A Cooperative Caching System in Heterogeneous Edge Networks

Junkun Peng, Qing Li, *Senior Member, IEEE*, Xun Tang, Dan Zhao, Chuang Hu, and Yong Jiang, *Member, IEEE*

***Abstract*—Recently, the rapid growth of video content and the increasing demand for high Quality of Experience (QoE) have significantly strained the backbone network. Edge caching is a promising approach to alleviate the strain by caching content closer to users. However, it confronts challenges stemming from the low capability of individual edge nodes and the high density of their distribution, resulting in low hit ratios and unbalanced workloads. In this paper, we conduct in-depth analyses of these challenges and formulate a typical cooperative edge caching problem. Based on the insights, we introduce MagNet, a cooperative edge caching system featuring two key mechanisms: Automatic Content Congregating (ACC) and Mutual Assistance Group (MAG). ACC improves hit ratios by intelligently guiding requests to their optimal edges, thereby facilitating content aggregation. Complementing this, Quick Cache is implemented to accelerate this congregation process by prefetching content and optimizing cache space, effectively boosting hit ratios. MAG, on the other hand, achieves workload balance by dynamically forming groups to augment edge capabilities and redistribute requests on overloaded edges. To elucidate the design principles of MagNet, we conduct detailed component-level comparisons and quantitative analyses. To validate the overall performance, we compare it with various caching solutions using real-world datasets, demonstrating significant performance improvements.**

***Index Terms*—Edge computing, cache, cooperative, workload balance, embedding.**

I. Introduction

IN RECENT years, the backbone network has faced severe pressure. According to the Cisco and Sandvine report, video streaming constitutes a significant portion of all Internet business traffic and is consistently increasing [1], [2]. This mainly arises from the rapid growth of the network users, the upgrade of video quality from traditional 1080P high-definition content to 4 K and 8 K levels [3], and the emerging of sophisticated human-computer interactions like 360 videos [4], VR [5], and AR [2].

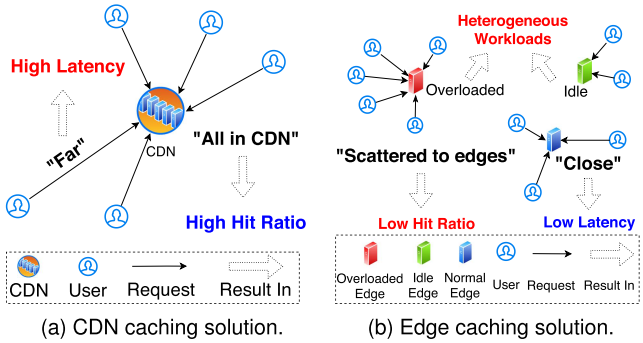


Fig. 1. CDN caching and the edge caching solution.

One unique property of the traffic is that content requests are highly concentrated [6]. Thus, the caching technology is an essential solution [7] to alleviate the pressure. It can reduce requests' latency, which can significantly improve users' Quality of Experience (QoE) [8], and relieve the heavy burden of the network transmission [9]. There are two types of caching solutions: 1) mature Content Delivery Network (CDN) [10], [11] solutions; and 2) edge caching solutions. CDN solutions consist of some super-powerful nodes, each of which can serve a large number of requests in a wide-range area. However, the super-powerful nodes are limited, even for giant companies, e.g., Akamai and Google [12], [13]. Thus, as shown in Fig. 1(a), all requests must be aggregated to the central node, which causes high latency [14] and low QoE [15].

The edge technology [16] provides a new perspective for content delivery and caching [9]. Edges can be used to significantly reduce the distances between contents and users, which can save enormous transmission traffic in the backbone and lower the latency effectively [17]. However, as the distribution of edges is dense and their serving areas are small, requests are scattered to different edges, resulting in a low hit ratio. In addition, edges suffer from unbalanced workloads due to the highly dynamic requests made by users with diverse distribution, content preferences, and habits. The limited and heterogeneous capacities of edges make the workload balance even worse.

Manuscript received 14 July 2023; revised 12 November 2023; accepted 20 November 2023. Date of publication 29 November 2023; date of current version 5 June 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB3105000, in part by the National Natural Science Foundation of China under Grant 61972189, in part by the Major Key Project of PCL under Grant PCL2023A06, and in part by the Shenzhen Key Lab of Software Defned Networking under Grant ZDSYS20140509172959989. Recommended for acceptance by X. Costa-Perez. *(Corresponding authors: Dan Zhao; Yong Jiang.)*

Junkun Peng and Yong Jiang are with the Shenzhen International Graduate School, Tsinghua University, Shenzhen, Guangdong 518055, China, and also with the Peng Cheng Laboratory, Shenzhen, Guangdong 518055, China (e-mail: pjk20@mails.tsinghua.edu.cn; jiangy@sz.tsinghua.edu.cn).

Qing Li and Dan Zhao are with the Peng Cheng Laboratory, Shenzhen, Guangdong 518055, China (e-mail: liq@pcl.ac.cn; zhaod01@pcl.ac.cn).

Xun Tang is with the Sun Yat-Sen University, Shenzhen, Guangdong 518107, China (e-mail: tangx66@mail2.sysu.edu.cn).

Chuang Hu is with the Wuhan University, Wuhan, Hubei 430072, China (e-mail: handc@whu.edu.cn).

Digital Object Identifier 10.1109/TMC.2023.3336955

To address the two challenges of edge caching solutions discussed above, we formulate a typical cooperative edge caching problem which jointly optimizes the latency, traffic, and workload balance. The problem is proved to be an NP-complete problem and is difficult to solve directly. To this end, we propose a heuristic approach, MagNet, to solve this problem.[1] Through cooperation between edges, more and more user requests can be guided to the optimal edges. Each edge node is like a magnet, forming the entire magnetic network. MagNet has two innovative mechanisms to address the two challenges: 1) the Automatic Content Congregating (ACC) mechanism. First, the ACC utilizes a neural embedding algorithm to generate vectors for all contents, which tries to capture underlying patterns of historical requests. Second, a novel clustering algorithm is designed to cluster the vectors into some types. Third, the ACC guides requests to their optimal edges by type so that requests of the same type tend to congregate in the same edges. As a result, each edge accumulates contents of one dominant type more than other types. Then, more requests of this type are attracted to the edge. This process forms a virtuous circle between edges and requests, which eventually leads to a high hit ratio. Besides, we also design the Quick Cache within the ACC mechanism. The Quick Cache includes the Hot Content Prefetch (HCP) and the Access Rule (AR), which improve the cache efficiency by populating the cache with trending content in place of the evicted unpopular content. In this way, the virtuous circle can be further accelerated. 2) the Mutual Assistance Group (MAG) mechanism. When an overloaded edge emerges, the MAG finds some nearby idle edges to form a temporary group to assist the overloaded edge. The group runs in a master-worker mode where the master, i.e., the overloaded edge, shifts some of its workloads to the workers, i.e., idle edges, according to their capacities to balance their workloads. In extreme situations where none of the nearby edges are eligible for assistance, the MAG will mitigate the workload by reducing the number of attracted requests. The group dismisses when the overload problem is eliminated.

To evaluate the MagNet performance, we conduct experiments from three different perspectives using a real-world dataset. We compare MagNet with some benchmark caching solutions, including classical, ML-based, and cooperative solutions. The result shows that MagNet can improve the hit ratio from 40% and 60% to 78% for non-cooperative and cooperative solutions, respectively, and significantly balance the edges' workloads. Additionally, we also conduct experiments to analyze components in MagNet including the Quick Cache and the clustering algorithm. We prove the contribution of the Quick Cache to the overall performance through ablative analysis. Apart from the FCA used in MagNet, we also design two other clustering algorithms and evaluate the performance of the three algorithms in hit ratio and workload balance through comparative experiments.

[1]This work was presented in part at the 2022 ACM Web Conference (WWW) [18]. This extended version integrates significant enhancements, including the Quick Cache and the enhanced workload balance mechanism, accompanying several more in-depth experimental analyses.

TABLE I
DATASETS

Name	Time Range	Amount	Area
Dataset1	13 days	50 million	Beijing City
Dataset2	50 days	12 billion	Anhui Province

II. Data Analyses and Challenges Reveal

In this section, two real-world datasets are analyzed to reveal the challenges of edge caching solutions, and further, to help find the underlying insights. As summarized in Table I, both datasets contain amounts of content requests collected from the Chinese top content providers. For Dataset1, each request contains a user ID, timestamp, latitude, longitude, and content ID. For Dataset2, each request contains a request IP, timestamp, content ID, and a boolean value indicating whether this request hit.

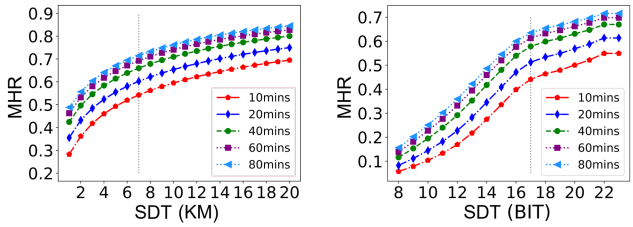
A. Low Hit Ratio Challenge

As illustrated in Fig. 1, in contrast with a CDN node that serves a large area, each edge's serving area is small since dense distribution and limited caching capacity. As such, each edge aggregates fewer requests, which results in a low hit ratio of edge caching solutions.

To reveal the low hit ratio challenge, we compare the edge caching method and the CDN method on Dataset1. A classical CDN solution which uses one powerful node to handle requests, as Fig. 1(a) shows, is used to represent the CDN method. A primary edge caching solution where requests are sent to only its closest edge, as Fig. 1(b) shows, is used to represent the edge caching method. In Dataset1, we filter out one day's requests in a $20\ \mathrm{KM} \times 20\ \mathrm{KM}$ experiment area of Beijing, and get 1,220,720 requests. These requests are sent orderly based on their timestamps. For both methods, we assume all their nodes have enough caching capacity to cache all the requested contents.

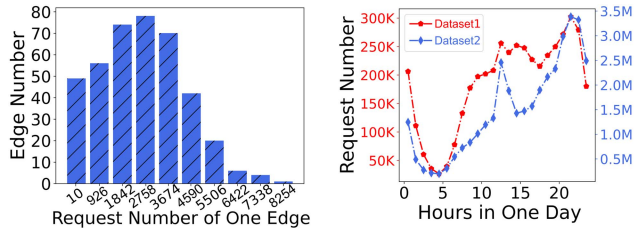
For the CDN solution, a hit ratio of 91.7% can be achieved as all requests are directed to the single CDN node. For the edge caching solution, we assume there are edges distributed 1KM apart from each other, which means there are 400 edges distributed evenly in the experiment area. Even if edges have enough caching capacity, the edge caching method has a hit ratio of only 55.99%. The big gap in the hit ratio, about 35.7%, reveals the edge caching method has the serious challenge of low hit ratio.

