Computer Communications 194 (2022) 301–310

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Chameleon: A Self-adaptive cache strategy under the ever-changing access frequency in edge network

Pengmiao Li [a], Yuchao Zhang [a,*], Wendong Wang [a,*], Weiliang Meng [b], Yi Zheng [b], Ke Xu [c], Zhili Zhang [d]

[a] State key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China
[b] ChuangCache company, Beijing, China
[c] Tsinghua University, Beijing, China
[d] University of Minnesota, Minneapolis, United States of America

ARTICLE INFO

Keywords:
Hit rate
Edge caching
Cache admission
Cache eviction
Access frequency

ABSTRACT

In recent years, with the maturity of 5G and Internet of Things technologies, the number of mobile applications and the amount of data access have increased explosively. However, the frequency of these accesses varies considerably at different times of the day, requiring different caching strategies in those limited-capacity edge servers. Existing caching strategies perform well when the access frequency is stable. However, they ignore the time-varying characteristics of user access frequency in different periods, resulting in a low hit rate in ever-changing frequency scenarios. To improve the hit rate in such scenarios, we propose a cache replacement policy called *Chameleon*, which consists of two components, *AutoFre*, and *Crates*. *AutoFre* is an admission algorithm that predicts the future access frequency category and calculates the admission thresholds based on the prediction result. While *Crates* is an eviction algorithm, it selects the contents evicted by designing a customized principal component analysis algorithm. We conduct a series of experiments with real application traces from ChuangCache. The trace has 9,839,213 user accesses in 48 h. The results demonstrate that *Chameleon* reaches about 98% in caching hit rate and outperforms *SecondHit-Crates* algorithm about 8% in frequency-changing edge networks.

1. Introduction

In recent years, with the maturity of 5G and Internet of Things technologies, mobile users are growing explosively. According to reports [1,2], nearly 300 million mobile applications will be downloaded by 2023, and the number of global mobile subscribers will reach 5.7 billion by 2023 from 5.31 billion in 2021, which introduces high requirements for content storage. To reduce the burden on cloud data centers and CDN networks, edge storage servers closer to users are widely deployed to cache popular content and provide a higher quality of service (QoS) by shortening access latency. However, storage resources on edge servers are limited compared with CDN servers. Therefore, the research on the cache replacement strategy of edge servers is critical to edge computing and storage areas.

Many studies have improved caching performance on edge servers. Such as LRU, LFU [3,4], or their simple variants [5] are easy to be implemented and have been widely used. Akamai [6] implemented the Cache-on-second-hit (*SecondHit*) [7] rule algorithm, which caches the content only when it is accessed twice. However, the total number of accesses received by the edge server in consecutive periods is variable due to the different user preferences for application usage, which is called time-varying. Although these works increase the hit rate on edge servers during stable frequency access, they ignore the time-varying characteristics of user access frequency in different periods.

Considering the mentioned conditions, we deeply analyze the real traces from ChuangCache [8] in China and find an essential observation (see Section 3) that is the user and content accessed has different concentrations at different frequencies periods. This observation provides a new perspective to review this caching problem, from the perspective of user and content concentrations. Therefore, this paper is mainly committed to designing the caching replacement algorithm utilizing the user and content concentration to improve the hit rate on edge servers in the ever-changing access frequency scenario.

To address this problem, we firstly predict the frequency category in the future by Decision Tree based on the content concentration information. Then we combine the prediction result and historical hit rate to design the self-adaptive admission threshold for admitting the access content or not. When admitting cache the content requested, we adopt the principal component analysis algorithm to select the

* Corresponding authors.
E-mail addresses: yczhang@bupt.edu.cn (Y. Zhang), wdwang@bupt.edu.cn (W. Wang).

https://doi.org/10.1016/j.comcom.2022.07.036
Received 17 March 2022; Received in revised form 13 June 2022; Accepted 20 July 2022
Available online 29 July 2022

eviction contents and remove them. On this basis, we finally propose a popular content protection mechanism to prevent the popular content cached from being removed. Through a series of experiments with different cache sizes and actual application data with two days of about 9,839,213 access frequency from ChuangCache in China, we demonstrate that *Chameleon* reaches about 98% in caching hit rate and outperforms the *SecondHit-Crates* algorithm by about 8%. The main contributions of this paper are listed as follows:

- We analyze real traces and find that the user and content concentrations change correspondingly with access frequency. It is a new perspective to address the low hit rate problem in the ever-changing frequency scenarios.
- We propose a self-adaptive cache replacement policy *Chameleon* for different access frequencies, which uses the Decision Tree algorithm to predict the range of future access frequencies. Then we design corresponding admission and eviction policies based on the prediction results and features above.
- We conduct a series of experiments using industry data and demonstrate the efficiency of *Chameleon*.

