by reducing space utilization. Therefore, the external consistency of the whole system can only be realized by backing up data. However, in the cache field, this problem is not very critical. Even if the data of each node in DCCC is inconsistent or missing, it will only increase the delay of data acquisition from the core network, and there will be no problem with the data ultimately obtained by users. Therefore, the design purpose of DCCC is no longer to take consistency as the first requirement but to take high performance as the primary purpose, that is, to increase the probability of cache hit. DCCC

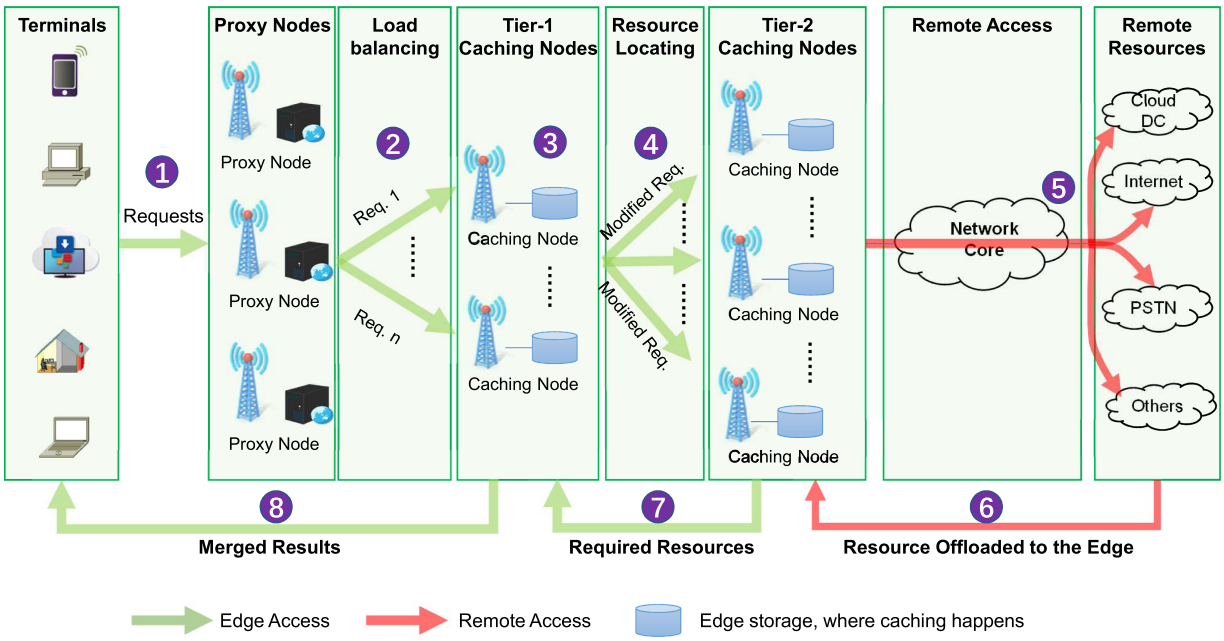

Fig. 2. DCCC system architecture.

achieves this requirement by maximizing the utilization of cache space.

The distributed computing system architecture is as shown in Fig. 2, and the workflow of DCCC is described in Algorithm 2. Overall, according to respective responsibilities, the edge nodes in the DCCC system are divided into two categories: proxy nodes and two-tiered caching nodes. This is only a logical classification, and physically one caching node can have two roles at the same time by running different processes. The proxy nodes, as the front-end load balancer, are responsible for load-balancing content requests to appropriate caching nodes. The role of tier-1 nodes is to locate tier-2 nodes that have the requested contents. The tier-2 caching nodes return the requested contents to the corresponding tier-1 node if they have been cached before, or remotely obtain the required contents from the core network and execute ICE to decide whether the contents should be cached, then return the results to the tier-1 node. Finally, the corresponding tier-1 node merges the results and replies to users with requested contents. As depicted in Fig. 2 and Algorithm 2, the whole working procedure of DCCC is described as follows.

Step 0: The DCCC algorithm initializes and starts the Proxy Nodes, Tier-1 Caching Nodes, Tier-2 Caching Nodes, and the standalone ICE algorithm in the service (Algorithm 2: lines 1–2).

Step 1: The content requests from terminals are first sent to proxy nodes.

Step 2: The proxy node leverages Linux Virtual Server (LVS) to load balance the received requests to appropriate tier-1 caching nodes (Algorithm 2: lines 4–5).

Step 3: When one tier-1 caching node (e.g., node x) receives content requests distributed by the proxy node, it will locate the responsible tier-2 caching nodes that have the requested contents through the hashing operation. The hash value of content n_i is calculated as below (Algorithm 2: line 7).

$$H = n_i \ mod \ p, \tag{29}$$

where p is the amount of caching nodes. The current caching node (node x) will find the IP address a_i of the target tier-2 caching node by searching the routing table and send an RPC (Remote Procedure Call) message to obtain the content n_i. The RPC message is as described in Fig. 3. The records in the routing table are defined as the tuple $U = \{H, Y, T\}$.