Compared to CDN solutions where all requests are directed to only one node, in edge solution, requests are scattered to the nearest edge to each user. This may seem reasonable from certain perspectives such as latency and content preference. However, this inflexible rule can cause a low hit ratio in some situations. For instance, some requests may not find the cached content in the closest edge which could be present in the sub-nearest edges nearby. In such scenarios, providing requests with more edge choices instead of confining them to only the closest edges could theoretically yield a high hit ratio. To prove this idea, here we utilize a quantitative method to analyze the relationship between the hit ratio and search area from the perspective of



(a) SDT-MHR in Dataset1. (b) SDT-MHR in Dataset2.

Fig. 2. Relationship between SDT and MHR.



(a) Request number histogram. (b) Request number in one day.

Fig. 3. Request distribution in one day.

each request, which is represented by the Maximum Hit Ratio (MHR) and the Search Distance Threshold (SDT), respectively.

First, we give the definition of MHR. For a request $r^i_{t,l}$, i, t, l stand for its content ID, request timestamp and location, respectively. Let $r^i_{t,l}$ denote the Potential Hit Request (PHR) $phr^i_{t,l}$. It exists only if there is another request $r^j_{t',l'}$ which satisfied conditions in all three dimensions: 1) in the content dimension, $i = j$, which means their content IDs should be the same; 2) in the temporal dimension, $0 < t - t' < TT$, which means $r^j_{t',l'}$ must exist before $r^i_{t,l}$ but not early than Time Threshold (TT); and 3) in the spatial dimension, $distance(l, l') < SDT$, which means their distance should be less than the Search Distance Threshold (SDT). More specifically, this distance is the network distance. For Dataset1, we use the geographical location to calculate the distance, as we can find a strong positive correlation between geographical distance and network distance. For Dataset2, we use the IP differences, as most IP segmentation are made considering their network distance [19]. After defining the PHR, the MHR ($MHR^{t_0,TP}_{TT,SDT}$) can be calculated with the two datasets. For $MHR^{t_0,TP}_{TT,SDT}$, its PHR should be satisfied with the SDT and the TT, and the request range continues for a Time Period (TP) from t_0. The total request is $TR_{t_0,TP}$ which also starts from t_0 , and continues for the TP. It can be formulated as:

$$MHR^{t_0,TP}_{TT,SDT} = \frac{\left\| \left\{ phr^i_{t,l} \mid phr^i_{t,l} \in TR_{t_0,TP} \right\} \right\|}{\| TR_{t_0,TP} \|}. \quad (1)$$

MHR represents the best hit ratio that an edge solution can achieve under specific parameters. When analyzing Dataset1, t_0 is set to be 17:00, and TP is set to be six hours, since the requests amount of this period is the highest in the whole day, which will be proved in Section II-B. It is easy to explain that these six hours are off work time, and people want to relax. In these six hours, the network is under high pressure, which drives us to analyze this period and reduce the network pressure. When analyzing the relationship, the requests in the whole Beijing city are all included to get a universal result. Besides, TT is set as one hour and SDT is set from 1 KM to 20 KM. The analysis result shown in Fig. 2(a) shows that the MHR increases as the SDT becomes bigger in general. It is also obvious that the trend of MHR becomes flat when the SDT is greater than 7 KM. When the TT is fixed as 60 minutes, only if the SDT is bigger than 7KM, the hit ratio has a chance to be greater than 70%.

Next, Dataset2 is used to continue the analysis. For geographic area, the whole Anhui province's requests are used. Similarly to the analysis of Dataset1, t_0 is also set to be 17:00 and TP to be six hours here for the high-pressure feature. To understand the impact of the SDT, it is set from 8 bits to 24 bits, where we use IP difference as the distance tool. More specifically, the XOR method is used to quantify the binary IPV4 address distance with the below equation:

$$distance(IP1, IP2) = \sum_{i=1}^{32} (IP1_i \oplus IP2_i) \quad (2)$$

Fig. 2(b) shows a consistent result that the MHR first increases and then becomes flat as the SDT increases. Only if the SDT is greater than 17 bits, the MHR is possible to be greater than 60% in the condition of TT as 60 minutes.

The above analyses of two datasets first demonstrate that there exists a high MHR, indicating a great potential for edge solutions to achieve a high hit ratio. Furthermore, the analyses also show that a large SDT is essential for achieving a high MHR. This confirms our previous hypothesis that providing more edge choices, rather than limiting requests to only the closest edge, can improve the hit ratio performance. However, providing more edge choices for each user request further raises a question: how to find the optimal edge among these edge choices to reach the best hit ratio? In this paper, we propose to address this issue through cooperation between edges.

B. Unbalanced Workloads Challenge

Edge caching solutions also suffer unbalanced workloads due to the limited and heterogeneous capacities and dynamic requests. First, most edges are limited in caching and computational capacities [9], [20], making them more sensitive to caching solutions. The limited caching capacity leads to competition between popular contents. The limited computational capacity causes excessive requests to be blocked. Therefore, it is necessary to allocate an appropriate number of requests for each edge. Moreover, users have varying location distributions, preferences, and habits, which results in their requests being unevenly distributed over time and space [21]. Through our analyses, this can be proved from a spatial perspective that the requests' number varies significantly in different edge serving areas. Additionally, dynamic changes in the time dimension further aggravate the unbalance of the entire situation.

We use Dataset1 to analyze the uneven distribution over space. We randomly choose one day and count the number of requests served by these 400 edges during the entire day. Fig. 3(a) shows

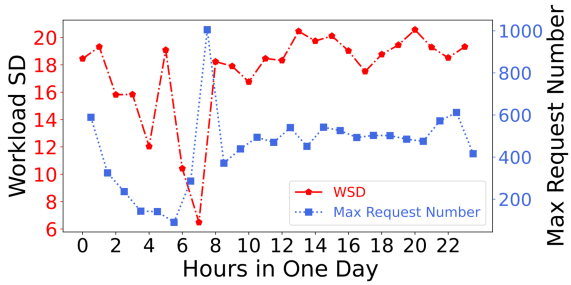


Fig. 4. Workload of 400 edges in one day.

that the number of requests served by different edges is quite different. Among these edges, the highest number of requests severed by a single edge per day can reach more than 7,000, while the lowest is only 10. These significant differences show the unbalanced workloads for these 400 edges in one day.

In the temporal dimension, we have a more fine-grained analysis with both Dataset1 and Dataset2. First, Fig. 3(b) illustrates the request number fluctuates significantly across a whole day. Then, Fig. 4 measures every hour's workload for these 400 edges in one day using the Workload Standard Deviation (WSD). In addition, the requests' number of the most overloaded edge is also counted in each hour. As can be seen, the WSD is always high except for 6:00 and 7:00. There are 21 hours in a day with a WSD of more than 14. The period from 22:00 to 23:00 is the requests' peak time for the 400 edges, while its dynamic is also very high. Only several hours in the early morning are relatively workload-balanced.

III. System Design

In this section, we first introduce the network topology and formulate a typical cooperative edge caching problem. Then, we propose MagNet which is a cooperative edge caching system, and present its framework.

A. Problem Formulation

1) Network Topology: The considered system includes a cloud server e_0 containing all the contents, E edges, C contents, and U users. Let $\mathcal{E} = \{e_1, e_2, \ldots, e_i, \ldots, e_E\}$ denote the set of edges. Let f_i denote the finite caching capacity of edge e_i. Edges can exchange caching information with the surrounding edges called "neighbors". Let $\mathcal{N}_i = \{e_{i_1}, e_{i_2}, \ldots, e_{i_{m_i}}\}$ denote the neighbors of edge e_i, where m_i is the number of neighbors and $\mathcal{N}_i \subset \mathcal{E}$. Let $\mathcal{C} = \{c_1, c_2, \ldots, c_j, \ldots, c_C\}$ denote the set of contents and s_j be the size of transmission with content c_j. Especially, let s_0 denote the size of transmission without any content. Let $\mathcal{U} = \{u_1, u_2, \ldots, u_g, \ldots, u_U\}$ denote the set of users.

2) Load Constraint Modeling: We consider a system operating over a finite time horizon. The requests $\mathcal{R} = [r_1, r_2, \ldots, r_p, \ldots, r_R]$ are generated by users in order, where R is the number of requests. Let $e_{p'}$[2] denote the home edge which is the closest edge to the user of r_p. And let $c_{p''}$ and $u_{p'''}$ denote the content and the user of r_p. In this Edge Caching problem, the request r_p is sent first to its home edge $e_{p'}$. If it hits, i.e., content $c_{p''}$ is cached in $e_{p'}$, then $c_{p''}$ is returned by $e_{p'}$; otherwise, $e_{p'}$ sends r_p to some neighbors in $\mathcal{N}_{p'}$. If it still misses at the neighbor edge $e_{p'_m}$, the request is sent to e_0 to fetched $c_{p''}$, which then will be cached in $e_{p'_m}$ and sent back to user $u_{p'''}$. To measure the workload of e_i, the set of requests sent to e_i is denoted as $\mathcal{R}_i = [r_{i_1}, r_{i_2}, \ldots, r_{i_{R_i}}]$, where R_i is the number of requests received by e_i.

In this paper, we use LRU [22] as the default cache replacement strategy for edges.[3]

Given the caching capacity f_i and the edge's history requests $[r_{i_1}, r_{i_2}, \ldots, r_{i_k}]$, the current edge caching contents is given as $LRU(f_i, [r_{i_1}, r_{i_2}, \ldots, r_{i_k}]) = UNIQ_{y^*}$, where $UNIQ$ is the set of unique request elements, $UNIQ_y = \{r_x | r_x \in [r_{i_y}, r_{i_{y+1}}, \ldots, r_{i_k}]\}$ and $y^* = \min\{y || UNIQ_y| \leq f_i\}$. Let $l(a, b, s)$ be the latency function of the element a and b where $a, b \in \{e_0\} \cup \mathcal{E} \cup \mathcal{U}$, and s is the size of the content. Let $y_i^p \in \{0, 1\}$ denote whether r_p hits in e_i:

$$y_i^p = \begin{cases} 1 & if\ c_{p''} \in LRU\left(f_i, [r_{i_1}, r_{i_2}, \ldots, r_{i_k}]\right) \\ 0 & else \end{cases}. \quad (3)$$

The edge e_i has limited caching capacity, therefore

$$|LRU\left(f_i, [r_{i_1}, r_{i_2}, \ldots, r_{i_k}]\right)| \leq f_i, \forall e_i \in \mathcal{E}. \quad (4)$$

The cloud server e_0 contains all the contents, therefore

$$y_0^p = 1, \forall r_p \in \mathcal{R}. \quad (5)$$

Let S_{CST} denote the size of the Cache Summary Table (CST) that is exchanged between edges to notify each other their cache summaries. Let EUP denote the period that an edge exchange the CST with its neighbors. Let RT denote the running time of the system.