The remainder of the paper is organized as follows. Next section, we introduce the related work about replacement strategies for admission and eviction. In Section 3, we deeply analyze the real traces and find an essential observation. In Section 4, the design of architecture is presented. Section 5 conducts experiments with a detailed introduction and analysis of the results. And our conclusions are presented in Section 6.

2. Related work

The explosive growth of Internet data and the emergence of new networks, such as edge computing, IoT, and 5G network, have put forward higher requirements for caching strategies. There are two typical analyses of caching strategies. One focuses on content admission to decide whether to cache content or not. The other caching strategy focuses on content eviction to decide which content to evict.

2.1. Admission policy

Only a small amount of content is stored on the edge servers because their edge storage resources are limited. Researchers recognize that not all content is the same in terms of access frequency, recency, size, etc., which makes it necessary to design an Admission Policy in cache replacement. Akamai [6] counted the number of web file accesses in a server cluster over two days and found that of the total 400 million or so files, 74% were accessed only once. Based on this feature, it implemented the Cache-on-second-hit (*SecondHit*) [7] rule algorithm, which caches the content only when it is accessed twice. *TinyLFU* [9], a cache admission policy also based on access frequency, determines whether the content is cached in edge servers by the freshness of the accessed content. Edge server tends to store smaller content rather than a bigger one so that it can cache more content. Daniel et al. proposed *AdaptSize* [10] based a novel Markov cache model to adjust a threshold for the size of admitted objects. Combining these factors (frequency, size, and recency), *RL-Cache* proposed a novel algorithm *RL-Cache* that adopts model-free reinforcement learning to decide whether or not to cache an accessed object in CDNs. *SecondHit*, *AdaptSize*, *TinyLFU* and *RL-Cache* have improved the hit rate significantly.

2.2. Eviction policy

Many traditional eviction algorithms have been proposed in different scenarios, such as First Input First Output (FIFO), Least Recently Used (LRU), Least Frequently Used (LFU), and Greedy-Dual-Size-Frequency (GDSF) [3,4,11,12]. These traditional caching algorithms with low complexity are easily implemented and have been widely used. The most commonly deployed caching replacement algorithms by some CDN providers, such as LRU, LFU, or their simple variants [5], are simple but do not explicitly consider the future popularity of content when making caching decisions.

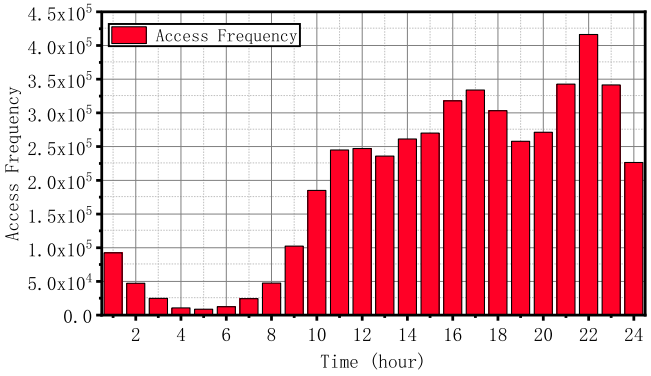


Fig. 1. Access frequency in every hour.

In the studies of eviction policies, researchers have proposed many solutions from different perspectives [13–18], such as location, size, and new scenarios. These studies are mainly based on content popularity to design. Paper [19] focused on caching in CDNs and proposed a new approach called LRB that uses machine learning to approximate the Belady MIN (oracle) algorithm. For better adaptation to the time-varying popularity patterns, a Forecast-Based cache replacement policy has been proposed for mobile video streaming [20]. In recent years, there has been a rapid increase in short video traffic about CDN, [21] presented AutoSight. It is a distributed edge caching system for short video networks, which consists of two main components a predictor (CoStore) and a caching engine (Viewfinder). In addition, the forecasting popularity [22–27] of online content has been extensively studied to provide technical support for caching.

Although these admission or eviction policies can increase the hit rate on edge servers, they ignore the access of variability frequency affects the caching performance. The reasons that make them inefficient in improving the hit rate of edge servers will be detailedly described in Section 3.