- $H = \{h_1, h_2, \ldots, h_n\}$ is the set of hash values. Each content requested by users will be assigned a unique hash value h_i.
- $Y = \{y_1, y_2, \ldots, y_n\}$ is the set of IP addresses of tier-2 caching nodes, and y_i refers to the IP address

RequestResource RPC	
Invoked by tier-1 caching nodes to gather resource	
Arguments:	
resourceId	unique identification of each resource
time	current time slot
Results:	
content	content of resource
success	true if tier-2 caching nodes successfully return contents
Receiver implementation:	
1. Return false if tier-2 caching nodes fail to gather contents 2. Return false if tier-2 caching nodes returning contents times out 3. Return false if content is null	

Fig. 3. RPC message.

of the node that stores the content with hash value h_i.

- $T = \{t_1, t_2, \ldots, t_n\}$ is the set of time and t_i means the time when the node with IP address y_i caches the content whose hash value is h_i.

Step 4: The current tier-1 caching node (node x) sends an RPC message with arguments consisting of the popularity, the size of the content n_i, and other identification information of the content n_i to the corresponding tier-2 caching node whose IP address equals y_i or the first node with IP address greater than y_i. Accordingly, when one tier-2 caching node receives an RPC message, it will first search its caching table to check whether the requested content n_i is cached. The records in the caching table are defined as the tuple $U = \{N, T, C\}$ (Algorithm 2: lines 8–9).

- $N = \{n_1, n_2, \ldots, n_n\}$ is the set of contents' identification information. In the system, each content requested by users has a unique identification information n_i.
- $C = \{c_1, c_2, \ldots, c_n\}$ indicates the set of contents.
- $T = \{t_1, t_2, \ldots, t_n\}$ has the same meaning as described in Step 3.

Step 5: If the content n_i does not exist in the caching table, then the tier-2 node will request the contents from the remote servers (Algorithm 2: lines 10–11).

Step 6: The requested contents are sent back to the tier-2 node from the remote servers. Then, the tier-2 node determines whether to cache this content by taking advantage of the ICE scheme (Algorithm 2: lines 12–14).

Step 7: Contrary to step 5, if the content n_i already exists in the caching table, then the tier-2 node will directly return the content c_i to the tier-1 caching node. Otherwise, the tier-2 node will transmit the remotely obtained contents in Step 6 to the tier-1 caching node (Algorithm 2: lines 12–14).

Step 8: When the tier-1 caching node receives the returned requested contents from tier-2 caching nodes, the tier-1 node merges the results and returns them to terminals (Algorithm 2: lines 16–18).

Notably, in DCCC, the number of caching nodes is adjustable according to the peak number of requests. To maintain the stability of the system, when the number of caching nodes changes, the hash consistency principle is used to adjust the content of caching tables stored in each caching node. Fig. 4 gives an example that illustrates the working procedure of locating the requested contents. Each caching node has a unique hash value h_i, and all caching nodes form a circular linked list in ascending order according to their respective hash values. Tier-1 caching nodes calculate a target hash value for each requested resource according to its unique ID number n_i. Different contents may get the same hash value, and the corresponding content may be cached in the same caching nodes. As Fig. 4 illustrates, the hash value of content n_0 is $h_{n_0} = 2$, which is the same as that of content n_1. The tier-2 caching node $node_2$ with hash value $h_{node_2} = 4$ is the first node whose hash value is greater than the hash value of n_0 and n_1. Thus, the node $node_2$ is responsible for storing the content of n_0 and n_1.

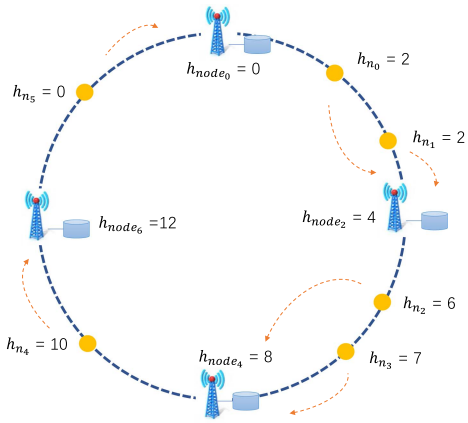

Fig. 4. Hash circle.

If a new caching node with a hash value h_j launches, the caching node first duplicates the caching table from the caching node with hash value h_{j+1} or h_0 (when j equals the total number of caching nodes p). If a caching node with a hash value h_j is offline, its caching table will be duplicated by the caching node with hash value h_{j+1} or h_0 (when j equals $p-1$). In this way, when the number of caching nodes changes (increases or decreases), only a few nodes' caching tables need to be changed instead of re-hashing all caching nodes, which enables the whole system with great scalability and stability.