Latency: The latency occurs during the following 3 steps. *Step 1:* When a request r_p is sent to its home edge $e_{p'}$, it either hits ($y_{p'}^p = 1$) or miss ($y_{p'}^p = 0$). The latency of this step is $l_{home}^h = l(u_{p'''}, e_{p'}, s_{p''})$ if it hits, or $l_{home}^m = l(u_{p'''}, e_{p'}, s_0)$ if it does not hit. *Step 2:* When the request is sent to neighbors one by one, if it hits in one neighbor $e_{p'_m}$, the latency of this step is,

$$l_{nei}^h = q^0 \left(\sum_{\substack{e_i \in \mathcal{N}_{p'} \\ e_i \neq e_{p'}}} x_i^p l\left(u_{p'''}, e_i, s_0\right) + l\left(u_{p'''}, e_{p'}, s_{p''}\right) \right). \quad (6)$$

where q is the number of neighbors that r_p is sent to. Let $x_i^p \in \{0, 1\}$ denote whether r_p is sent to e_i. q satisfies $\sum_{e_i \in \mathcal{N}_{p'}} x_i^p = q$ and

$$0 \leq q \leq |\mathcal{N}_{p'}| < E. \quad (7)$$

If it does not hit in any neighbor, the latency of this step is given as $l_{nei}^m = q^0 \sum_{e_i \in \mathcal{N}_{p'}} x_i^p l(u_{p'''}, e_i, s_0)$. In particular, $q = 0$ indicates r_p is not forwarded to any neighbor. Whether r_q hits or not in neighbors can be formulated as $h_{nei} = \prod_{e_i \in \mathcal{N}_{p'}} x_i^p y_i^p$.

[2]For consistent using the edge symbol e_i, the content symbol c_j, the user symbol u_g, and the request symbol r_p, the p', p'' and p''' are used to denote the indexes of the home edge, the content and the user of this request r_p, respectively.

[3]Other replacement strategies can also be used. A comparison of replacement strategies is presented in Section III-B.

Step 3: If r_p is sent to the cloud server by neighbor $e_{p'_m}$, the latency of this step is $l_{e_0} = l(e_i, e_0, s_{p''}) + l(e_i, u_{p'''}, s_{p''})$. Summing up the latency incurred in the above 3 steps, the total latency of request r_j is given as: $l_j = y^p_{p'} l^h_{home} + (1 - y^p_{p'})(h_{nei} l^h_{nei} + (1 - h_{nei})(l^m_{nei} + l_{e_0}))$. The sum latency L of all requests in a period is $L = \sum_{j \in \mathcal{R}} l_j$.

Traffic: There are two types of traffic in the system, the Backbone Traffic (BT) and the Local Traffic (LT). The BT only occurs when contents need to be fetched from the cloud server e_0, given as $BT = \sum_{p \in \mathcal{R}} (1 - y^p_{p'})(1 - h_{nei}) s_{p''}$. The LT is given as $LT = \sum_{p \in \mathcal{R}} s_{p''} + (RT/EUP) \sum_{e_i \in \mathcal{E}} \sum_{e_{i_m} \in N_i} S_{CST}$.

Workload Balance: The Workload Standard Deviation (WSD) is calculated in the following four steps to measure the workload balance status:

- Step 1: because the number of requests varies over time, normalize request numbers into [0,100] interval by the min-max normalization: $R'_i = \frac{R_i - \min(R_a)}{\max(R_b) - \min(R_a)} * 100, \forall a, b \in \mathcal{E}$.
- Step 2: calculate the average capacity: $f_{avg} = \frac{\sum_{e_j \in \mathcal{E}} f_j}{E}$.
- Step 3: because of edges' heterogeneous capacities, normalized numbers are processed considering capacities:

$$R''_i = \frac{R'_i}{\frac{f_i}{f_{avg}}}.$$

- Step 4: calculate WSD:

$$WSD = \sqrt{\frac{\sum\limits_{e_i \in \mathcal{E}} \left(R''_i - \frac{\sum_{e_k \in \mathcal{E}} R''_k}{E} \right)^2}{E}}. \tag{8}$$

3) Cooperative Edge Caching Problem. Problem 1: Given α, β, γ, θ, $\mathcal{R}$, $\mathcal{E}$, $\mathcal{C}$, $\mathcal{U}$, s_0, S_{CST} and RT, find x^p_i and q to jointly minimize latency, traffic and WSD as follows:

$$\begin{aligned} \min: \ & \alpha L + \beta BT + \gamma LT + \theta WSD \\ \text{subject to}: \ & (3), (4), (5) \text{ and } (7). \end{aligned} \tag{9}$$

4) Complexity Analysis: To analyze the complexity of Problem 1, we first consider Problem 1′, a simplified case of Problem 1 where we assume requests' orders are already settled down. In this case, if a specific solution is given, the overall result of Problem 1 can be calculated in a polynomial time. Moreover, the main contribution of this problem is to choose the neighbor, which can be represented by $y^p_{p'_m}$. This simplified problem can be converted to the Helper Decision Problem [23] in a polynomial time. As the Helper Decision Problem has been proved as an NP-complete problem, the Problem 1′ is an NP-complete problem. Therefore, the more complex Problem 1 is at least an NP-complete problem. For now, there is no efficient polynomial solution for this problem. In this paper, we try to solve it heuristically and propose a novel solution named MagNet.

B. MagNet Framework

MagNet is a cooperative edge caching system. This system can guide requests to their optimal edges and achieve a high hit ratio and balanced workloads. The guidance is based on content type, called the Guidance Type (GT). MagNet clusters all contents into some GTs, which can be done offline and periodically, e.g., per day.

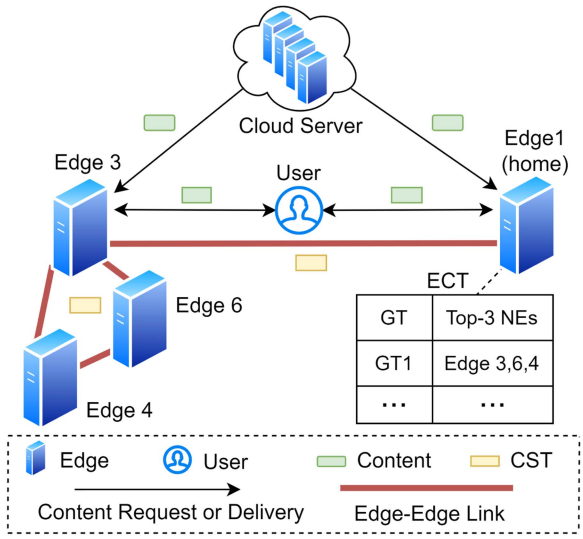


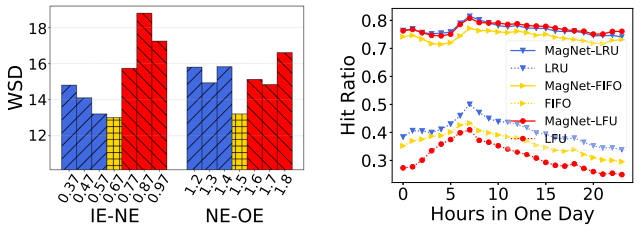
Fig. 5. Framework of MagNet.

In MagNet, each edge maintains its own Cache Summary Table (CST), which records the cached size of each GT in the edge. At runtime, the edge exchanges its CST with its neighbors periodically to update the cashing information in its neighbors. Based on its and neighbors' CSTs, the edge builds and updates a table named Edge Candidate Table (ECT). In ECT, the edge designates the Top-k Neighbor Edges (Top-k NEs) for each GT, where k represents the number of edge candidates with the largest GT sizes. As illustrated in Fig. 5, here Top-3 NEs are maintained for each GT in ECT. The selection of k values can be customized to accommodate different system needs. It involves trade-offs between hit ratio and latency, which will be analyzed detailedly in Section V-C3.

Every request in MagNet is processed in the following three stages:

- Stage 1: the user sends it to the home edge, the closest edge to the user. If the home edge contains the requested content, the content is returned directly to the user. Otherwise, the home edge suggests the user the Top-3 NEs to try next based on the content's GT, and the request goes to Stage 2.
- Stage 2: the user sequentially sends requests to the neighbor edges recommended by the home edge. If any of these neighbor edges contains the requested content, this traverse process is terminated and the content is promptly returned to the user. If none of the suggested neighbor edges possess the requested content, the request proceeds to Stage 3.
- Stage 3: the third suggested NE fetches the content from the cloud server, which stores all contents. Then, the third suggested NE caches the content locally [4] and sends it back to the user.

[4]All visited NEs will fetch and cache content locally based on the Access Rule, while the third suggested NE is responsible for sending back the requested content to the user.



(a) IE-NE/NE-OE thresholds. (b) Different caching strategies.

Fig. 6. Impacts of different IE-NE/NE-OE thresholds and different caching strategies.

Over every two minutes, each edge counts the number of received requests and updates its status. Particularly, we define two threshold, i.e., IE-NE threshold and NE-OE threshold. If the number of received requests is over IE-NE threshold times the expected service quantity, it is treated as an Overloaded Edge (OE); if the number of received requests is less than NE-OE threshold times the expected service quantity, it is treated as an Idle Edge (IE); otherwise, it is treated as a Normal Edge (NE). The expected service quantity is set considering each edge's caching and computational capacities. When an OE appears, it tries to ally with some Idle Edges (IEs) to form a temporary Mutual Assistance Group (MAG) to alleviate the overloaded situation.

We evaluate the impacts of different IE-NE thresholds and NE-OE thresholds on WSD, under the experimental setup introduced in Section V-A. Fig. 6(a) illustrates that the best overall system performance is achieved when the IE-NE threshold is set to 1.5 and the NE-OE threshold is set to 0.67. In the rest of this paper, we adopt these two thresholds. It is worth noting that these two thresholds can be adjusted to meet the requirements of different circumstances. Influence factors include varying request volumes within different regions, the total number of edges in the system, as well as the caching capabilities of each edge node.

