MagNet: Cooperative Edge Caching by Automatic Content Congregating

Junkun Peng
International Graduate School, Tsinghua University
Shenzhen, China
Peng Cheng Laboratory
Shenzhen, China

Qing Li*
Peng Cheng Laboratory
Shenzhen, China
liq@pcl.ac.cn

Xiaoteng Ma
Tsinghua-Berkeley Shenzhen Institute, Tsinghua University
Shenzhen, China
Peng Cheng Laboratory
Shenzhen, China

Yong Jiang
International Graduate School, Tsinghua University
Shenzhen, China
Peng Cheng Laboratory
Shenzhen, China

Yutao Dong
International Graduate School, Tsinghua University
Shenzhen, China
Peng Cheng Laboratory
Shenzhen, China

Chuang Hu
the Hong Kong Polytechnic University
Hongkong, China

Meng Chen
Chongqing University
Chongqing, China

ABSTRACT

Nowadays, the surge of Internet contents and the need for high Quality of Experience (QoE) put the backbone network under unprecedented pressure. The emerging edge caching solutions help ease the pressure by caching contents closer to users. However, these solutions suffer from two challenges: 1) a low hit ratio due to edges' high density and small coverages. 2) unbalanced edges' workloads caused by dynamic requests and heterogeneous edge capacities. In this paper, we formulate a typical cooperative edge caching problem and propose the MagNet, a decentralized and cooperative edge caching system to address these two challenges. The proposed MagNet system consists of two innovative mechanisms: 1) the Automatic Content Congregating (ACC), which utilizes a neural embedding algorithm to capture underlying patterns of historical traces to cluster contents into some types. The ACC then can guide requests to their optimal edges according to their types so that contents congregate automatically in different edges by type. This process forms a virtuous cycle between edges and requests, driving a high hit ratio. 2) the Mutual Assistance Group (MAG), which lets idle edges share overloaded edges' workloads by forming temporary groups promptly. To evaluate the performance of MagNet, we conduct experiments to compare it with classical, Machine Learning (ML)-based and cooperative caching solutions using the real-world trace. The results show that the MagNet can improve the hit ratio from 40% and 60% to 75% for non-cooperative and cooperative solutions, respectively, and significantly improve the balance of edges' workloads.

CCS CONCEPTS

• **Networks** → **Cloud computing**; • **Information systems** → **Multimedia streaming**.

KEYWORDS

edge computing, cache, cooperative, workload balance, embedding

ACM Reference Format:
Junkun Peng, Qing Li, Xiaoteng Ma, Yong Jiang, Yutao Dong, Chuang Hu, and Meng Chen. 2022. MagNet: Cooperative Edge Caching by Automatic Content Congregating. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3485447.3512146

1 INTRODUCTION

In recent years, the backbone network has faced severe pressure. It mainly arises from the rapid growth of the network users, the upgrade of video quality from traditional 1080P high-definition content to 4K and 8K levels [39], and the emergence of sophisticated human-computer interactions like 360 videos [34], VR [20], and AR [6], which are more sensitive to latency [25].

In order to alleviate the pressure, mature Content Delivery Network (CDN) solutions [22, 32] have been widely used. However, nodes in CDN solutions are limited and are still far away from users, which results in high latency and low QoE [4].

The state-of-the-art edge computing technology has provided a new perspective for content distribution. Edges can be used to significantly reduce the distances between contents and users, which can save enormous transmission traffic in the backbone and lower

*Corresponding author

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France

ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3512146

the latency effectively [18]. However, as the distribution of edges is dense and their serving areas are small, requests are sent to different edges, resulting in a low hit ratio. We prove this in Section 5. 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 workloads' balance even worse.

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. To solve it, we propose the MagNet, a decentralized and cooperative edge caching system. The 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 traces. 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. 2) the Mutual Assistance Group (MAG) mechanism. When an overloaded edge emerges, the MAG finds some idle edges for it to form a temporary group. The group runs in a master-worker mode where the master, i.e., the overloaded edge, shift some of its workloads to the workers, i.e., idle edges, according to their capacities to balance their workloads. 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 trace dataset. We compare the MagNet with some benchmark caching solutions, including classical, ML-based and cooperative solutions. The result shows that the MagNet can improve the hit ratio from 40% and 60% to 75% for non-cooperative and cooperative solutions, respectively, and significantly balance the edges' workloads.

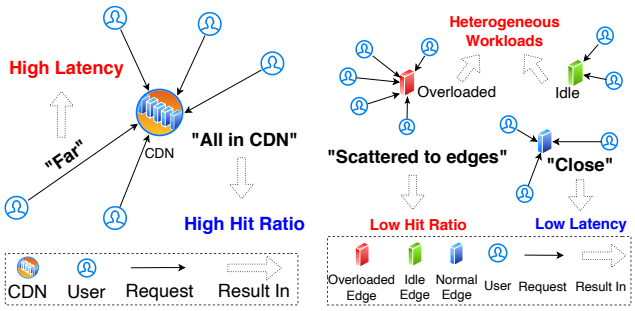
2 BACKGROUND AND MOTIVATION

2.1 Background

It has been predicted by Cisco that Internet video streaming will occupy 82% of all Internet business traffic by 2022 [6]. One unique property of the traffic is that the content requests are highly concentrated [5]. Thus, caching technology has become an essential solution [36]. It can reduce requests' latency, which can significantly improve users' Quality of Experience (QoE) [27], and relieve the heavy burden of network transmission [23]. There are two types of caching solutions: 1) traditional CDN 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 [1, 8]. Thus, as shown in Figure 1a, all requests must be aggregated to the central node, which causes high latency [7] and low QoE [4].