Finally, we briefly discuss the deployment of DCCC. DCCC is inherently suited for contemporary deployment systems such as Docker and Kubernetes (k8s). We deploy our cache nodes, edge caching services, and the ICE module within Docker containers, managing them through Kubernetes and facilitating inter-container communication using RPC. In the subsequent experimental section, we follow the logic of containerized deployment to deploy DCCC on Alibaba Cloud, completing the associated experiments.

VI. Experiments

This section evaluates the effectiveness of our ICE and DCCC approaches. To validate the superiority of our approaches, we compare ICE and DCCC with both traditional caching schemes and DRL-based intelligent caching methods in terms of cache hit rate, transmission delay, energy cost, and contents exchanging rate.

A. Experimental Settings

Experiments were conducted on a Linux workstation with 2.80 GHz Intel (R) Quad-Core CPU and 8 GB memory. We implement ICE using Python v3.7.0 and use TensorFlow 1.9.0 [40] as the training platform. The language used to build DCCC is Go, and the version of Go SDK is 1.17.3. The edge nodes are implemented as containers in the Docker environment (Docker engine version 20.10.5). In experiments, the size of contents ranges from $1MB$ to $2500MB$, the cache size of each node is

Algorithm 2: DCCC Algorithm.

Initialize ICE in each caching node.
Initialize the routing table and caching table in each caching node.
while *True* **do**
  The proxy node transfers the request to a caching node k_i, which has a lower network load.
  The caching node k_i calculates the number of contents in the request, noted as N.
  for $i \in [1, N]$ **do**
    k_i gets contents' identity n_i from the request and calculates the hash value h_i of n_i. ;
    k_i searches routing table to find the target caching node k_j whose hash value equals to or firstly greater than h_i.;
    k_i send a modified request to k_j by RPC. k_j obtains RPC request and calculate the result of ICE of n_i;
    if n_i *is not cached in* k_j **then**
      k_j request the content of n_i from core network;
      if *ICE in* k_j *decides to cache* n_i **then**
        k_j caches the content of n_i ;
      end
    end
    k_j returns content of n_i to k_i;
  end
  k_i merges all contents and returns them to users;
end

TABLE II
PARAMETER SETTINGS IN OUR SIMULATIONS

Parameter	Value	Description
Training-batch size	7	The number of experience cases used for each training step.
Update Frequency	5	A training step is performed every 5 processing times.
Experience replay buffer size	30,000	The number of stored training cases.
Learning rate	0.0001	The update rate of neural network parameters.
Neural network update rate	0.001	The update rate of Q-network.

set to 32,000 MB, and the requests for contents are dynamically and randomly generated, and the random seed is time. Table II summarizes a list of key parameter settings.

B. Baseline Algorithms

To compare the performance of our methods with other methods, we implement two traditional caching schemes and one DRL-based caching scheme, as listed below.

- *Least Frequently Used (LFU) [12]:* A widely used traditional method, which prefers to remove the least frequently used contents.
- *Least Recently Used (LRU) [14]:* A widely used traditional method, which will remove the least recently used content first. DQN [29]: an RL-based caching method, which applies the traditional DQN model without considering the content popularity.
- *Actor-Critic (AC):* An RL-based caching method. The compared AC algorithm here is the traditional Actor-Critic algorithm, which was proposed in the work [41]. It combines the advantages of value-based and policy-based approaches. It can strike a balance between stability and efficiency in learning the optimal policy. In our comparison experiments, the model parameters of the used Actor-Critic algorithm here are consistent with DQN to ensure the fairness and effectiveness of the comparison experiments.

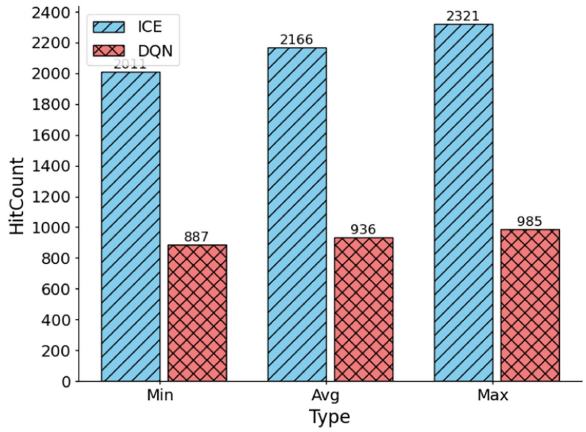

Fig. 5. Performance of ICE and DQN.

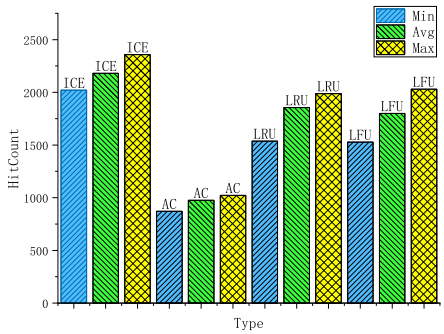

Fig. 6. Hit count.