The proposed MagNet framework is, in essence, agnostic to cache replacement strategies, making it modular in design. Various replacement strategies can be seamlessly integrated into MagNet, including Least Recently Used (LRU) [22], Least Frequently Used (LFU) [24], and First In First Out (FIFO). Notably, as a cooperative caching solution, MagNet consistently exhibits exemplary system performance across different replacement strategies. In Fig. 6(b), we evaluate three classical replacement strategies and compare the hit ratio performances before and after their integration into MagNet. It is evident that MagNet consistently maintains a significant performance advantage, though LRU slightly outperforms the other two strategies. In this paper, LRU is employed in edges as the cache replacement strategy.

IV. Two Mechanisms of MagNet

In this section, we introduce the two key mechanisms in the MagNet system: the Automatic Content Congregating (ACC) and the Mutual Assistance Group (MAG).

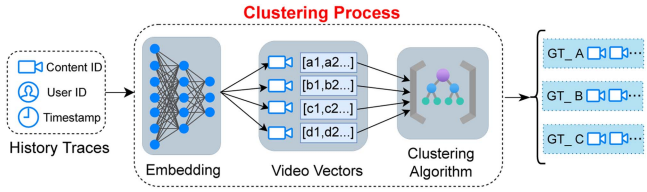
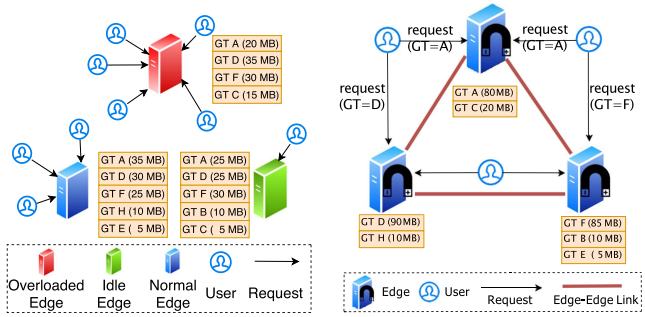


Fig. 7. Contents clustering process.



(a) Without ACC. (b) With ACC.

Fig. 8. Automatic content congregating (ACC).

A. Automatic Content Congregating (ACC)

As discussed in Section II, the low hit ratio caused by limited edge choices is a challenging problem in edge caching solutions. As shown in Fig. 8, the ACC is designed to guide requests to their optimal edges among multiple edge choices based on cooperation between edges.

1) Cluster Contents: In the ACC, contents are clustered and tagged with GT. The gist of clustering is to cluster the contents of consecutive requests into different GTs. As such, the ACC can use the GT to divert consecutive requests to different edges evenly. In the process, as illustrated in Fig. 7, the history requests are used to generate the vector for each content using embedding technology. Then, the vectors are clustered into some sets, which means the corresponding contents are clustered into some GTs.

Content Embedding: The critical step of clustering is to capture the underlying pattern of the history requests. To this end, we use the embedding technology [25] to get the vector for every content [26]. The embedding technology utilizes a neural embedding algorithm to conduct collaborative filtering [27].

This content embedding process only requires user ID, content ID, and timestamp as input features since it uses collaborative filtering to get the implicit relationship between contents. This facilitates the adaptability of MagNet.

Furthest Clustering Algorithm (FCA): Clustering the vectors generated by the embedding layer is essential, as clustering algorithms generate different GTs and then affect the content guidance. Under the assumption[5] that popular contents are divided into the same GT, the popular contents tend to compete for the limited caching capacity, causing excessive requests guided to a small number of edges, which results in unbalanced workloads. To avoid intra-GT competition, the FCA is proposed. The FCA makes vectors that are far apart from each other clustered in the

[5]We design another clustering algorithm with this assumption in Section V-C.

Algorithm 1: Furthest Clustering Algorithm.

Input: $vectors$ of contents, NGT
Output: GTs
Initialize: $GTs \leftarrow [\,][\,], i \leftarrow 0$
while $|vectors| \neq 0$ **do**
  for $j = 0$ to NGT **do**
    $v \leftarrow GetMaxDistanceVector(vectors, GTs[j])$
    $GTs[j][i++] \leftarrow v$
    $Remove(vectors, v)$
  end for
end while
return GTs

Algorithm 2: ECT Update Algorithm.

Input: $CSTs$ from neighbors, ECT, k
Output: ECT
for each GT in ECT **do**
  Initialize $ECT[GT].NEQ$ and $ECT[GT].sizeQ$ as min-priority queues
end for
for each CST in $CSTs$ **do**
  for each $entry$ in CST **do**
    if $entry.size > ECT[entry.GT].sizeQ.top()$ or $ECT[entry.GT].sizeQ.size() < k$ **then**
      $ECT[entry.GT].NEQ.push(CST.edge)$
      $ECT[entry.GT].sizeQ.push(entry.size)$
      if $ECT[entry.GT].NEQ.size() > k$ **then**
        $ECT[entry.GT].NEQ.pop()$
        $ECT[entry.GT].sizeQ.pop()$
      end if
    end if
  end for
end for

same GT. This means that contents of the same GT are less likely to be requested consecutively. Because the distance between two vectors is inversely proportional to the probability of the two contents being requested consecutively, reducing the probability is equivalent to maximizing the distance between the same-GT vectors. The process of the FCA can be described by Algorithm 1, where NGT is the Number of GT.

2) Guide Requests by GT: With the assistance of the Edge Candidate Table (ECT), when a request misses at the home edge, the user can receive a suggestion of edges to try next based on the request's GT. Among numerous neighbor edges, they are considered to have the highest potential for fulfilling the user's request.

This process leads to an interesting phenomenon. Initially, when requests for a particular GT frequently appear in an area, the edge accumulates more contents of this GT compared to other GTs. Consequently, a dominant GT (DGT) gradually emerges within this edge, occupying a relatively large portion of the total cache capacity. The edge with this DGT becomes the primary consideration when recommending requests for the corresponding GT at the neighbor edges. In return, more requests of this GT are directed to the edge, further increasing the cached size of this DGT and establishing the edge's unique advantage. Eventually, each edge accumulates contents that are specific to its DGT and attracts requests of that GT like a magnet. This virtuous cycle between user requests and the size of DGT leads to a high hit ratio.

3) ECT Update Algorithm: In addition to establishing competitive hit advantages, it is also essential to spreading these advantages among neighbor edges. This enables each edge to perceive the dominant GTs of its neighbors. The ECT plays a necessary role in disseminating these hit advantages and is kept updated by exchanging CSTs with neighbor edges. It's crucial to choose the appropriate cooperative scope, which is controlled by the Number of Exchanging Neighbors (NEN). A scope that is too narrow can lead to a low hit ratio, as demonstrated in Section II-A, while an excessively broad scope may result in heavy traffic between edges. The impact of NEN on the performance of MagNet is discussed in Section V-C.

At each ECT Updating Period (EUP), an edge fetches its neighbors' CSTs and updates its own ECT according to Algorithm 2. During this update process, for each GT, the Top-k NEs that have the greatest cached GT size are selected. By utilizing the GT and a well-maintained ECT, requests can be efficiently directed toward the edges that have great potential to yield a hit.

4) Quick Cache: To further accelerate the formation of DGT and unleash the virtuous circle, we propose the incorporation of the Quick Cache within the ACC mechanism. The Quick Cache comprises two essential components (see Fig. 10): Hot Content Prefetch (HCP) and Access Rule (AR).

Hot Content Prefetch (HCP): Instead of merely caching the requested content passively, HCP proactively caches content that is likely to be popular. As illustrated in Algorithm 3, taking GT-A as an example, each edge examines the cached size of its largest GT periodically. If GT-A occupies more than a certain proportion (e.g., 10%) of the edge's total cache capacity, it is deemed as having the potential to become the DGT. This also indicates that the content associated with GT-A is likely trending or in high demand within the edge's coverage area. At this point, the HCP is executed in two phases on the edge. First, HCP actively purges the cache of less popular content. Specifically, the edge actively removes the cached content of several GTs with the smallest proportion, each occupying less than 3% of the total cache capacity. As analyzed later in Fig. 9, the majority of requests for such content are unlikely to recur. This implies that caching these contents does not contribute to the overall hit rate but instead occupies valuable cache space. Subsequently, with the freed cache space, the edge fetches and locally caches the content of GT-A from the cloud, populating the cache with trending content in place of the evicted unpopular content. Within the MagNet framework, HCP identifies trending content according to the cache size of GTs and then expands the size of the biggest GT. In this way, HCP ensures the relevance of the prefetched content (within the same GT), meanwhile accelerating the formation of the DGT. By leveraging the strengths of the ACC mechanism,

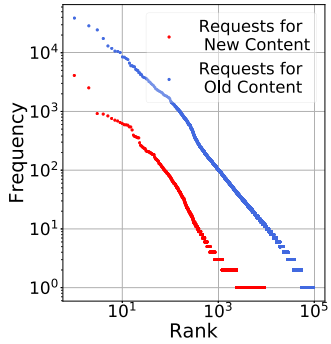


Fig. 9. Long-tail distribution of requests in Dataset1.

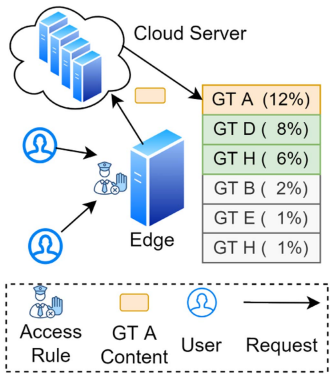


Fig. 10. HCP and AR in the Quick Cache.

it further attracts similar user requests, thereby boosting the hit ratio.

Access Rule (AR): As depicted in Fig. 9, we first conduct an analysis of the occurrence frequency of user requests in Dataset1. The x-axis represents the logarithmic rank of requests and the y-axis shows the logarithmic frequency of these requests. This log-log diagram indicates a distinct long-tail distribution of the request occurrence frequency. In this diagram, "new content" refers to content that has never been requested in the past week, possibly fresh content emerging on the Internet. Conversely, "old content" pertains to content that has been previously requested, likely denoting content that has been available online for some time and has garnered a certain degree of popularity. The analytical findings from Fig. 9 reveal that 61% of requests for new content only occur once, while 13% of requests for old content will not appear for the second time. This can be explained by the fact that users might casually browse fresh content, while classic content is more likely to be revisited multiple times.

However, conventional cache replacement strategies, such as LRU and LFU, would overlook these differences. As a result, these strategies might inadvertently allocate precious cache space to less popular content, consequently diluting the cached size advantage of the DGT. To solve this issue, we formulate the Access Rule (AR) to selectively choose which content to locally cache beyond cache replacement strategies. To ensure that AR will not disrupt the edge's caching function, its execution is bifurcated based on the progression of DGT formation. Initially, when DGT has yet not formed in the edge, AR specifies that the edge caches new content only after its request appears twice on the same day. However, once DGT is formed in the edge, to safeguard the DGT cache from being compromised, AR specifies that the edge does not cache any content upon its first appearance. Thus, HCP effectively precludes content that appears only once from occupying limited cache space, thereby fostering the formation of the scale of DGT.