3. Motivation

In this section, we firstly find that the variability frequency feature leads to some challenges for improving the hit rate of edge servers by analyzing the real trace from ChuangCache in China in Section 3.1. Then we discuss possible solutions from new perspectives to address this problem in Section 3.2.

3.1. Content access frequency is time-varying

Due to different user preferences, edge servers receive the frequency of content accessed changeably at different times. Fig. 1 illustrates the total number of accesses during one day from ChuangCache in China. It is worth noting that all the data analyzed in this paper comes from the access log within 24 h of an application in ChuangCache. The average hourly user access frequency is 192,772 per hour, and the average number amount of access content accessed hourly is 32,635 per hour. In Fig. 1, it is easy to find a variation in frequency accessed with different hours. For example, the frequency of content accessed is 416,468 during the 22nd hour. However, the frequency of content accessed is 342,634 during the 21st hour, as for the 23rd hours, the frequency is 341,468. In general, the access frequency varies significantly per hour. For example, the access frequency from 21st to 23rd hours increased 33 times compared to the 4th–6th hours. The total number of content access frequency is time-varying and leads to a low hit rate, which is due to two reasons as follows.

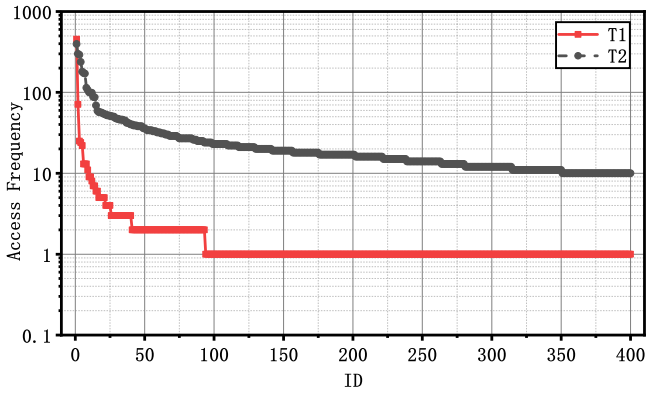


Fig. 2. The access frequency of contents in different period.

- **Admission: thresholds setting in different frequency accessed**
  Statistically [6], edge storage servers have large requests during periods, and about 74% of the content is accessed only once. The researchers believe that not having these contents stored improves the hit rate. Some researchers have proposed *SecondHit*, which allows the accessed content into the cache only on the second access time within a period. However, the cache hit rate is not optimized all the time by admitting the content accessed with second access and may sometimes be reduced. For example, two lists $l_1 = \{ABACBDEFAC\}$ and $l_2 = \{ABCDCDEF\}$ have the same contents and different access frequencies. The hit rates of the two lists with different admission thresholds for the same eviction algorithm (LRU) are calculated separately. The hit rate by first access (all) admission and *SecondHit* for list l_1 is 3/10 and 1/2, and list l_2 are 1/2 and 1/4, respectively. When the second access times as admission thresholds, the hit rate is higher than the first one in list l_1 and the opposite in list l_2. Therefore, we can obtain a viewpoint that using a fixed admission threshold at the different frequencies accessed may reduce the hit rate.
- **Eviction: popular content prediction at different frequency accessed**
  It is not difficult to understand that the greater the differences between the access frequency for different contents, the easier it is to select content as popular ones. What is particularly noteworthy here is that the differences between the frequency of accesses for different contents are practically indistinguishable in some periods. In Fig. 2, the T1 (see red line) and T2 (see black line) respectively present the access frequency from the top 400 contents during hour range 1–8 and other ranges.
  We can find that the access frequency is 2 for nearly 50 contents after Id 50 in T1, which lead difficult to select content as future popular content in the same access frequency. However, the difference between the access frequencies of different contents in T2 is more evident compared to T1. It indicates that popular contents are difficult to predict in some periods (like T1) than in others.

Based on the above analysis, we get two reasons that the hit rate needs to be improved in the ever-changing frequency scenario. To improve the hit rate, we deeply analyze the real traces from ChuangCache and find new observations about the concentration of content and user accessed. Therefore, the next subsection uncovers the potential of solving the above problem by analyzing the concentration characteristics in real traces.

3.2. Concentration varies with access frequency

It is not difficult to imagine that the hit rate is likely to drop when requesting more types of content because of the limited storage space. In addition, the access frequency is partly determined by the number of users online. Therefore, based on these two viewpoints, we analyze the concentration of contents and users and find the following two perspectives to solve the problem introduced in Section 3.1.