C. Experimental Results

1) Cache Hit Count and Hit Rate: The cache hit rate and hit count of a caching algorithm are considered the most important metrics for determining its effectiveness. Figs. 6 and 7 demonstrate the number of hit counts and hit rates of our ICE and compared algorithms, respectively. From Fig. 6, we can observe that our ICE achieves the highest hit counts for all cases. In terms of hit rate, as shown in Fig. 7, our ICE achieves an average of 21.8% accumulative hit rate counting in all types of contents, while LRU, LFU, and AC only reach 18.5%, 18.1%, and 9.8% hit rates on average, respectively. Fig. 7 also shows that our ICE demonstrates greater advantages for larger-sized

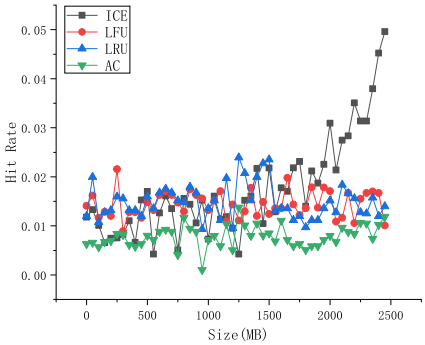

Fig. 7. Hit rate of each type of content.

TABLE III
COMPARISONS OF CACHE HIT EFFICIENCY

Methods	ICE	LRU	LFU	AC
Cache hit efficiency	19.60%	6.70%	7.60%	5.10%

contents. In addition, Fig. 5 compares the performance of our ICE and the DQN scheme [29] in terms of successful cache hit counts, where the content data size for input was set as a random number between 2000 MB and 2500 MB. It can be observed that our ICE method significantly outperforms DQN. These achievements evidently prove that our ICE outperforms its competitors in terms of both hit counts and hit rates. The better performance of ICE in cache hit rate mainly gives the credit to our effective Newton's cooling law-based content popularity model, which is prone to caching the contents that are more likely to be visited again in the future.

2) Transmission Time and Energy Conservation: Because the amount of energy cost in content data transmission is proportional to its size, thus caching strategies should prioritize larger contents. Table III compares the cache hit efficiency of different methods, which is calculated as the ratio of the top five large-sized contents' hit count to the total hit count. The results reveal that our ICE achieves the highest cache hit efficiency, which is 12.9%, 12.0%, and 14.5% higher than LRU, LFU, and AC, respectively. This fully convinces the advantages of our ICE in terms of identifying contents that are more beneficial in minimizing transmission latency and energy costs. Besides, in terms of the time-reducing rate, which is computed as the percentage of time saved compared to the time cost without using caching techniques, Fig. 8 reveals that ICE achieves 35.1% time-reducing rate on average, which is 11.6%, 12.0%, and 22.1% higher than LRU, LFU, and AC, respectively. Moreover, from Fig. 10 we can observe that, in terms of the energy-reducing rate, which is calculated as the percentage of energy saved compared to the energy cost without using caching techniques, the performance of our ICE is 11.8%, 12.2%, and 22.7% higher than that of LRU, LFU, and AC, respectively. These achievements imply that the popularity and other relevant valuable information of contents are fully exploited in ICE, making the caching decisions more advantageous.

3) Model Performance With Fixed Patterns of Content Requesting: In the loading process of programs in most real-world

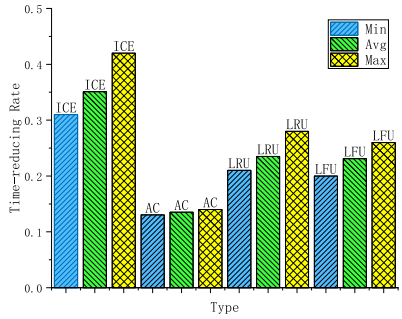

Fig. 8. Time reducing rate.

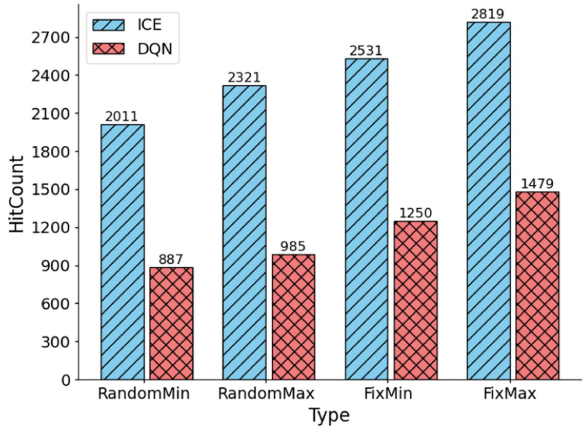

Fig. 9. Performance with fixed or randomized contents.