The edge computing is drawing unprecedented attention with its rapid evolution and is used to address various challenges [26] including the efficiency of content delivery and caching [23]. According to estimates by the Uptime Institute [38], by 2021, half of all workloads will be offloaded from data center infrastructures to edge computing devices. Compared with CDN nodes, edges are closer to users, which deliver contents to users with a lower latency when requests hit [18]. However, edge caching solutions still face some challenges due to their characteristics.

2.2 Challenges of Edge Caching Solution



(a) CDN Caching (b) Edge Caching

Figure 1: Comparison between CDN solutions and edge caching solutions

First, edge caching solutions suffer from a **Low Hit Ratio**. The hit ratio result severely affects requests' latency and network traffic [28]. As illustrated in Figure 1, in contrast with a CDN node that serves a large area, each edge's serving area is small since edges are densely distributed. As such, each edge aggregates fewer requests, which results in a low hit ratio of edge caching solutions. Considering a typical edges' density [42], like one edge per square kilometer, even with the state-of-the-art machine-learning caching algorithms, the hit ratios of non-cooperative caching systems are low. This is demonstrated in Section 5 of the paper. Therefore, cooperation among edges is necessary.

Second, edge caching solutions face **Heterogeneous Workloads** due to the limited and heterogeneous capacities and dynamic requests. Most edges are limited in caching and computational capacities [23, 30], 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. However, most cooperative edge caching solutions [14, 19, 37, 41, 43–45] assume edges have the same caching and computational capacities. Moreover, the number of requests varies greatly in different areas, resulting in dynamic requests [41]. The dynamic requests add more heterogeneity to the edges' workloads. Therefore, allocating appropriate workloads according to edges' capacities can improve the flexibility and robustness of a cooperative edge caching solution under highly dynamic requests.

2.3 Related Work

In recent years, some caching solutions have been proposed to **Improve The Hit Ratio**. The Least Recently Used (LRU) [15], the Least Frequently Used (LFU) [21] and the Adaptive Replacement Cache (ARC) [24] are widely used in industry. However, these solutions consider only the frequency and recency of requests and are therefore hard to adapt to highly dynamic requests. With the rapid development of ML, some ML-based caching solutions have been proposed. The LeCaR [40] and CacheUs [29] use ML to identify which eviction strategy is currently suitable. The Cocktail [45] utilizes an ensemble method to learn the dynamic trend of user requests. The CACA [11] 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. Some cooperative caching solutions have been proposed to promote the hit ratio by edges' cooperation. The CCSP [19] allows requests to be sent to at most four edges for a high hit ratio. The DIMA [35] and MacoCache [41] use the deep reinforcement learning to control the forwarding logic to pursue a high hit ratio. Nevertheless, they cost enormous computational capacities in edges which have limited computational capacities.

Some cooperative edge caching solutions have also been proposed to **Utilize Heterogeneous Capacities**. There are two types of solutions to organize the cooperative edge caching systems: 1) centralized cooperative solutions [19, 37, 43, 44], which use some central edges to record and manage the information. 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 [14, 41], which manage edges in a distributed way. These solutions forward requests to many of their neighbors, which linearly increases edges' burden with the increased number of forwarding.

3 SYSTEM DESIGN

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

3.1 Problem Formulation

3.1.1 Network Topology. The considered system includes a cloud server e_0 containing all the contents, E edges, C contents, and U users. Let $\mathcal{E} = \{e_1, e_2, \cdots, e_i, \cdots, e_E\}$ 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 $\mathcal{N}_i = \left\{e_{i_1}, e_{i_2}, \cdots, e_{i_{m_i}}\right\}$ denote the neighbors of edge e_i, where m_i is the number of neighbors and $\mathcal{N}_i \subset \mathcal{E}$. Let $\mathcal{C} = \{c_1, c_2, \cdots, c_j, \cdots, c_C\}$ denote the set of contents and s_j be the size of transmission with content c_j. Specially, let s_0 denote the size of transmission without any content. Let $\mathcal{U} = \{u_1, u_2, \cdots, u_g, \cdots, u_U\}$ denote the set of users.

3.1.2 Load Constraint Modeling. We consider a system operating over a finite time horizon. The requests $\mathcal{R} = \left[r_1, r_2, \cdots, r_p, \cdots, r_R\right]$ are generated by users in order, where R is the number of requests. Let $e_{p'}$ [1] 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 firstly 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 $\mathcal{R}_i = \left[r_{i_1}, r_{i_2}, \cdots, r_{i_{R_i}}\right]$, where R_i is the number of requests received by e_i.