Algorithm 3: Hot Content Prefetch Algorithm.

Input: $edge, ClearNumber$
while Every two minutes **do**
  $GT\_A \leftarrow edge.GetLargestGT()$
  if $GT\_A.size > 0.1 * edge.Capacity$ **then**
    $leastGTs \leftarrow edge.GetLeastGTs(ClearNumber)$
    for each GT in $leastGTs$ **do**
      if $GT.size < 0.03 * edge.Capacity$ **then**
        $edge.EvictCache(GT)$
      end if
    end for
  end if
  $SpareSize \leftarrow edge.Capacity - edge.CacheSize$
  $edge.FetchfromCloud(GT\_A, SpareSize)$
end while

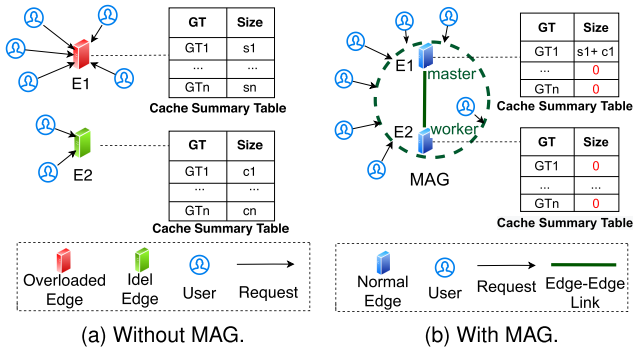


Fig. 11. Mutual assistance group (MAG).

B. Mutual Assistance Group (MAG)

Balanced workloads facilitate a stable and robust system. However, user requests are unevenly distributed both timely and spatially as previously analyzed in Section II-B. This causes different workloads on different edges, which may eventually lead to some of them being overloaded while others may be idle, as illustrated in Fig. 11(a). This situation causes a waste of resources and degraded QoE. In order to address this problem, MagNet designs the Mutual Assistance Group (MAG). MAG consists of two stages to allocate appropriate workloads to different edges according to their capacities.

In Stage 1, each OE tries to find some IEs for assistance in its neighbor edges and form a temporary MAG according to Algorithm 4. A MAG runs in a master-worker mode where the original OE is the master that records all the caching information of all the workers. Workers in the MAG help to handle requests together according to their different capacities. IEs are added to

Algorithm 4: MAG Running Procedure.

Input: $edge$, $workload$
while Every two minutes **do**
  $edge.status \leftarrow$ UpdateStatus($edge, workload$)
  if Stage $==$ 1 **then**
    if $edge.status ==$ OE **then**
      while $edge.status \neq$ NE **and** ExistNeighbor(IE) **do**
        $ie \leftarrow$ FindIE() in edge.neighbors
        JoinMAG(ie)
        UpdateMAG()
      end while
      if $edge.status ==$ OE **then**
        $Stage \leftarrow 2$
      end if
    else if $edge.status ==$ IE **then**
      while $edge.status \neq$ NE **and** $edge.workers \neq null$ **do**
        ReleaseWorker()
        UpdateMAG()
      end while
    end if
  else if Stage $==$ 2 **then**
    $GT, size \leftarrow$ MAG.GetCachedGTSize()
    $VisibleSize \leftarrow$ UpdateVisibleSize($size, edge$)
    UpdateCST($GT, VisibleSize$)
  end if
end while

Algorithm 5: Visible Size Updating Algorithm.

Input: $RealSize$, $status$, $discount$
Output: $VisibleSize$
Initialize $discount$ as 1
if $status$ is OE **then**
  $size \leftarrow$ GetMidValue($0, RealSize * discount$)
else
  $size \leftarrow$ GetMidValue($RealSize * discount, RealSize$)
end if
$discount \leftarrow size/RealSize$
if $discount \geq 0.9$ **then**
  $VisibleSize \leftarrow RealSize$
  $Stage \leftarrow 1$
else
  $VisibleSize \leftarrow size$
end if
return $VisibleSize$

a MAG one by one until the original OE of the MAG becomes an NE, or there is no IE left around its neighboring scope. The MAG appears as a whole to the outside, as if all contents are cached in the master edge while the workers have none. As illustrated in Fig. 11(b), the master's Cache Summary Table (CST) shows it contains all the contents of the MAG, while the worker's CST shows it contains nothing. Therefore, the neighbors and users do not even know it is a MAG, which helps maintain the system's consistency. As a result, the master attracts all feasible requests and then distributes them to the workers. To avoid a MAG occupying too much caching capacity to become a "CDN", each MAG is restricted to cache one GT content. The workers in a MAG will be released once the master becomes an IE.

Considering latency and bandwidth factors, during Stage 1, the formation of MAG is restricted to seeking IEs within the neighboring scope of the OE as workers. However, considering dynamic request distribution, there might be situations where no IEs are available to assist within the neighboring area of an OE. This leaves the overloaded situation unresolved, which leads to the initiation of Stage 2 of the MAG mechanism.

In Stage 2, MAG's objective is to alleviate overload pressure by reducing the number of requests it attracts. This challenge stems from the ACC mechanism within MagNet, which guides user requests to optimal edges based on GT size. As a result, edges with larger caches for a particular GT naturally tend to draw more requests. To mitigate this, MAG strategically decreases the visible cached size of the GT within the group. This adjustment in cached visibility results in fewer incoming requests, thereby easing the workload. Specifically, the master edge determines the visible cached size to be showcased to other edges during CST exchanges according to Algorithm 5. Drawing inspiration from the binary search concept [28], if the current master edge remains overloaded, the visible cached size is consistently set at half of the actual cached size. If the master edge is no longer overloaded, the visible cached size is re-calibrated to the median of the previous round's visible size and the current actual size. Once the visible cached size recovers to more than 90% of the actual cached size, it is believed that the overloaded situation has been substantially alleviated. MAG then exits Stage 2 and resumes the process in Stage 1.

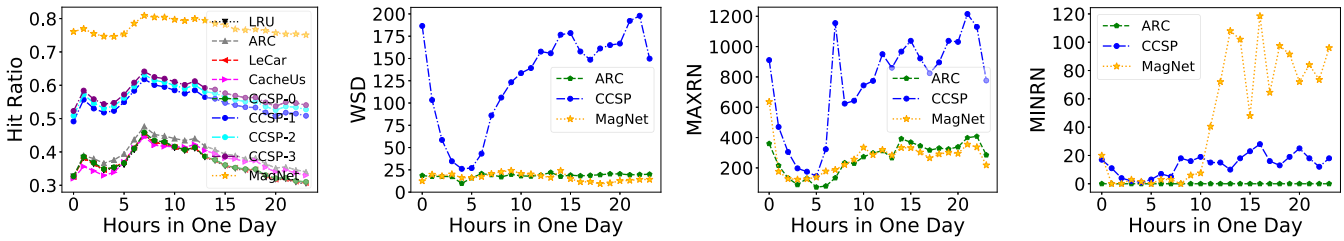
V. Evaluation

In this section, we first conduct experiments on the real dataset to evaluate the performance of MagNet. The results show MagNet's superiority in terms of the hit ratio and workload balance over the classical, learning-based, and cooperative edge caching solutions. Then we conduct experiments to analyze several components in MagNet.

A. Methodology

Dataset. Throughout this section, we use Dataset1 described in Table I as the data set of the experiments. For Dataset1, each request contains a user ID, timestamp, latitude, longitude, and content ID, which identify when and where the user requested which specific content. We divide Dataset1 into two parts: 1) data from the first 12 days, which is used for training and analyzing purposes, and 2) data from the last day, which is used for testing and evaluation purposes.

Testbed: Since the spatial area of Dataset1, i.e., the whole Beijing city, is too big for experiment purposes, we choose



(a) Hit ratio performance. (b) Workloads' standard deviation. (c) Max requests' number. (d) Min requests' number.

Fig. 12. Performance in one whole day.

a 20 KM $\times$ 20 KM target area in Beijing with 400 edges deployed. Since files can be divided into units of the same size in caching environment, we assume all contents have the same unit size. Each edge is set to have the same caching capacity to cache 135 files by default. NEN is set to be a default value of 154, which is the number of edges in a 7KM-radius circle. We set EUP as 6 minutes by default to maintain the balance between the timeliness and traffic overhead. We set the Number of GT (NGT) as 621. We use Java [29] to implement MagNet with object-oriented programming [30]. It runs on a computer with one GeForce GTX 1660 TI GPU, one Intel i7-9700 CPU and 16 GB memory. Additionally, we build a multi-edge caching experiment platform. It accepts a prepared dataset to simulate parallel sending requests to edges. It can measure edges' hit ratios and workloads. In the experiments, we use Dataset1 and set 400 edges in the platform.

Baseline Solutions: To evaluate the MagNet performance, classical, ML-based and cooperative solutions are used for comparison.

- *ARC [31]:* It is a self-tuning and low-overhead caching solution. It combines the advantages of LRU and LFU, which enables it to be flexibly adjusted in different scenarios.
- *LeCaR [32]:* It is a caching framework that uses reinforcement learning and regrets minimization to adjust the usage of LRU and LFU dynamically.
- *CACHEUS [33]:* It is an adaptive caching algorithm which uses online reinforcement learning to take advantage of some state-of-the-art caching algorithms, including ARC, SR-LRU, CR-LRU [31] and LIRS [34].
- *CCSP [35]:* It is a cooperative caching scheme. It chooses some central nodes which retrieve caching summaries from their neighbors and provide caching information for other normal nodes. Each request in it can try four edges at most.

Evaluation Metrics.

- Hit ratio: is calculated by dividing the total number of hits by the total number of requests.
- Workload SD: WSD is calculated according to (8) to observe the workload balance of the system.
- Max Requests' Number of A Single Edge (MAXRN): the maximum served requests' number of one edge in the 400 edges.
- Min Requests' Number of A Single Edge (MINRN): the minimum served requests' number of one edge in the 400 edges.

Since the same edge in different non-cooperative solutions receives the same amount of requests, we use the ARC solution as a representative in WSD, MAXRN, as well as MINRN.

B. Overall Performance

We conduct experiments from three different perspectives to evaluate the overall performance of MagNet.