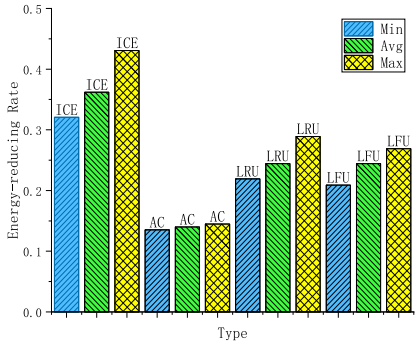

Fig. 10. Energy reducing rate.

scenarios, such as large-scale programs like games, page replacement is commonly performed. The amount of page replacements is often determined based on the demands at the current time point. Similarly, the size of the content replaced when triggering page replacement in the cache is also determined by service requirements. This is a challenging scenario to simulate due to its inherent randomness, involving numerous unpredictable factors. Therefore, initially, we used randomly generated content sizes as the raw input for our experiments. However, in certain situations, such as the need to rapidly retrieve a substantial amount of data from the disk in an instant, when exceeding the hardware's reading capacity, there may be a period

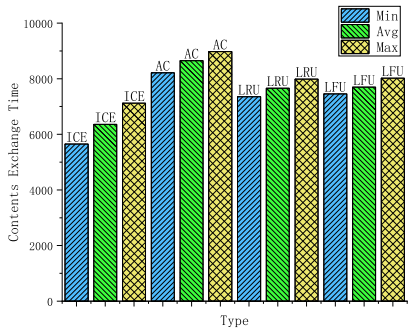

Fig. 11. Contents exchanging times.

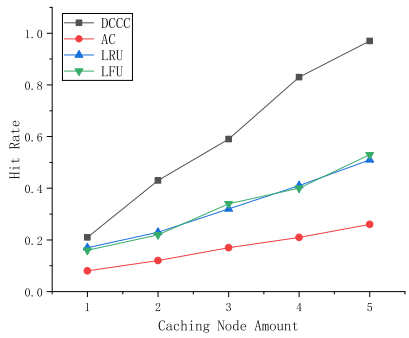

Fig. 12. DCCC hit rate.

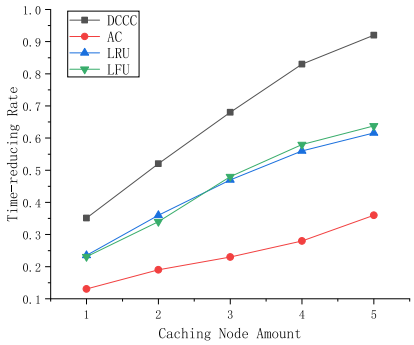

Fig. 13. DCCC time reducing rate.

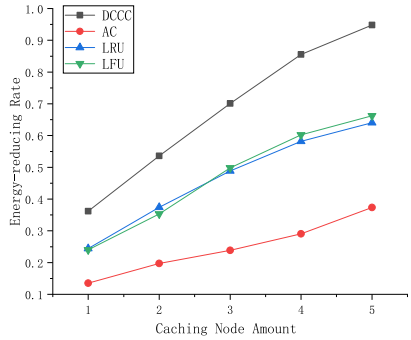

Fig. 14. DCCC energy reducing rate.

of time during which a fixed-size content is read, depending on the different operating systems. To account for this, we further evaluated the performance of ICE and DQN using a fixed-size content request pattern, where the requested content size is set to 2500 MB. The experimental results shown in Fig. 9 demonstrate that the performance of both ICE and DQN under fixed-sized content settings is superior to that of randomized settings. Moreover, with the incorporation of Newton's cooling law-based popularity model, ICE achieves higher hit counts than DQN.

4) Contents Exchanging Rate: Fig. 11 demonstrates the performance of all approaches in terms of content exchange times. The results reveal that ICE achieves the minimum number of content exchanges, which is 78.1%, 81.8%, and 82.5% of that in AC, LFU, and LRU, respectively. Note that our ICE performs a different content exchange policy compared with the other three approaches, where LRU, LFU, and AC exchange contents whenever the content does not hit the cache, while ICE takes actions to add or delete contents depending on the judgment of the agent about whether or not the content will be requested in a short time.

5) Cache Hit Rate of DCCC: Fig. 12 compares the overall hit rates of DCCC, LRU, LFU, and AC with a different number of caching nodes. The experimental results reveal that when the number of caching nodes increases, the hit rates of all methods increase accordingly. We can observe that DCCC achieves up to 97.1% hit rate, while the hit rates of LRU, LFU, and AC are only 53.2%, 51.0%, and 26.3%, respectively. Evidently, DCCC significantly outperforms the other three methods in terms of hit rate. The main reason behind this achievement is that DCCC can ensure that the contents cached by DCCC have no duplications in all caching nodes. Comparatively, the other three caching methods can not fully exploit or maximize the utilization of multiple caching nodes since many copies of the same content may be stored in different cache nodes.

6) Transmission Time and Energy Efficiency of DCCC: Figs. 13 and 14 demonstrate the comparison results of DCCC, LRU, LFU, and AC in terms of the total transmission time and energy reduction rate with a different number of caching nodes. The experimental results expose that when adding caching nodes at the edge, the time-reducing rate of DCCC, which extends caching capacity by collaborating multiple distributed caching nodes, achieves up to 92.1%, while the time-reducing rates of LRU, LFU, and AC are only 61.6%, 63.8%, and 36.1%, respectively. The energy-reducing rate of DCCC is 94.8%, which is 57.4%, 30.8%, and 28.6% higher than AC (37.3%), LRU (64%) and LFU (66.2%), respectively. The improvement of DCCC in transmission time and energy efficiency mainly benefits from the high cache hit rate of DCCC, which effectively utilizes the cooperative storage resources of multiple caching nodes.