- **Admission: content concentration**.
  Although access frequency is variable over successive periods, the same access frequency exists under different periods. Under the same total access frequency, it is easy to believe that a substantial variation in access content frequency leads to a higher hit rate than a tiny difference, whether in admission or eviction algorithms. As a result, using access frequency alone as the basis for admission threshold with variation is insufficient. To fully capture the features of different periods, we illustrate the real access log information from one day in Fig. 3. The red line is the standard deviation of the content requested number. In Fig. 3, we can see that the standard deviation of the accessed content (which we call content concentration) varies even for the same access frequency in consecutive periods. Therefore, the access frequency and content concentration are combined to refine the characteristics of extracting different access times in this paper. It provides information to support the setting of access thresholds under different access frequencies.
- **Eviction: user concentration**.
  The number of access frequencies is inextricably linked with users. More specially, the number of specific user groups is tiny but with a high access frequency [28]. We assume that these users play an important role in cache replacement for improving hit rates. Therefore, we capture the characteristics of specific user groups by analyzing the real traces shown in Fig. 3. The red line is the percentage of the user number of specific user groups among all users. As seen in red, specific user groups are distributed differently throughout the day. Their accesses to content are more concentrated than other user groups during hours 1–6, which is attributed to their higher average number of content accesses [28] than during other periods. To sum up, specific user group play an essential role in those low-frequency periods than the general users. The contents accessed by this specific user group have a much higher possibility to become popular content.

Based on the above findings, we can provide new perspectives to examine the above-caching problems by predicting frequency to design a self-adaptive admission threshold and adopting specific user groups information to capture popular contents as eviction policy in the low-frequency period. However, there are two challenges: *(a) How to predict the future frequency and design a self-adaptive admission policy based on the prediction result, (b) How to capture the relationship and popular content with specific user groups, and improve hit rate based on the relationship on edge servers.* To address the challenges, we propose solutions described in detail in the next section.

4. System models and design

In this section, we firstly present the caching background in Section 4.1. Then we present the admission algorithm *AutoFre* by decision tree technology to predict the future frequency category in Section 4.2. Next, we adopt the principal component analysis algorithm to obtain the relationship between popular contents and core users in Section 4.3.1. On this basis, we propose a popular contents protection mechanism in Section 4.3.2. Finally, we introduce the overall design of caching strategy of *Chameleon* in Section 4.4.

4.1. Background

The framework overview is shown in Fig. 4, which mainly includes network infrastructures, the flow of access contents by users, and a caching replacement strategy. The network infrastructure is made up of clients, edge servers, CDNs, and Cloud. In this architecture, content statuses of users' accesses include: hit, CDN hit, and miss. When users request content, it is searched in edge servers, firstly. If the content

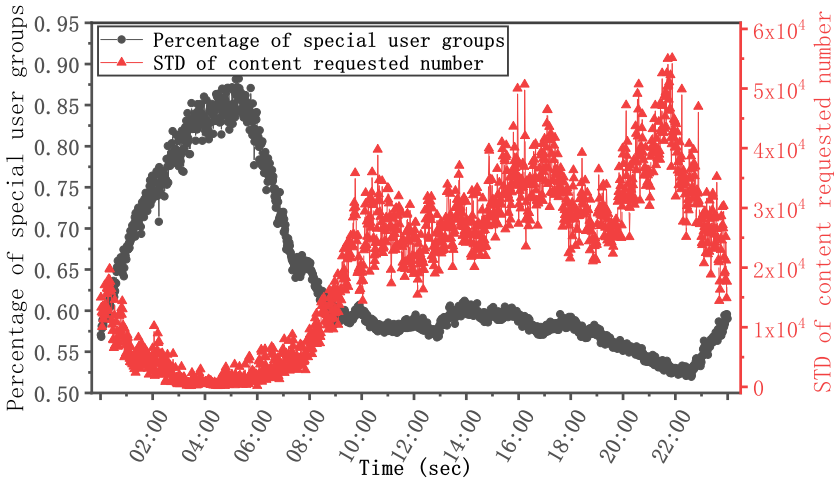


Fig. 3. The access information of Special user groups and content. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