Performance in One Whole Day: In this case, we use the default standard setups. In Fig. 12(a), we find MagNet has a significantly higher hit ratio than the others, exceeding 75% in the most time of one day. In Fig. 12(a), CCSP-0 represents CCSP where each request can try only its home edge; CCSP-1, CCSP-2 and CCSP-3 represent CCSPs where each request can try at most one, two and three more edges, respectively, except its home edge. Compared with CCSP, MagNet enjoys about a 20% higher hit ratio than the CCSP-3. This outstanding hit ratio performance is because the ACC helps guide requests to their optimal edges. Fig. 12(b) shows the WSD of each hour, where MagNet has lower WSD than others in most hours, especially in the request-intensive hours (13:00 to 24:00, as shown in Fig. 4), meaning it has the most balanced workload. Fig. 12(c) and (d) indicate that no edge is under excessive pressure and MagNet utilizes edges efficiently. This case proves that MagNet has a higher hit ratio and a more balanced workload than others in most time of one day.

Different Edge Densities: Next, we investigate the impact of different edge densities on the performance of MagNet. In this $20KM \times 20KM$ target area, we set the number of edges from 4 to 900. As indicated in Fig. 13(a), the hit ratio decreases as the edge number increases for all caching solutions. Because as the density increases, requests are served by more edges and then the hit ratio decreases, which is consistent with the result from the data analyses in Section II-A. The hit ratios of all non-cooperative solutions decrease drastically by about 35%. The CCSP solution has a better hit ratio performance than non-cooperative solutions, but its hit ratio still decreases significantly by 25%. In contrast to other solutions, MagNet achieves an excellent hit ratio result, which only decreases less than 5%. Fig. 13(b), (c) and (d) show that the workload balance of all solutions is better as the edge density increases, while the edges' workloads are reduced. Moreover, MagNet has the best workload balance performance than the others from all three metrics. This case proves that MagNet has an outstanding hit

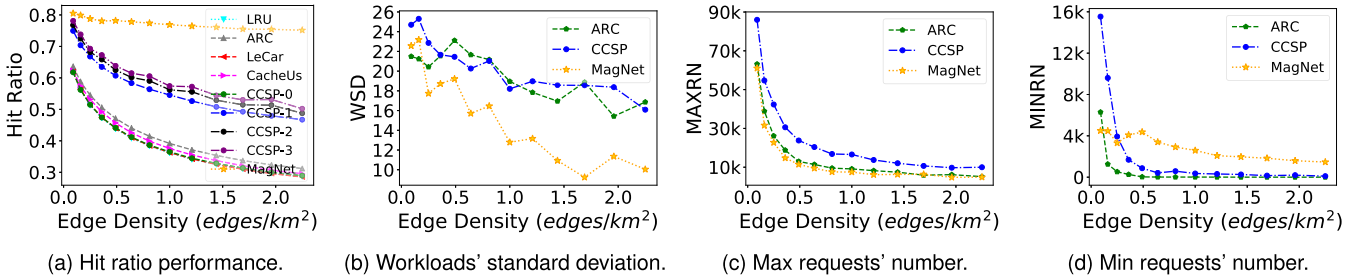


Fig. 13. Performance under different edge densities.

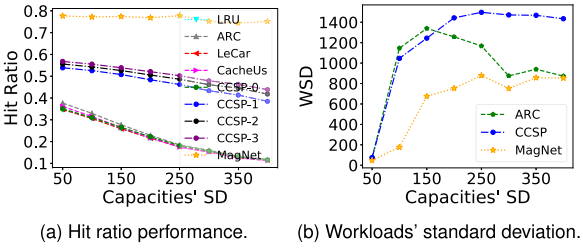


Fig. 14. Different capacities' standard deviation.

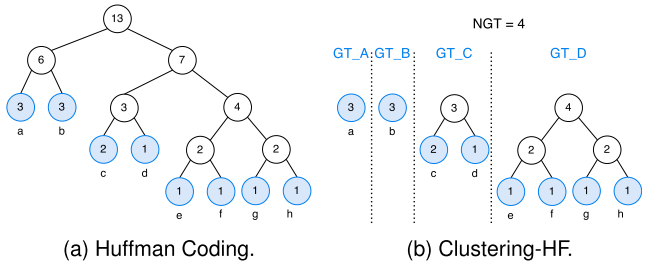


Fig. 15. Content clustering inspired by Huffman coding.

ratio performance and prominent workload balance in various edge-density areas.

Different Capacities' Standard Deviation: In this case, we set the 400 edges with different caching capacities. In real-world scenarios, the capacities of different devices are generally different, which challenges the adaptability and flexibility of the solution. We use the capacities' standard deviation (SD) to control the capacities' difference degree. In Fig. 14(a), with the increase of capacities' SD, the MagNet's hit ratio steadily remains at a high level, while the hit ratios of the other solutions show a significant downward trend. Fig. 14(b) reveals that MagNet achieves a more balanced workload. In this case, the WSD values are quite prominent because the WSD considers the differences of capacities in (8). This case proves that MagNet is more flexible and adaptable under the environment of heterogeneous edges' capacities.

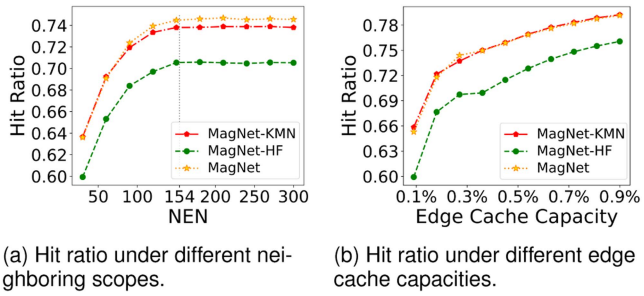
Quantitative Analyses of System Overhead: Finally, we analyze MagNet's overhead and compare it with other solutions. As a cooperative edge caching solution, MagNet requires periodic information exchange between edges, generating measurable overhead. In our experiments, each edge communicates with 154 NEs to exchange CSTs every six minutes. The CST, recording types and sizes of GTs cached at each edge (up to 135 types), incurs a maximum overhead of approximately 3.69 kbps per edge. This is significantly lower than other cooperative caching solutions. For instance, in DIMA [36], each edge's DRL agent network parameters are exchanged among all edges in each time slot. Considering the network architecture, transmitting one network parameter per edge is estimated to require at least 257kbps, based on a hidden layer with 128 neurons. In another example, [37] reports that each edge must exchange a tuple of observation, action, and reward space data for DRL agents in each time slot. This includes extensive data volumes such as requested multimedia content, delivery deadline, and content priority. The overhead for this exchange far exceeds that of MagNet. Additionally, MagNet's information exchange is confined to a limited neighboring area, thus not burdening the backbone network. Given the significant performance benefits due to edge cooperation, MagNet's overhead is deemed acceptable.

C. Components Analysis Study

1) Comparative Analysis of Clustering Algorithm: In MagNet, we design the FCA to do content clustering. The clustering is the crucial step of the ACC. To find a better way, we also design two other clustering algorithms: The Huffman-based algorithm and the KMeans-based clustering algorithm.

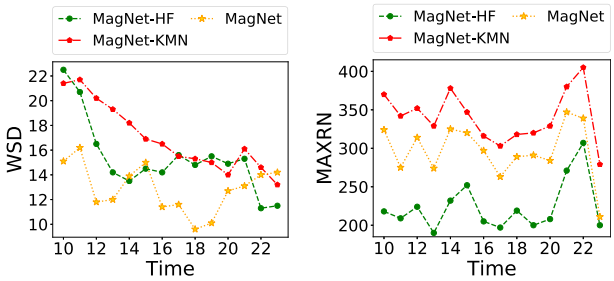
MagNet-HF: The MagNet-HF stands for the MagNet Framework with the Huffman-Based clustering algorithm, which is inspired by the Huffman coding. Huffman coding assigns shorter codes for more frequent words and longer codes for less frequent words. In MagNet-HF, we give more resources and attention to the more frequent contents and vice versa (as indicated in Fig. 15). The clustering process is treated as the combining process of the Huffman coding, which stops until the number of combined trees equals the NGT. Then, the contents in one tree are divided into one GT, making the GTs have different numbers of contents. This algorithm can protect the popular contents from being replaced due to intra-GT competition. The difference between the Huffman-based clustering algorithm and the FCA is that the former enforces each GT to have the same frequency, while the latter focuses more on the sequence of requests and tries to avoid the sequential requests requesting contents in the same GT.

MagNet-KMN: The MagNet-KMN stands for the MagNet Framework with the K-means based clustering algorithm. In Section IV-A1, the FCA is designed to disperse the contents of consequential requests. From another perspective, clustering



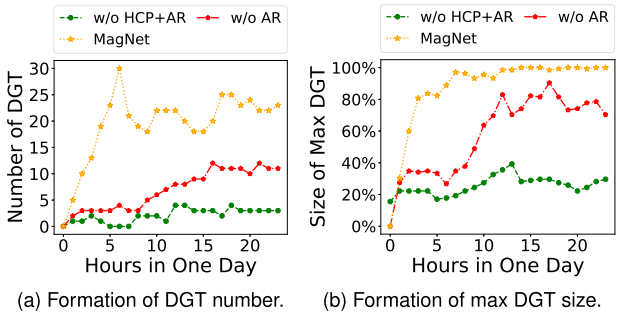
(a) Hit ratio under different neighboring scopes. (b) Hit ratio under different edge cache capacities.

Fig. 16. Hit ratio performance.



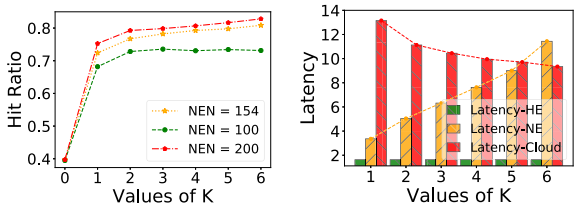
(a) Workloads' standard deviation in high pressure time. (b) Max request number of one edge in high pressure time.

Fig. 17. Workload compare in high pressure time.



(a) Formation of DGT number. (b) Formation of max DGT size.

Fig. 18. DGT formation in one whole day.



(a) Hit ratio performance. (b) Latency performance.

Fig. 19. Performance under different k values.

the contents of consequential requests into one GT can make consequential requests to be guided to the same edge. For this reason, the K-means algorithm [38] is used to cluster closer vectors into one GT. We use the KMeans approach from sklearn python package [39] to do the clustering.

Hit Ratio Comparison: First, to observe the impact of NEN, we compare the MagNet-KMN, the MagNet-HF and the default MagNet on the hit ratio. In Fig. 16(a), it can be found that as NEN increases, the hit ratio increases significantly at the beginning and then stays at a stable level. For different NEN, the neighboring scope of an edge is different. More importantly, when NEN is large enough, same-GT requests within a 7 KM radius can be aggregated into one edge, which is the necessary condition for a high hit ratio, as revealed in Section II-A.