7) Scalability of DCCC: Fig. 15 presents the performance of our ICE and the other three approaches in terms of cache hit rate as the number of caching nodes grows. Overall, the cache hit rates of the four approaches will increase as the number of caching nodes grows. Comparatively, as shown in Fig. 15, with the expansion of the system scale, DCCC achieves the fastest increase in cache hit ratio. For example, when the number of

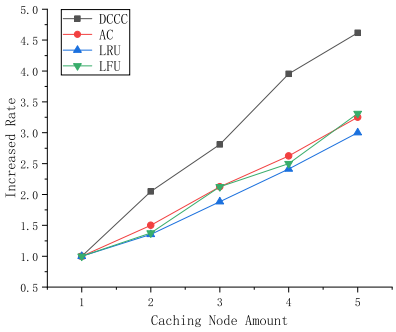

Fig. 15. DCCC increased hit rate.

caching nodes is increased to five, the cache hit rate of DCCC increases by 4.6 times compared with that of a single node, where the hit rates of AC, LRU, and LFU only increase by 3.25 times, 3 times, and 3.31 times, respectively, for the same circumstance. As a multi-node cooperative caching mechanism based on distributed computing, the reason why DCCC can significantly improve the overall cache hit rate is mainly due to that DCCC can minimize the probability of contents being repeatedly cached on multiple different nodes with the assistance of routing tables and caching tables. More specifically, one user's request for the content n_i will be transferred to a unique caching node whose hash value is equal to the content's hash value h_i. As a result, all requests for the same content will be transferred to the same caching nodes, avoiding the same content being repeatedly cached on different nodes. Therefore, in DCCC, the same contents are ensured to be cached in the same caching nodes, and other caching nodes will obtain the contents from the corresponding adjacent caching nodes, which cache the content, rather than request contents from the core network. In this manner, as the amount of caching nodes increases, the space utilization of each caching node can be maximized.

8) Discussion: Although conventional caching strategies (e.g., LRU and LFU) are effective for similar-sized contents, they neglect the factor of transmission latency, which has a considerable impact on the caching efficiency in case the sizes of cached contents differ significantly. Although the natural deep reinforcement learning strategy (AC) possesses decision-making ability, the strategy can not distinguish which resources will be accessed again in the future. Furthermore, these existing algorithms may cache contents that may not be important and helpful in reducing waiting time and energy. Thus, they usually cannot improve the performance of edge caching well. Comparatively, our ICE scheme explores and considers the popularity and other valued information of data, and thus dramatically increases the overall cache hit ratio and reduces the energy consumption for transmission. Nevertheless, with the explosive growth of edge traffic, the cache space of a single caching node is too small to meet the increasing needs of end users' high QoE requirements. Aiming to achieve a higher caching efficiency, based on the single-node ICE scheme, we further design a new multi-node DCCC scheme, which can enable multiple nodes of cooperative caching. DCCC manages the requests assisted by the well-designed consistent hashing mechanism with good load balance, and ICE is executed in each caching node to improve the hit rate. In this way, the contents can be distributed more reasonably with the lowest content repetition rate.

VII. Conclusion

In this work, we investigated the edge caching problem. We first designed a new Newton's cooling law-based content popularity model, which prefers to choose more time-consuming and energy-consuming contents with higher priorities, and mathematically formulate the caching problem a multi-objective optimization problem that aims to maximize the reduction ratio of transmission delay and energy consumption. Then, we proposed a new DRL-based caching approach named ICE to effectively solve the formulated problem. Moreover, to further improve the overall caching efficiency, we further presented a distributed computing-based multi-node cooperative caching scheme, named DCCC. DCCC enables multiple nodes to collaborate on joint caching and achieves good load balancing with our efficient consistent hashing mechanism and resource allocation scheme. Comprehensive experimental results demonstrate that, compared with the existing approaches, our DRL-based ICE scheme achieves better performance in minimizing the transmission delay and energy consumption, and our DCCC further significantly improves the overall cache hit rate of multiple caching nodes with higher resource utilization.

References

[1] T. Wang, J. Mao, M. Chen, G. Liu, J. Di, and S. Yu, "ICE: Intelligent caching at the edge," in *Proc. IEEE Glob. Commun. Conf.*, 2021, pp. 01–06.

[2] L. Wang, Y. L. Che, J. Long, L. Duan, and K. Wu, "Multiple access mmWave design for UAV-aided 5G communications," *IEEE Wirel. Commun.*, vol. 26, no. 1, pp. 64–71, Feb. 2019.

[3] A. Tian et al., "Efficient federated DRL-based cooperative caching for mobile edge networks," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 1, pp. 246–260, Mar. 2023.