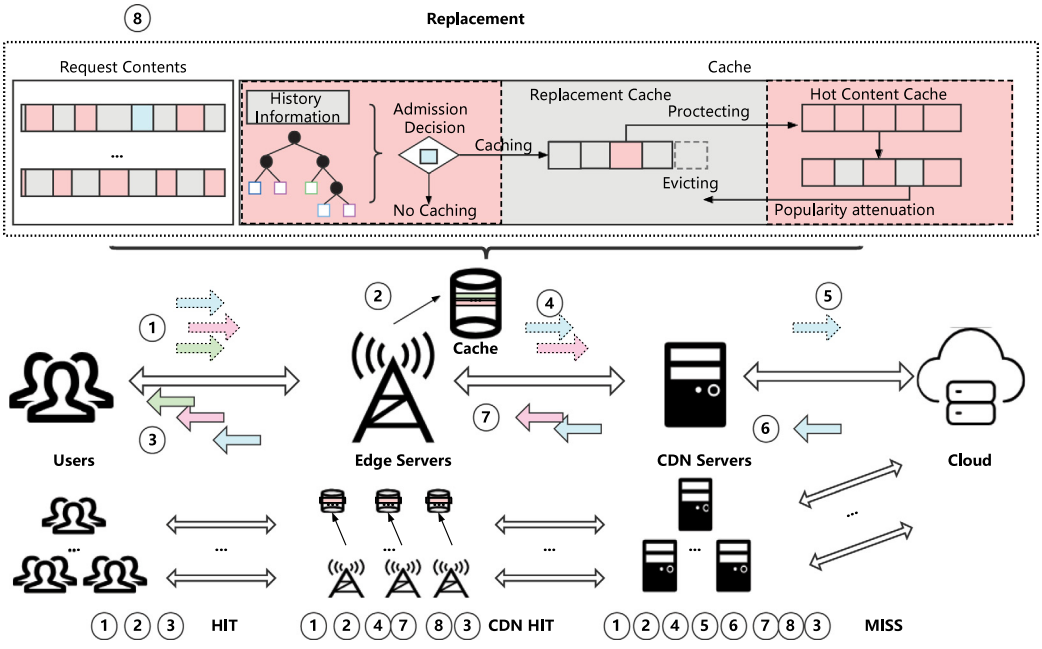


Fig. 4. The framework overview.

exists, we call its status hit, and it is sent to the user (①②③). If not, the content is found in CDNs as CDN hit (①②④⑦⑧③). Else, the access content is only stored in the remote Cloud, and the statue is miss (①②④⑤⑥⑦⑧③).

In this paper, we propose a caching strategy, *Chameleon*, which includes two parts *AutoFre* and *Crates*. The system architecture of *Chameleon* is shown in Fig. 5, which shows two cases when receiving a request. (1) When the edge server finds the requested content is requested missing, *AutoFre* makes the admission decision. If the missing content admits in caching, *Crates* selects the eviction contents, and then the edge server responds to the missing content after fetching it from the cloud. Otherwise, the missing content is responded to the user. (2) When the content requested is hit, the edge server responds to users directly. The following will introduce *AutoFre* and *Crates*, respectively.

4.2. Admission algorithm: AutoFre

The first challenge, to predict the future access frequency and design a self-adaptive admission policy based on the prediction result, is divided into two parts: (1) predicting future access frequency category by adopting a Decision Tree [29], and (2) defining a dynamic adaptive admission threshold based on the prediction result and historical hit rate, which will be detailed below Section 4.2.1 and Section 4.2.2, respectively.

4.2.1. Decision Tree

Because precisely predicting future access frequency is a tough undertaking, we anticipate the future access frequency will fall into four categories: high frequency and concentrated access hc, high frequency and discrete access hd, low frequency and concentrated access lc, and low frequency and discrete access ld to replace the precise access frequency to simplify it. Since the Decision Tree C4.5 algorithm can generate predictions quickly [29] and is suitable for making caching decisions, it is used in this paper to predict future access frequency categories. As a result, a decision tree technique is used in this paper to forecast future access categories.

- **Capturing the attributes.**
  Based on the study in the preceding Section 3.2, we created a decision tree model using the access frequency and access content

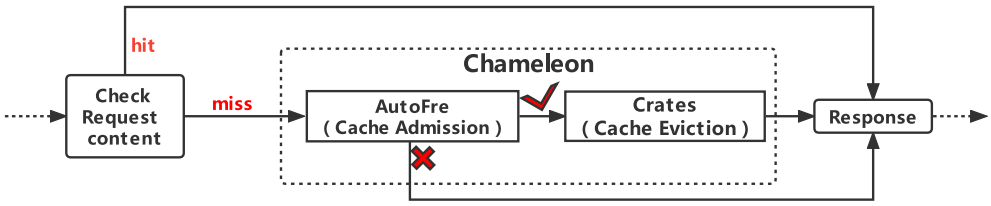


Fig. 5. The system architecture of *Chameleon*.