The stationary point of NEN happens at around 154, which is also consistent with NEN calculated based on a 7 KM radius circle and $1\ \mathrm{edge}/\mathrm{KM}^2$ density. Second, we analyze three MagNet framework based approaches on the different caching capacities. When caching capacity increases, three approaches' hit ratios all go up. The hit ratio difference between the MagNet-KMN and MagNet is not significant, while the MagNet-HF has about 5% lower hit ratio than them. This is because the MagNet-HF only focuses on the balance of GTs' requests.

Workload Comparison: Fig. 17 shows the WSD and the Max Requests' Number of A Single Edge (MAXRN) in high-pressure time, from 10:00 to 24:00, for the three algorithms. It shows that the MagNet-KMN has a higher WSD and MAXRN than the other two, which means it performs the worst in terms of workload balance. Because the MagNet-KMN clusters consequential contents in one GT, which can enhance the locality effects and improve the hit ratio. However, the popular contents in one GT may attract excessive requests, which makes the workload unbalanced. The MagNet-HF has a lower MAXRN than the other two because it balances the requests' number of each GT. On the whole, MagNet has excellent performance on both the hit ratio and the workload balance as it disperses the consequential requests to avoid the intra-GT competition.

2) Ablative Analysis of Quick Cache: To accelerate the formation of the Dominant Guidance Type (DGT), we introduce the Quick Cache within the ACC framework. Here we conduct experimental ablation to validate the contribution of the Quick Cache on DGT formation. As shown in Fig. 18, we compare MagNet without Hot Contents Prefetch (HCP), MagNet without both HCP and Access Rule (AR), and the complete MagNet system in terms of DGT formation. We assess the formation of DGT from two perspectives: the number of DGT formed in the system and the cache proportion of the DGT with the biggest cache size. Here DGT is defined as GT that occupies more than 20% of the edge's cache capacity. As demonstrated in Fig. 18(b), the Quick Cache plays an essential role in rapidly increasing the number of DGT and maintaining them at a consistently high level. Besides, Fig. 18(b) indicates the substantial effect of the Quick Cache on expanding the scale of DGT, with the largest DGT almost occupying the entire cache space. In summary, the results from the above ablation experiments provide compelling evidence that the Quick Cache effectively accelerates DGT formation in MagNet.

3) Parameter Variation Analysis of Top-K NEs: As illustrated in Section III-B, in MagNet, if users' requests don't hit in home edges, users are recommended to route their requests to the Top-k Neighbor Edges (NEs). To further investigate the optimal k value that can achieve the best system overall performance, here we conduct experiments to compare both hit ratio and latency performance under different k parameter values.

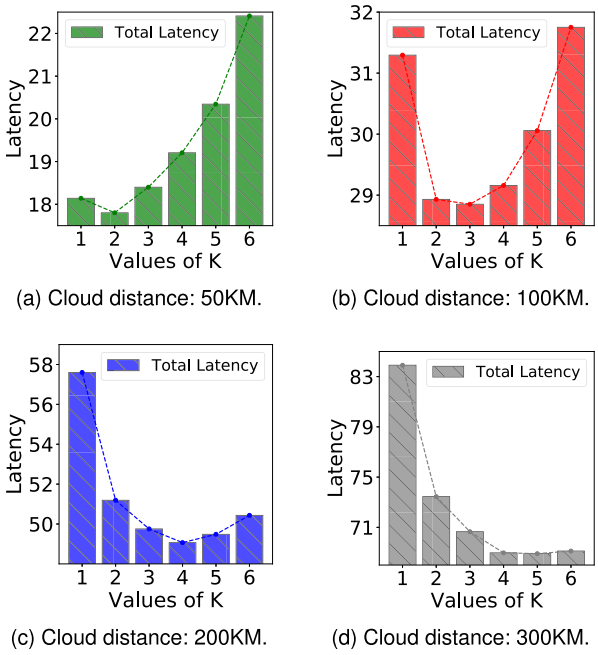


Fig. 20. Latency performance under different k values.

Hit Ratio Comparison: As shown in Fig. 19(a), we first analyze the impact of different k values on the hit ratio. When the value of K is 0, the hit ratio is significantly low, around 40%. And there is a substantial gap in the hit ratio between the k value being 0 and 1. When the k value exceeds 1, the hit ratio gradually increases with k increasing, and the rate of increase tends to plateau. In addition to setting the value of NEN at 154, we also investigate the impact of k values on hit ratio when NEN is set to 100 and 200 respectively.

Latency Comparison: Next, we analyze the impact of different k values on the latency. As mentioned in Section III-A, the total latency consists of three parts: latency from the user to the home edge (Latency-HE), latency during the user's traversal of Top-k NEs (Latency-NE), and latency to fetch content from the cloud (Latency-Cloud). The latency between any two nodes is directly proportional to the distance between them. To better analyze the influence of k values on the latency, we compare the variations in latency from different parts under different k values. As shown in Fig. 19(b), as the k value increases, Latency-HE remains unaffected, while Latency-NE continuously increases, due to traversing more neighboring edges. However, Latency-Cloud (assuming 100 km distance between users and the cloud) gradually decreases as the k value increases. This phenomenon occurs because the improvement in the hit ratio enables users' requests to be fulfilled from nearby edge nodes, rather than retrieving content from distant clouds.

Considering that the two components of total latency exhibit opposite trends as the k value changes, we further investigate the impact of the k value on latency performance under different distances between the cloud and users (see Fig. 20). We observe that as the cloud distance increases, the optimal k value (considering only from the latency perspective) also gradually increases. For example, when the cloud distance is 100 KM, the total latency is lowest under Top-3 NEs, whereas when the cloud distance is 300 KM, the total latency is lowest under Top-5 NEs. A greater distance to the cloud implies a higher time cost of fetching the content from the cloud if the edges fail to satisfy the requests. Thus, improving the hit ratio will lead to a larger compensatory latency performance. Therefore, when dealing with scenarios with long distances to the cloud, traversing more edges and improving the hit ratio would be a better choice.

VI. Related Work

A. Hit Ratio Improving

In recent years, some caching solutions have been proposed to improve the hit ratio. The Least Recently Used (LRU) [22] and the Least Frequently Used (LFU) [24] are widely used in the industry. Considering the temporal locality of requests, the LRU selects the least recently used items for eviction. While the LFU considers items with the least frequency as the first candidate for eviction. In addition, there are advanced algorithms such as Adaptive Replacement Cache (ARC) [31] and CLOCK with Adaptive Replacement (CAR) [40]. These algorithms aim to combine the advantages of both LRU and LFU. The ARC employs a self-tuning algorithm that learns and adapts to the workload, while the CAR is based on the clock replacement policy, providing a simpler yet equally effective alternative to ARC. However, these above classical caching solutions are based on assumptions regarding static request patterns, such as the frequency and recency. These solutions are only expected to be optimal under limited circumstances and are difficult to adapt to highly dynamic requests.

With the rapid development of Machine Learning (ML), some ML-based caching solutions have been proposed to address the dynamic challenge. The LeCaR [32] utilizes reinforcement online learning with regret minimization to learn the patterns in the workload and adapt the cache replacement strategy accordingly. The CacheUs [33] introduces four workload primitive types. Inspired by LeCaR, it also uses ML to identify which eviction strategy is currently suitable based on these workload primitive types. The Cocktail [41] utilizes an ensemble method to deal with the highly dynamic workloads and caching configurations. A Deep Reinforcement Learning (DRL) agent is trained to adaptively select the best constituent caching policy based on their merits. The CACA [42] combines the video category and the author's information of a video to predict the video's popularity, but it is restricted in the environment with video category and author attributes. In [43], a dynamic online learning method is proposed to solve the non-stationary caching problem. Their method is proved to achieve nearly optimal by a proposed online optimization framework. These solutions leverage the characteristics of requests to select appropriate cache admission and cache replacement strategies, thereby improving the hit ratio of individual nodes. However, compared with the CDN, the edge has limited caching and computational capacities, which brings new challenges to improve the performance of the edge caching system.

Some cooperative caching solutions have been proposed to promote the hit ratio by edges' cooperation. The CCSP [35]

elects representative edges for small areas. These representative edges summarize the cache information of at most four neighbor edges in the area. Hence, The CCSP allows requests to reach all cached contents in the neighboring area with the help of the representative edge. The DIMA [44] models the cooperative caching problem as a Markov decision process (MDP) and proposes a DRL solution. Caching and replacement decisions in edges are made through a distributed double dueling deep Q-network (D3QN). The MacoCache [45] proposes a multi-agent deep reinforcement learning (MADRL) based solution. Independent Q-learning (IQL) is used in each agent to learn its own best policy and other agents are considered as part of the environment. Meanwhile, the Long Short Term Memory network (LSTM) is integrated for time series dynamics and diversities. In [37], the cooperative cache replacement problem is modeled as a Partially Observable Markov Decision Process (POMDP) and solved also by MADRL. A Multi-Agent Recurrent Deep Deterministic Policy Gradient (MARDDPG) algorithm is proposed to decide whether to evict the requested content based on global states. Nevertheless, these solutions cost enormous computational capacities in edges which have limited computational capacities.

B. Heterogeneous Capacities Utilizing

Some cooperative edge caching solutions have also been proposed to utilize heterogeneous capacities. As discussed in Section II, edges have heterogeneous caching and computational capacities. Therefore, researchers have conducted studies on how to leverage these heterogeneous capacities. There are two types of solutions to organize cooperative edge caching systems: 1) centralized cooperative solutions [35], [46], [47], [48], which use some central edges to record and manage the information. In [46], for example, edges in the system are divided into small clusters. A central storage device is selected for each cluster and keeps its own cache data as well as a synchronized database of all other nodes in this given cluster. These solutions make the central edge a key and high-pressure part of the system, resulting in poor robustness. Additionally, synchronizing all edges' cache information with the central edge becomes intractable as cache replacements happen frequently. 2) decentralized cooperative solutions [45], [49], which manage edges in a distributed way. In [49], the cooperative caching problem is modeled as a multi-agent multiarmed bandit problem and Q-learning is used to learn how to coordinate the caching decisions between different edges. In [45], agents in different edges can also be joint action learners and learn caching decisions in conjunction with heterogeneous edges. These solutions forward requests to many of their neighbors indiscriminately, which linearly increases edges' burden with the increased number of forwarding.