In this paper, we use LRU [15] as the default cache replacement strategy for edges.

Given the caching capacity f_i and the edge's history requests $\left[r_{i_1}, r_{i_2}, \cdots, r_{i_k}\right]$, the current edge caching contents is given as $LRU\left(f_i, \left[r_{i_1}, r_{i_2}, \cdots, r_{i_k}\right]\right) = UNIQ_{y^*}$, where $UNIQ$ is the set of unique contents, $UNIQ_y = \left\{c_{p''} | r_p \in \left[r_{i_y}, r_{i_{y+1}}, \cdots, r_{i_k}\right]\right\}$ and $y^* = \min\left\{y || UNIQ_y| \le f_i\right\}$. 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:

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

The edge e_i has limited caching capacity, therefore

$$|LRU\left(f_i, \left[r_{i_1}, r_{i_2}, \cdots, r_{i_k}\right]\right)| \le f_i, \forall e_i \in \mathcal{E}. \tag{2}$$

The cloud server e_0 contains all the contents, therefore

$$y_0^p = 1, \forall r_p \in \mathcal{R}. \tag{3}$$

Let S_{CST} denotes the size of the Cache Summary Table (CST) that is exchanged between edges to notify each other their cache summaries. Let EUP denotes the period that an edge exchanges its CST with its neighbors. Let RT denotes 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 $l_{home}^h = l\left(u_{p'''}, e_{p'}, s_{p''}\right)$ if it hits, or $l_{home}^m = l\left(u_{p'''}, e_{p'}, s_0\right)$ 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 $l_{nei}^h = \sum_{\substack{e_i \in \mathcal{N}_{p'} \\ e_i \ne e_{p'_m}}} x_i^p l\left(u_{p'''}, e_i, s_0\right) + l\left(u_{p'''}, e_{p'_m}, s_{p''}\right)$. Let q denotes 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

$$0 \le q \le |\mathcal{N}_{p'}| < E. \tag{4}$$

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

[1] 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.

$h_{nei} = \sum_{e_i \in \mathcal{N}_{p'}} x_i^p y_i^p$. *Step 3:* If r_p is sent to the cloud server by neighbor ${e_{p'}}_m$, the latency of this step is $l_{e_0} = l\left(e_i, e_0, s_{p''}\right) + l\left(e_i, u_{p'''}, s_{p''}\right)$. Summing up the latency incurred in the above 3 steps, the total latency of request r_j is given as: $l_j = y_{p'}^p l_{home}^h + \left(1 - y_{p'}^p\right)\left(h_{nei} l_{nei}^h + (1 - h_{nei})\left(l_{nei}^m + l_{e_0}\right)\right)$. 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 $BT = \sum_{p \in \mathcal{R}} \left(1 - y_{p'}^p\right)(1 - h_{nei})\, s_{p''}$. 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:

$$WSD = \sqrt{\sum_{e_i \in \mathcal{E}} \left(R_i'' - \frac{\sum_{e_k \in \mathcal{E}} R_k''}{E} \right)^2}. \tag{5}$$

3.1.3 Cooperative Edge Caching Problem.

Problem 1. *Given $\alpha, \beta, \gamma, \theta, \mathcal{R}, \mathcal{E}, C, \mathcal{U}, s_0, S_{CST}$ and RT, find x_i^p and q to jointly minimize latency, traffic and WSD as follows:*

$$\begin{aligned} &min: \alpha L + \beta BT + \gamma LT + \theta WSD \\ &subject\ to: (1), (2), (3)\ and\ (4). \end{aligned} \tag{6}$$

3.1.4 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 [31] in a polynomial time. As the Helper Decision Problem has been proved as an NP-complete problem, 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.

3.2 MagNet Framework

The MagNet is a decentralized and 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 the content type, called the Guidance Type (GT). The MagNet clusters all contents into some GTs, which can be done offline and periodically, e.g., per day.

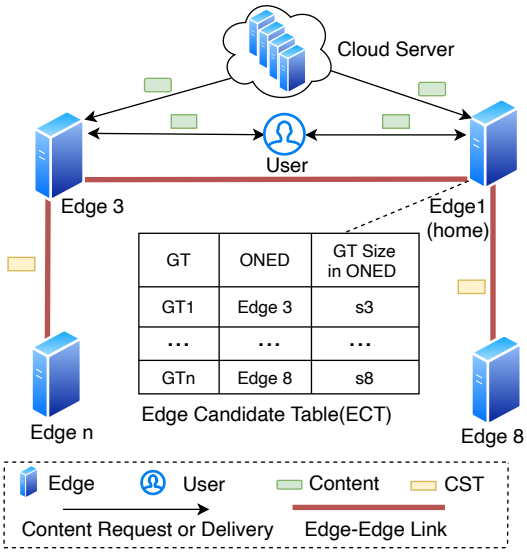


Figure 2: Framework of the MagNet

In MagNet, each edge has 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. Based on its and neighbors' CSTs, the edge builds or updates a table named Edge Candidate Table (ECT), as illustrated in Figure 2. In ECT, the edge designates each GT an Optimal Neighbor EDge (ONED). The process of designating ONEDs concerns many factors, e.g., the latency between the two neighbors and the cached-content size of the corresponding GT, which is another complex and valuable problem we may do more research on in the future. In this paper, without loss of generality, the neighbor which contains the maximum corresponding GT size is chosen as the ONED.