[4] Z. Yu, J. Hu, G. Min, Z. Zhao, W. Miao, and M. S. Hossain, "Mobility-aware proactive edge caching for connected vehicles using federated learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 8, pp. 5341–5351, Aug. 2021.

[5] D. R.-J. G.-J. Rydning, R. John, and G. John, "The digitization of the world from edge to core," *Framingham: Int. Data Corporation*, vol. 16, pp. 1–28, 2018.

[6] C. Stergiou, K. E. Psannis, B. G. Kim, and B. Gupta, "Secure integration of IoT and cloud computing," *Future Gener. Comput. Syst.*, vol. 78, no. PT.3, pp. 964–975, 2016.

[7] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Gener. Comput. Syst.*, vol. 79, pp. 849–861, 2018.

[8] A. Abouaomar, A. Filali, and A. Kobbane, "Caching, device-to-device and fog computing in 5th cellular networks generation : Survey," in *Proc. Int. Conf. Wirel. Netw. Mobile Commun.*, 2017, pp. 1–6.

[9] Soubhik and Chakraborty, "Algorithmic nuggets in content delivery," *Comput. Rev.*, vol. 57, no. 2, pp. 103–103, 2016.

[10] 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.

[11] Z. Su, Q. Xu, F. Hou, Q. Yang, and Q. Qi, "Edge caching for layered video contents in mobile social networks," *IEEE Trans. Multimedia*, vol. 19, no. 10, pp. 2210–2221, Oct. 2017.

[12] K. Geetha and N. A. Gounden, "Dynamic semantic LFU policy with victim tracer (DSLV): A customizing technique for client cache," *Arabian J. Sci. Eng.*, vol. 42, pp. 725–737, 2017.

[13] T. G. Hendrantoro and A. Affandi, "Early result from adaptive combination of LRU, LFU and FIFO to improve cache server performance in telecommunication network," in *Proc. Int. Seminar Intell. Technol. Appl.*, 2015, pp. 429–432.
[14] B. Jiang, P. Nain, and D. Towsley, "LRU cache under stationary requests," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 2, pp. 24–26, 2017.
[15] M. C. Gursoy, C. Zhong, and S. Velipasalar, "Deep multi-agent reinforcement learning for cooperative edge caching," *Mach. Learn. Future Wirel. Commun.*, John Wiley & Sons, Ltd., ch. 21, pp. 439–457, 2020. [Online]. Available: https://doi.org/10.1002/9781119562306.ch21
[16] D. Qiao, S. Guo, D. Liu, S. Long, P. Zhou, and Z. Li, "Adaptive federated deep reinforcement learning for proactive content caching in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 4767–4782, Dec. 2022.
[17] Q. Fan, X. Li, J. Li, Q. He, K. Wang, and J. Wen, "PA-cache: Evolving learning-based popularity-aware content caching in edge networks," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, pp. 1746–1757, Jun. 2021.
[18] N. Nomikos, S. Zoupanos, T. Charalambous, and I. Krikidis, "A survey on reinforcement learning-aided caching in heterogeneous mobile edge networks," *IEEE Access*, vol. 10, pp. 4380–4413, 2022.
[19] W.-X. Liu, J. Zhang, Z.-W. Liang, L.-X. Peng, and J. Cai, "Content popularity prediction and caching for ICN: A deep learning approach with sdn," *IEEE Access*, vol. 6, pp. 5075–5089, 2017.
[20] Y. Im, P. Prahladan, T. H. Kim, Y. G. Hong, and S. Ha, "SNN-cache: A practical machine learning-based caching system utilizing the inter-relationships of requests," in *Proc. 52nd Annu. Conf. Inf. Sci. Syst.*, 2018, pp. 1–6.
[21] K. C. Tsai, L. Wang, and Z. Han, "Mobile social media networks caching with convolutional neural network," in *Proc. IEEE Wirel. Commun. Netw. Conf. Workshops*, 2018, pp. 83–88.
[22] A. Lekharu, M. Jain, A. Sur, and A. Sarkar, "Deep learning model for content aware caching at MEC servers," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 2, pp. 1413–1425, Jun. 2022.
[23] Y. Zhang, Y. Li, R. Wang, J. Lu, X. Ma, and M. Qiu, "PSAC: Proactive sequence-aware content caching via deep learning at the network edge," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 4, pp. 2145–2154, Fourth Quarter 2020.
[24] N. Kumar, S. N. Swain, and C. Siva Ram Murthy, "A novel distributed Q-learning based resource reservation framework for facilitating D2D content access requests in LTE-a networks," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 2, pp. 718–731, Jun. 2018.
[25] S. Liu, C. Zheng, Y. Huang, and T. Q. S. Quek, "Distributed reinforcement learning for privacy-preserving dynamic edge caching," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 3, pp. 749–760, Mar. 2022.
[26] H. Zheng, H. Zhou, N. Wang, P. Chen, and S. Xu, "Reinforcement learning for energy-efficient edge caching in mobile edge networks," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2021, pp. 1–6.
[27] R. Wang, Z. Kan, Y. Cui, D. Wu, and Y. Zhen, "Cooperative caching strategy with content request prediction in Internet of Vehicles," *IEEE Internet Things J.*, vol. 8, no. 11, pp. 8964–8975, Jun. 2021.
[28] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep reinforcement learning-based edge caching in wireless networks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 6, no. 1, pp. 48–61, Mar. 2020.
[29] 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. 10 190–10 203, Nov. 2018.
[30] Z. Song et al., "Learning relaxed belady for content distribution network caching," in *Proc. 17th USENIX Symp. Netw. Syst. Des. Implementation*, 2020, pp. 529–544.
[31] J. Yan, Y. Jiang, F. Zheng, F. R. Yu, and X. You, "Distributed edge caching with content recommendation in fog-RANs via deep reinforcement learning," in *Proc. IEEE Int. Conf. Commun. Workshops*, 2020, pp. 1–6.
[32] 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.
[33] 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.
[34] S. Chen, Z. Yao, X. Jiang, J. Yang, and L. Hanzo, "Multi-agent deep reinforcement learning-based cooperative edge caching for ultra-dense next-generation networks," *IEEE Trans. Commun.*, vol. 69, no. 4, pp. 2441–2456, Apr. 2021.
[35] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Online collaborative data caching in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 02, pp. 281–294, Feb. 2021.
[36] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading tasks with dependency and service caching in mobile edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 11, pp. 2777–2792, Nov. 2021.
[37] H. Wang, G. Tang, K. Wu, and J. Wang, "PLVER: Joint stable allocation and content replication for edge-assisted live video delivery," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 01, pp. 218–230, Jan. 2022.
[38] X. Wang, C. Wang, X. Li, V. C. M. Leung, and T. Taleb, "Federated deep reinforcement learning for internet of things with decentralized cooperative edge caching," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9441–9455, Oct. 2020.
[39] Z. Zhao, W. Zhou, D. Dan, J. Xia, and L. Fan, "Intelligent mobile edge computing with pricing in Internet of Things," *IEEE Access*, vol. 8, pp. 37727–37735, 2020.
[40] Google, Tensorflow1.9.0, 2018. [Online]. Available: https://github.com/tensorflow/tensorflow/releases/tag/v1.9.0
[41] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Adv. Neural Inf. Process. Syst.*, vol. 12, pp. 1057–1063, 1999.