VII. Conclusion

In this paper, we analyze the challenges of edge caching solutions and propose a decentralized and cooperative edge caching system named MagNet. MagNet has two innovative key mechanisms. The ACC mechanism uses the neural embedding technology and novel clustering algorithm to cluster contents and then guides requests to their optimal edges to enhance the hit ratio. The MAG mechanism combines the overloaded edges and the idle edges into temporary groups to enhance the workload balance. Experiments are conducted to analyze the contributions of these components in MagNet. To evaluate the overall performance of MagNet, we compare it with the classical, ML-based and cooperative caching solutions from three different perspectives using a real dataset. The results prove that MagNet significantly outperforms other solutions in terms of both hit ratio and workload balance.

Acknowledgment

Preliminary results in this paper are presented at the ACM Web Conference, 2022 [18].

References

[1] Sandvine, "The global internet phenomena report. 2023," 2023. [Online]. Available: https://www.sandvine.com
[2] Cisco, "Cisco annual internet report (2018–2023) white paper," 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11--741490.html
[3] L. Van Ma, V. Q. Nguyen, J. Park, and J. Kim, "NFV-based mobile edge computing for lowering latency of 4K video streaming," in *Proc. 10th Int. Conf. Ubiquitous Future Netw.*, 2018, pp. 670–673.
[4] A. Tang and O. Fakourfar, "Watching 360 videos together," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2017, pp. 4501–4506.
[5] A. Lécuyer, F. Lotte, R. B. Reilly, R. Leeb, M. Hirose, and M. Slater, "Brain-computer interfaces, virtual reality, and videogames," *Computer*, vol. 41, no. 10, pp. 66–72, 2008.
[6] J. Choi, A. S. Reaz, and B. Mukherjee, "A survey of user behavior in VoD service and bandwidth-saving multicast streaming schemes," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 1, pp. 156–169, First Quarter, 2012.
[7] A.-T. Tran et al., "Hit ratio and latency optimization for caching systems: A survey," in *Proc. Int. Conf. Inf. Netw.*, 2021, pp. 577–581.
[8] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," *ACM Comput. Surv.*, vol. 35, no. 4, pp. 374–398, 2003.
[9] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, Fourth Quarter 2017.
[10] A.-J. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante, "Drafting behind Akamai: Inferring network conditions based on CDN redirections," *IEEE/ACM Trans. Netw.*, vol. 17, no. 6, pp. 1752–1765, Dec. 2009.
[11] C.-F. Lin, M.-C. Leu, C.-W. Chang, and S.-M. Yuan, "The study and methods for cloud based CDN," in *Proc. Int. Conf. Cyber-Enabled Distrib. Comput. Knowl. Discov.*, 2011, pp. 469–475.
[12] Google, "Google cloud CDN locations," 2021. [Online]. Available: https://cloud.google.com/cdn/docs/locations
[13] Akamai, "Media delivery network map," 2021. [Online]. Available: https://www.akamai.com/visualizations/media-delivery-network-map
[14] X. Fan, E. Katz-Bassett, and J. Heidemann, "Assessing affinity between users and CDN sites," in *Proc. Int. Workshop Traffic Monit. Anal.*, Springer, 2015, pp. 95–110.
[15] P. Casas, A. D'Alconzo, P. Fiadino, A. Bär, and A. Finamore, "On the analysis of QoE-based performance degradation in YouTube traffic," in *Proc. 10th Int. Conf. Netw. Serv. Manage. Workshop*, 2014, pp. 1–9.
[16] Q.-V. Pham et al., "A survey of multi-access edge computing in 5G and beyond: Fundamentals, technology integration, and state-of-the-art," *IEEE Access*, vol. 8, pp. 116 974–117 017, 2020.
[17] K. Kumaran and E. Sasikala, "Learning based latency minimization techniques in mobile edge computing (MEC) systems: A comprehensive survey," in *Proc. Int. Conf. Syst. Computation Automat. Netw.*, 2021, pp. 1–6.
[18] J. Peng et al., "MagNet: Cooperative edge caching by automatic content congregating," in *Proc. ACM Web Conf.*, 2022, pp. 3280–3288.
[19] C. Guo, Y. Liu, W. Shen, H. J. Wang, Q. Yu, and Y. Zhang, "Mining the web and the internet for accurate IP address geolocations," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2009, pp. 2841–2845.
[20] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[21] M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras, and S. Chouvardas, "Placing dynamic content in caches with small population," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
[22] H. Johnson and J. Larson, "Data management for microcomputers," in *Proc. Compcon Fall*, 1979, pp. 191–192.
[23] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "FemtoCaching: Wireless content delivery through distributed caching helpers," *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 8402–8413, Dec. 2013.
[24] D. Lee et al., "On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 1999, pp. 134–143.
[25] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowl.-Based Syst.*, vol. 151, pp. 78–94, 2018.
[26] O. Barkan and N. Koenigstein, "Item2Vec: Neural item embedding for collaborative filtering," in *Proc. IEEE 26th Int. Workshop Mach. Learn. Signal Process.*, 2016, pp. 1–6.
[27] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Adv. Artif. Intell.*, vol. 2009, 2009, Art. no. 4.
[28] A. Lin, "Binary search algorithm," *WikiJ. Sci.*, vol. 2, no. 1, pp. 1–13, 2019.
[29] J. Gosling, "The feel of Java," *Computer*, vol. 30, no. 6, pp. 53–57, 1997.
[30] N. Kobayashi and A. Yonezawa, "Type-theoretic foundations for concurrent object-oriented programing," *ACM SIGPLAN Notices*, vol. 29, no. 10, pp. 31–45, 1994.
[31] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," in *Proc. 2nd USENIX Conf. File Storage Technol.*, 2003, pp. 115–130.
[32] G. Vietri et al., "Driving cache replacement with ML-based LeCaR," in *Proc. 10th USENIX Workshop Hot Top. Storage File Syst.*, 2018, Art. no. 3.
[33] L. V. Rodriguez et al., "Learning cache replacement with CACHEUS," in *Proc. 19th USENIX Conf. File Storage Technol.*, 2021, pp. 341–354.
[34] S. Jiang and X. Zhang, "LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 1, pp. 31–42, 2002.
[35] A. Larbi, L. Bouallouche-Medjkoune, and D. Aissani, "Improving cache effectiveness based on cooperative cache management in MANETs," *Wireless Pers. Commun.*, vol. 98, no. 3, pp. 2497–2519, 2018.
[36] H. Tian et al., "DIMA: Distributed cooperative microservice caching for Internet of Things in edge computing by deep reinforcement learning," *World Wide Web*, vol. 25, no. 5, pp. 1769–1792, 2022.
[37] M. K. Somesula, R. R. Rout, and D. V. Somayajulu, "Cooperative cache update using multi-agent recurrent deep reinforcement learning for mobile edge networks," *Comput. Netw.*, vol. 209, 2022, Art. no. 108876.
[38] K. Krishna and M. N. Murty, "Genetic K-means algorithm," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 29, no. 3, pp. 433–439, Jun. 1999.
[39] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
[40] S. Bansal and D. S. Modha, "CAR: Clock with adaptive replacement," in *Proc. 3rd USENIX Conf. File Storage Technol.*, 2004, pp. 187–200.
[41] T. Zong, C. Li, Y. Lei, G. Li, H. Cao, and Y. Liu, "Cocktail edge caching: Ride dynamic trends of content popularity with ensemble learning," in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.
[42] Y. Guan, X. Zhang, and Z. Guo, "CACA: Learning-based content-aware cache admission for video content in edge caching," in *Proc. 27th ACM Int. Conf. Multimedia*, 2019, pp. 456–464.
[43] S. Zhou et al., "Caching in dynamic environments: A near-optimal online learning approach," *IEEE Trans. Multimedia*, vol. 25, pp. 792–804, Mar. 2023, doi: 10.1109/TMM.2021.3132156.
[44] H. Tian et al., "DIMA: Distributed cooperative microservice caching for Internet of Things in edge computing by deep reinforcement learning," *World Wide Web*, vol. 25, pp. 1769–1792, 2022.
[45] F. Wang, F. Wang, J. Liu, R. Shea, and L. Sun, "Intelligent video caching at network edge: A multi-agent deep reinforcement learning approach," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 2499–2508.
[46] E. E. Ugwuanyi, S. Ghosh, M. Iqbal, T. Dagiuklas, S. Mumtaz, and A. Al-Dulaimi, "Co-operative and hybrid replacement caching for multi-access mobile edge computing," in *Proc. Eur. Conf. Netw. Commun.*, 2019, pp. 394–399.
[47] 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.*, 2018, pp. 1–6.
[48] 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./Dec. 2018.
[49] W. Jiang, G. Feng, S. Qin, T. S. P. Yum, and G. Cao, "Multi-agent reinforcement learning for efficient content caching in mobile D2D networks," *IEEE Trans. Wireless Commun.*, vol. 18, no. 3, pp. 1610–1622, Mar. 2019.

Junkun Peng received the BS degree in information management and information systems from Shanghai University, Shanghai, China, in 2015. He is currently working toward the PhD degree in computer science with Tsinghua University. His research focuses on in-network caching/computing, and robot learning.

Qing Li (Senior Member, IEEE) received the BS degree in computer science and technology from the Dalian University of Technology, Dalian, China, in 2008, and the PhD degree in computer science and technology from Tsinghua University, Beijing, China, in 2013. He is currently an associate researcher with the Peng Cheng Laboratory, Shenzhen, China. His research focuses on network function virtualization, in-network caching/computing, intelligent self-running networks, and edge computing.

Xun Tang is currently a junior in intelligent science and technology from Sun Yat-sen University, Shen-Zhen, China. His research interests include real-time video transport, in-network caching, and UAV swarm intelligence.

Dan Zhao received the bachelor's degree majored in telecommunications from the Beijing University of Posts and Telecommunications, in 2011, and the PhD degree in electronic engineering from the Queen Mary University of London, in 2015. She is currently an assistant researcher with Peng Cheng Laboratory, Shenzhen, China. She was a postdoctoral researcher with the School of Electronic Engineering, Dublin City University.

Chuang Hu received the BS and MS degrees from Wuhan University, in 2013 and 2016, and the PhD degree from the Hong Kong Polytechnic University, in 2019. He is currently an associate researcher with the School of Computer Science, Wuhan University. His research interests include edge learning, federated learning/analytics, and distributed computing.

Yong Jiang (Member, IEEE) received the BS and PhD degrees from Tsinghua University, Beijing, China, in 1998 and 2002, respectively. He is currently a full professor with the Division of Information Science and Technology, Tsinghua Shenzhen International Graduate School, Shenzhen, China, and the Peng Cheng Laboratory, Department of Mathematics and Theories, Shenzhen. He mainly focuses on the future internet architecture, the Internet of Things, edge computing, and AI for networks.