Every request in MagNet is processed in the following four 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 request goes to Stage 2.
- Stage 2: if the requested content has no GT, which means it is new coming after the last content clustering, the home edge fetches the content from the cloud server that stores all contents. Then, the home edge caches the content locally and sends it back to the user. Otherwise, the home edge suggests the user an ONED to try next based on the content's GT, and the request goes to Stage 3.
- Stage 3: the user sends the request to the ONED. If the ONED contains the content, the content is returned to the user. Otherwise, the request goes to Stage 4.

- Stage 4: the ONED fetches the content from the cloud server. Then, the ONED caches the content locally [2] and sends it back to the user.

During the processing of the request, only one neighbor edge is tried except for the home edge, because the performance of trying one neighbor edge is already good enough from experience. If there are sufficient edges' resources and network's resources, the framework of MagNet is easy to adapt to trying more neighbors for higher a hit ratio. As the way of choosing the ONED for one GT, it is easy to maintain the second and even third priority trying neighbors in an ECT.

Over every two minutes, each edge counts the number of received requests and updates its status: if this number is greater than 1.5 times the expected service quantity, it is treated as an Overloaded Edge (OE); if this number is less than 0.67 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. The values 1.5 and 0.67 are both adjustable factors, which can be customized to accommodate different system needs. When an OE appears, it tries to ally with some Idle Edges (IEs) to form a temporary Mutual Assistance Group (MAG) to alleviate the workload.

4 TWO MECHANISMS OF THE MAGNET

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

4.1 Automatic Content Congregating (ACC)

As discussed in Section 2.2, the low hit ratio caused by the lack of efficient cooperation among edges is a challenging problem in edge caching solutions, leading to high latency and excessive traffic. The ACC is designed to guide requests to their optimal edges evenly.

4.1.1 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 Figure 3, the history traces 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.

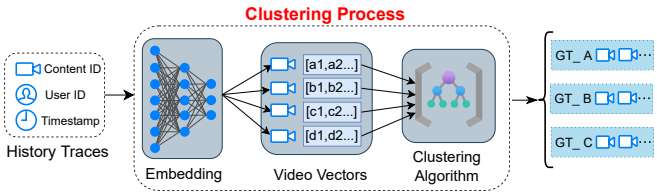


Figure 3: Contents Clustering Process

Content Embedding. The critical step of clustering is to capture the underlying pattern of the history traces. To this end, we use the embedding technology [10] to get the vector for every content [2]. The embedding technology utilizes a neural embedding algorithm to conduct collaborative filtering [33]. The vectors are generated in such a way that the distance between vectors is as close as possible to the probability they are requested consecutively by users. More specifically, first, all the contents accessed by one same user within a period, e.g., 1 day or 1 week, are extracted into one standalone set. According to collaborative filtering, the contents in one set are considered to have a closer relationship. Second, by using the item2vector [2] method, the skip-gram model [12], which is well packaged in the fastText library [3, 16], takes the extracted sets as input to calculate the vector representation of each content. 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.

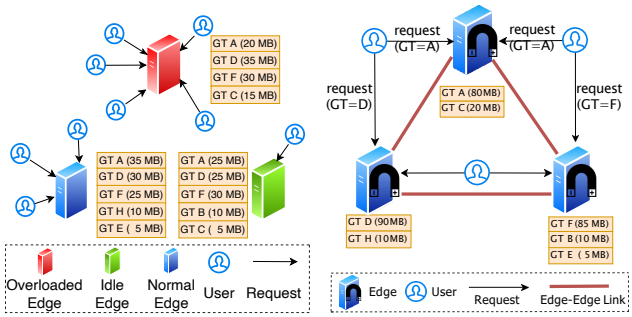
[2]Whether the fetched content should be cached or not depends on the caching replacement strategy that is out of the scope of this paper. In order to show the universality of the framework, all edges use the Least Recently Used (LRU) [15] caching strategy.

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**
    if $|vectors| = 0$ **then**
      break
    end if
    $v \leftarrow GetMaxDistanceVector(vectors, GTs[j])$
    $GTs[j][i++] \leftarrow v$
    $Remove(vectors, v)$
  end for
end while
return *GTs*

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 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 the intra-GT competition, the FCA is proposed. The FCA makes vectors that are far apart from each other clustered in the 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.

4.1.2 Guide Requests By GT. With the help of ECT, when one request misses in the home edge, the user can get a suggestion of the ONED based on the request's GT. This process causes an interesting phenomenon. Initially, when one GT's requests frequently appear in an area, the edge accumulates the contents of this GT (called the



(a) Without ACC (b) With ACC

Figure 4: Automatic Content Congregating (ACC)

dominant GT) more than those of the other GTs. In return, more requests of the dominant GT are attracted to the edge, which further drives up the cached contents of the dominant GT and forms the edge's unique hit advantage. Eventually, each edge accumulates its dominant GT contents and attracts requests of its dominant GT like a magnet. This virtuous cycle leads to a high hit ratio. Edges in MagNet forms a Magnetic Network (MagNet).