Ting Wang (Senior Member, IEEE) received the PhD degree in computer science and engineering from the Hong Kong University of Science and Technology, Hong Kong, China, in 2015. He is currently an associate professor with the Software Engineering Institute, East China Normal University, Shanghai, China. Prior to joining ECNU in 2020, he worked at Bell Labs as a research scientist from 2015 to 2016, and at Huawei as a senior engineer from 2016 to 2020. His research interests include cloud/edge computing, data center networks, machine learning, and AI-aided intelligent networking.

Yuxiang Deng received the bachelor's degree in software engineering from the Hefei University of Technology (HFUT), Hefei, China, in 2022. He is currently working toward the master's degree with Software Engineering Institute, ECNU, Shanghai, China. His research interests include cloud computing, edge computing, and machine learning.

Jiawei Mao received the bachelor's degree in computer science from Nantong University, China, in 2019, and the master's degree with Software Engineering from East China Normal University, Shanghai, China, in 2022. His research interests include cloud computing, edge computing, and machine learning.

Mingsong Chen (Senior Member, IEEE) received the PhD degree in computer engineering from the University of Florida, Gainesville, in 2010. He is currently a professor with the Software Engineering Institute at East China Normal University. His research interests include cloud computing, design automation of cyber-physical systems, parallel and distributed systems, and formal verification techniques. Currently, he serves as the director of MoE Engineering Research Center of Software/Hardware Codesign Technology and Application, and the vice director of technical committee of embedded systems of China Computer Federation (CCF). He is an associate editor of IET Computers & Digital Techniques, and Journal of Circuits, Systems and Computers.

Gang Liu (Member, IEEE) received the BS and PhD degree in computer science from Northwestern Polytechnical University in 2002 and 2008. Currently, he works as a research scientist with Bell Labs China since 2008. His research interests include SDN/NFV, edge/cloud computing, and AI-assisted cellular networks.

Jieming Di received the MS degree in computer science from the Hong Kong University of Science and Technology, Hong Kong, China, in 2013. He worked in EMC, Microsoft as senior software engineer for many years. Currently, he is a software engineer with Meta (formerly named Facebook) since 2022. His research interests include software defined networks, Big Data, natural language processing (NLP), and GenAI.

Keqin Li (Fellow, IEEE) is a SUNY distinguished professor of computer science with the State University of New York. He is also a distinguished professor with Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, embedded systems and cyber-physical systems, heterogeneous computing systems, Big Data computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has published more than 680 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He currently serves or has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Services Computing*, and *IEEE Transactions on Sustainable Computing*.