4.1.3 ECT Update Algorithm. Besides forming the competitive hit advantages, spreading the advantages to edges' neighbors is also essential. The spreading step ensures that each edge can perceive its neighbors' dominant GTs. The ECT is the key to spread hit advantages and is maintained by exchanging CSTs with their neighbors. The system should choose an appropriate cooperative scope controlled by the Number of Exchanging Neighbors (NEN) or the maximum distance threshold between two neighbor nodes. An overly small scope results in a low hit ratio, while an overly extensive scope causes heavy traffic between edges. The scope of this cooperation can be flexibly adjusted according to the needs of the system. In this paper, we choose $7km$ as the cooperative scope from experience. Over each ECT Updating Period (EUP), the edge fetches its neighbors' CSTs and updates its ECT according to Algorithm 2. In the updating process, for each GT, the MagNet chooses the edge that has the biggest cached GT size as its ONED. This choice means that for each GT, the edge with a bigger cached GT size is more attractive for requests of this GT. With GT and maintained ECT, requests can be guided to their ONEDs.

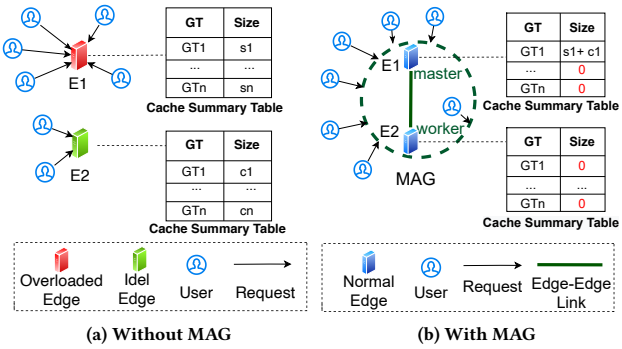
4.2 Mutual Assistance Group (MAG)

The MAG is designed to balance the edges' workloads under high dynamic requests. Balanced workloads facilitate a stable and robust system. In MagNet, each OE tries to find some IEs to form a temporary MAG according to Algorithm 3. IEs are added to a MAG one by one until the original OE of the MAG becomes an NE, or there is no IE left. 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. 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 Figure 5b, 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.

Algorithm 2 ECT Update Algorithm

Input: *CSTs* from neighbors, *ECT*
Output: *ECT*

```
for each CST in CSTs do
    for each entry in CST do
        if ECT[entry.GT].size < entry.size then
            let ECT[entry.GT].ONED ← CST.edge
            let ECT[entry.GT].size ← entry.size
        end if
    end for
end for
```



(a) Without MAG (b) With MAG

Figure 5: Mutual Assistance Group (MAG)

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 contents. The workers in a MAG will be released once the master becomes an IE. The MAG is operated on the fly and thus can help relieve a hot spot area's pressure promptly and promote the MagNet's resource utilization.

5 EVALUATION

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

5.1 Methodology

Dataset. Throughout this section, we use Trace1 described in Table 1 as the data set of the experiments. For Trace1, 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 Trace1 into two parts: 1) data of the first 12 days, which is used for training and analyzing purposes, and 2) data of the last day, which is used for testing and evaluation purposes.

Algorithm 3 MAG Running Algorithm

Input: *edge*,*workload*

while Every two minutes **do**
  $edge.status \leftarrow UpdateStatus(edge, workload)$
  if *edge.status*==OE **then**
    while *edge.status* ≠NE **and** Exist(IE) **do**
      $ie \leftarrow FindIE()$
      $JoinMAG(ie)$
      $UpdateMAG()$
    end while
  else if *edge.status*==IE **then**
    while $edge.status \neq NE$ **and** $edge.workers \neq null$ **do**
      ReleaseWorker()
      $UpdateMAG()$
    end while
  end if
end while

Name	From	Time Range	Amount	Area
Trace1	iQiYi	13 days	50 Million	Beijing City

Table 1: Trace Dataset

Testbed. Since the spatial area of Trace1, i.e., the whole Beijing city, is too big for experiment purposes, we choose a $20km \times 20km$ 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 [9] to implement the MagNet with object-oriented programing [17]. It runs on a computer with one GeForce GTX 1660 TI GPU, one Intel i7-9700 CPU and 16GB memory. Additionally, we build a multi-edge caching experiment platform. It accepts a prepared trace to simulate parallel sending requests to edges. It can measure edges' hit ratios and workloads. In the experiments, we use Trace1 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 [24]. 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 [40]. It is a caching framework that uses reinforcement learning and regrets minimization to adjust the usage of LRU and LFU dynamically.
- CACHEUS [29]. 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 [24] and LIRS [13].
- CCSP [19]. 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 Eq.(5) 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.

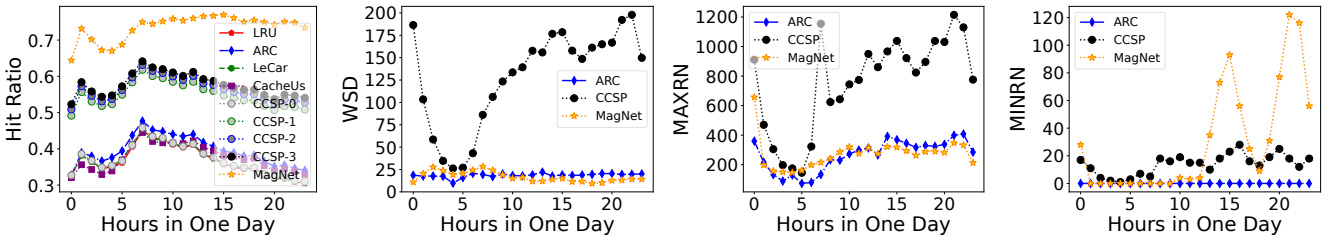
Since the same edge in different non-cooperative solutions receives the same amount of requests, we use the ARC as a representative in WSD, MAXRN and MINRN.

5.2 Overall Performance

We conduct experiments from three different perspectives.

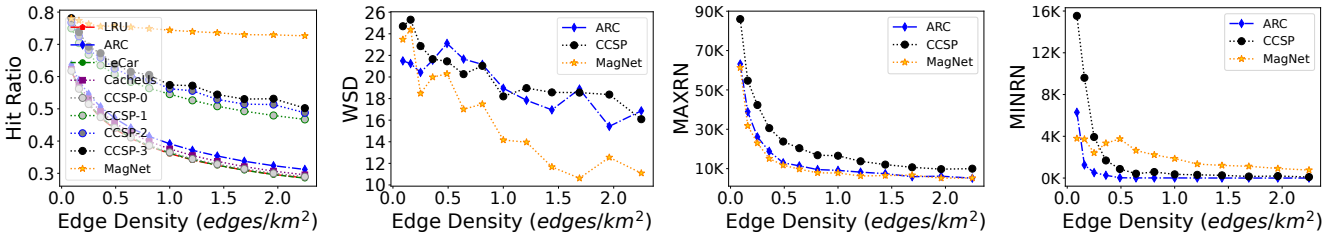
Performance in one whole day. In this case, we use the default standard setups. In Figure 6a, we find the MagNet has a significantly higher hit ratio than the others, exceeding 70% in the most time of one day. In Figure 6a, CCSP-0 represents CCSP where each request can try only its home edge; CCSP-1, 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, the MagNet selects only one more edge to try except its home edge, and it enjoys about a 15% higher hit ratio than the CCSP-3. This outstanding hit ratio performance is because the ACC helps guide requests to their optimal edges. Figure 6b shows the WSD of each hour, where the MagNet has lower WSD than others in most hours, especially in the request-intensive hours (13:00 to 24:00) [41], meaning it has the most balanced workload. Figure 6c and Figure 6d indicate that no edge is under excessive pressure and the MagNet utilizes edges efficiently. This case proves that the 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 Figure 7a, 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. 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%. Even though the MagNet only tries one more edge except for the home edge, it achieves an excellent hit ratio result, which only decreases only 5%. Figure 7b , 7c and 7d show that the workload balance of all solutions is better as the edge density increases, while the edges' workloads are reduced. Moreover, the MagNet has the best workload balance performance than the others from all three metrics. This case proves that the MagNet has an outstanding hit ratio performance and prominent workload balance in various edge-density areas.



(a) Hit Ratio Performance (b) Workloads' Standard Deviation (c) Max Requests' Number of One Edge (d) Min Requests' Number of One Edge

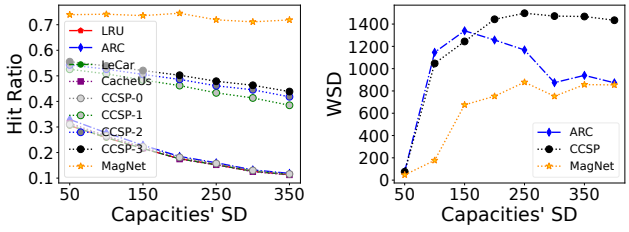
Figure 6: Performance in one whole day



(a) Hit Ratio Performance (b) Workloads' Standard Deviation (c) Max Requests' Number of One Edge (d) Min Requests' Number of One Edge

Figure 7: Different Edge Densities

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 Figure 8a, 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. Figure 8b reveals that the MagNet achieves a more balanced workload. In this case, the WSD values are much more prominent because the WSD considers the differences of capacities in Eq.(5). This case proves that the MagNet is more flexible and adaptable under the environment of heterogeneous edges' capacities.



(a) Hit Ratio Performance (b) Workloads' Standard Deviation

Figure 8: Different Capacities' Standard Deviation

6 CONCLUSION

In this paper, we analyze the challenges of edge caching solutions and propose a decentralized and cooperative edge caching system named MagNet. The 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. To evaluate the MagNet, we compare it with the classical, ML-based and cooperative caching solutions from three different perspectives using a real trace. The results prove that the MagNet significantly outperforms other solutions in terms of both hit ratio and workload balance.

7 CODE

To introduce the technical details of MagNet more clear, we share the code, which includes the Java version implementation of MagNet and the edge-based caching evaluation environment, at https://github.com/pengjunkun/MagNet.

ACKNOWLEDGMENTS

This work is supported by Guangdong Province Key Area R&D Program under grant No. 2018B010113001, National Natural Science Foundation of China under grant No. 61972189, and the Shenzhen Key Lab of Software Defined Networking under grant No. ZDSYS20140509172959989.

REFERENCES

[1] Akamai. 2021. Media Delivery Network Map. https://www.akamai.com/visualizations/media-delivery-network-map

[2] Oren Barkan and Noam Koenigstein. 2016. Item2vec: neural item embedding for collaborative filtering. In *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 1–6.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching Word Vectors with Subword Information. *arXiv preprint arXiv:1607.04606* (2016).

[4] Pedro Casas, Alessandro D'Alconzo, Pierdomenico Fiadino, Arian Bär, and Alessandro Finamore. 2014. On the analysis of QoE-based performance degradation in YouTube traffic. In *10th International Conference on Network and Service Management (CNSM) and Workshop*. IEEE, 1–9.

[5] Joonho Choi, Abu Sayeem Reaz, and Biswanath Mukherjee. 2012. A Survey of User Behavior in VoD Service and Bandwidth-Saving Multicast Streaming Schemes. *IEEE Communications Surveys Tutorials* 14, 1 (2012), 156–169.

[6] Cisco. 2020. Cisco Annual Internet Report (2018–2023) White Paper. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[7] Xun Fan, Ethan Katz-Bassett, and John Heidemann. 2015. Assessing affinity between users and CDN sites. In *International Workshop on Traffic Monitoring and Analysis*. Springer, 95–110.

[8] Google. 2021. Google Cloud CDN Locations. https://cloud.google.com/cdn/docs/locations

[9] James Gosling. 1997. The feel of Java. *Computer* 30, 6 (1997), 53–57.

[10] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 151 (2018), 78–94.

[11] Yu Guan, Xinggong Zhang, and Zongming Guo. 2019. CACA: Learning-Based Content-Aware Cache Admission for Video Content in Edge Caching. In *Proceedings of the 27th ACM International Conference on Multimedia*. ACM, 456–464.

[12] David Guthrie, Ben Allison, Wei Liu, Louise Guthrie, and Yorick Wilks. 2006. A closer look at skip-gram modelling.. In *LREC*, Vol. 6. Citeseer, 1222–1225.

[13] Song Jiang and Xiaodong Zhang. 2002. LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance. *ACM SIGMETRICS Performance Evaluation Review* 30, 1 (2002), 31–42.

[14] Wei Jiang, Gang Feng, Shuang Qin, Tak Shing Peter Yum, and Guohong Cao. 2019. Multi-agent reinforcement learning for efficient content caching in mobile D2D networks. *IEEE Transactions on Wireless Communications* 18, 3 (2019), 1610–1622.

[15] HR Johnson and JA Larson. 1979. Data management for microcomputers. In *1979 Compcon Fall*. IEEE, 191–192.

[16] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. FastText.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651* (2016).

[17] Naoki Kobayashi and Akinori Yonezawa. 1994. Type-theoretic foundations for concurrent object-oriented programing. *ACM SIGPLAN Notices* 29, 10 (1994), 31–45.

[18] K Kumaran and E Sasikala. 2021. Learning based Latency Minimization Techniques in Mobile Edge Computing (MEC) systems: A Comprehensive Survey. In *2021 International Conference on System, Computation, Automation and Networking (ICSCAN)*. IEEE, 1–6.

[19] Ali Larbi, Louiza Bouallouche-Medjkoune, and Djamil Aissani. 2018. Improving cache effectiveness based on cooperative cache management in MANETs. *Wireless Personal Communications* 98, 3 (2018), 2497–2519.

[20] Anatole Lécuyer, Fabien Lotte, Richard B Reilly, Robert Leeb, Michitaka Hirose, and Mel Slater. 2008. Brain-computer interfaces, virtual reality, and videogames. *Computer* 41, 10 (2008), 66–72.

[21] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. 1999. On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies. In *Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. 134–143.

[22] Chia-Feng Lin, Muh-Chyi Leu, Chih-Wei Chang, and Shyan-Ming Yuan. 2011. The study and methods for cloud based CDN. In *2011 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*. IEEE, 469–475.

[23] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B. Letaief. 2017. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys Tutorials* 19, 4 (2017), 2322–2358.

[24] Nimrod Megiddo and Dharmendra S Modha. 2003. ARC: A Self-Tuning, Low Overhead Replacement Cache. In *Fast*, Vol. 3. USENIX Association, 115–130.

[25] Georgios Papaioannou and Iordanis Koutsopoulos. 2019. Tile-based caching optimization for 360 videos. In *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM, 171–180.

[26] Quoc-Viet Pham, Fang Fang, Vu Nguyen Ha, Md Jalil Piran, Mai Le, Long Bao Le, Won-Joo Hwang, and Zhiguo Ding. 2020. A survey of multi-access edge computing in 5G and beyond: Fundamentals, technology integration, and state-of-the-art. *IEEE Access* 8 (2020), 116974–117017.

[27] Stefan Podlipnig and Laszlo Böszörmenyi. 2003. A Survey of Web Cache Replacement Strategies. *ACM Comput. Surv.* 35, 4 (2003), 374–398.

[28] Buvaneswari A Ramanan, Lawrence M Drabeck, Mark Haner, Nachi Nithi, Thierry E Klein, and Chitra Sawkar. 2013. Cacheability analysis of HTTP traffic in an operational LTE network. In *2013 Wireless Telecommunications Symposium (WTS)*. IEEE, 1–8.

[29] Liana V. Rodriguez, Farzana Yusuf, Steven Lyons, Eysler Paz, Raju Rangaswami, Jason Liu, Ming Zhao, and Giri Narasimhan. 2021. Learning Cache Replacement with CACHEUS. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*. USENIX Association, 341–354.

[30] Mahadev Satyanarayanan. 2017. The emergence of edge computing. *Computer* 50, 1 (2017), 30–39.

[31] Karthikeyan Shanmugam, Negin Golrezaei, Alexandros G Dimakis, Andreas F Molisch, and Giuseppe Caire. 2013. Femtocaching: Wireless content delivery through distributed caching helpers. *IEEE Transactions on Information Theory* 59, 12 (2013), 8402–8413.

[32] Ao-Jan Su, David R Choffnes, Aleksandar Kuzmanovic, and Fabian E Bustamante. 2009. Drafting behind Akamai: Inferring network conditions based on CDN redirections. *IEEE/ACM transactions on networking* 17, 6 (2009), 1752–1765.

[33] Xiaoyuan Su and Taghi M Khoshgoftaar. 2009. A survey of collaborative filtering techniques. *Advances in artificial intelligence* 2009 (2009).

[34] Anthony Tang and Omid Fakourfar. 2017. Watching 360 videos together. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 4501–4506.

[35] Hao Tian, Xiaolong Xu, Tingyu Lin, Yong Cheng, Cheng Qian, Lei Ren, and Muhammad Bilal. 2021. DIMA: distributed cooperative microservice caching for internet of things in edge computing by deep reinforcement learning. *World Wide Web* (2021), 1–24.

[36] Anh-Tien Tran, Demeke Shumeye Lakew, The-Vi Nguyen, Van-Dat Tuong, Thanh Phung Truong, Nhu-Ngoc Dao, and Sungrae Cho. 2021. Hit Ratio and Latency Optimization for Caching Systems: A Survey. In *2021 International Conference on Information Networking (ICOIN)*. IEEE, 577–581.

[37] Emeka E. Ugwuanyi, Saptarshi Ghosh, Muddesar Iqbal, Tasos Dagiuklas, Shahid Mumtaz, and Anwer Al-Dulaimi. 2019. Co-Operative and Hybrid Replacement Caching for Multi-Access Mobile Edge Computing. In *2019 European Conference on Networks and Communications (EuCNC)*. IEEE, 394–399.

[38] Uptime. 2021. Uptime Institute Global Data Center Survey 2021. https://uptimeinstitute.com/

[39] Linh Van Ma, Van Quan Nguyen, Jaehyung Park, and Jinsul Kim. 2018. NFV-Based Mobile Edge Computing for Lowering Latency of 4K Video Streaming. In *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 670–673.

[40] Giuseppe Vietri, Liana V. Rodriguez, Wendy A. Martinez, Steven Lyons, Jason Liu, Raju Rangaswami, Ming Zhao, and Giri Narasimhan. 2018. Driving Cache Replacement with ML-based LeCaR. In *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*. USENIX Association.

[41] Fangxin Wang, Feng Wang, Jiangchuan Liu, Ryan Shea, and Lifeng Sun. 2020. Intelligent Video Caching at Network Edge: A Multi-Agent Deep Reinforcement Learning Approach. In *IEEE INFOCOM 2020*. 2499–2508.

[42] Xinhua. 2021. China reports expansion in 5G network coverage. http://english.www.gov.cn/statecouncil/ministries/202107/18/content_WS60f424d4c6d0df57f98dd302.html

[43] Chen Zhong, M Cenk Gursoy, and Senem Velipasalar. 2018. A deep reinforcement learning-based framework for content caching. In *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 1–6.

[44] Hao Zhu, Yang Cao, Wei Wang, Tao Jiang, and Shi Jin. 2018. Deep reinforcement learning for mobile edge caching: Review, new features, and open issues. *IEEE Network* 32, 6 (2018), 50–57.

[45] Tongyu Zong, Chen Li, Yuanyuan Lei, Guangyu Li, Houwei Cao, and Yong Liu. 2021. Cocktail Edge Caching: Ride Dynamic Trends of Content Popularity with Ensemble Learning. In *IEEE INFOCOM 2021*. IEEE, 1–10.