Algorithm 1 *AutoFre*

input: Training feature set:D; Historical frequency of access content: $rn_{c_{new}}$;
output: Admission Decision a

function DECISION TREE(D)
  Create Tree CT
  while $D \neq \emptyset$ **do**
    Compute the entropy of D by equation (1)
    for Iterate over all the attribute A **do**
      Compute the entropy of A by equation (3)
      Compute the information gain by equation (4)
      Compute $SplitInfo_A(D)$ of attribute A by equation (5).
      Compute $GainRatio(A)$ by equation (6)
    end for
    Find the node n with the max $GainRatio(A)$ by equation (7)
    Pop node n in D
    CT add the leaf node n
  end while
end function

function ADMISSION THRESHOLDS($rn_{c_{new}}$)
  Predict the future access frequency category f_t by CT
  Compute the admission thresholds r_i^t by equation (8)
  if $rn_{c_{new}} > r_i^t$ **then**
    $a = 1$
  else
    $a = 0$
  end if
  return a
end function

concentration obtained from the historical access information as attribute values of training data. However, these two criteria cannot predict future access frequency categories by adopting these attribute values. Furthermore, we append the attributes with a future access change trend attribute, which is the increase or decrease in access frequency during two continue periods. 1, −1, and 0 indicate the gradual increase, gradual reduction, and no change, respectively. It provides guarantees for predicting future access frequency categories. Therefore, the dataset for predicting future access frequency categories includes three attributes, access frequency, access content concentration, and trend, denoted by D.

- **Generation of the decision tree.**
  The main idea of the C4.5 algorithm is to build a decision tree by classifying instances in the dataset and the information entropy gain rate according to each attribute classification. The information entropy of the original dataset $Info(D)$ can be calculated by

$$Info(D) = -\sum_{i=1}^{M} \frac{|DC_i|}{|D|} log_2^{\frac{|D_i|}{|D|}}, \tag{1}$$

where D denotes a dataset composed of M samples and W category. $DC_i(1 \le i \le M)$ is the category sample set C_i in the dataset D. Let $N = |D|$ denotes the samples number in the dataset D, $|DCi|$ denotes the samples number of the subset DC_i for the category C_i. The information entropy $Info_A(D)$ is computed by the division of attribute A category. Due to the continuous nature of these attributes, we must discretize the access frequency and access number concentration before computing $InfoA(D)$. These attributes are divided into subsets by mapping into a few discrete data, and the formula is as follows:

$$f(A) = \lfloor A/10^k \rfloor * 10^k, \tag{2}$$

where k presents the magnitudes of A. Then $Info_A(D)$ can be obtained by

$$Info_A(D) = \sum_{j=1}^{V_A} \frac{|D_j|}{|D|} * Info(D_j), \tag{3}$$

where V_A is the number of subsets by the mapping equation. $D_j(1 \le j \le V_A)$ presents the jth partition subset, and $\frac{|D_j|}{|D|}$ is the proportion of the j*th* partition. Information gain of attribute A is the original information entropy minus the information entropy of attribute A. It can be presented by Eq. (4). The larger the value of information gains, the purer the division according to that category is. And then, we calculate the split information entropy of attribute A by Eq. (5). Lastly, the information gain rate $GainRatio(A)$ of the attribute A by Eq. (6). The above processes are repeated for all attributes until the final decision tree model is generated, and the primary operations are shown in function Decision Tree of Algorithm 1. A decision tree includes nodes and leaf nodes. The nodes of this paper's decision tree includes three attributes: access frequency, access content concentration, and trend. The leaf nodes present four predicted outcomes *hd, hc, ld, and lc*. It means the decision tree is four-level in this paper. Different values are formed based on the division of attributes as described before and operations such as mapping. One combination of these attribute values can be considered as an input. We can use the input to generate the corresponding prediction results (i.e., the categories of future access frequencies) by a decision tree model.

$$Gain(A) = Info(D) - Info_A(D) \tag{4}$$

$$SplitInfo_A(D) = -\sum_{j=1}^{V_A} \frac{|D_j|}{|D|} * log_2^{\frac{|D_j|}{|D|}} \tag{5}$$

$$GainRatio(A) = \frac{|Gain(A)|}{|SplitInfo_A(D)|} \tag{6}$$

$$\max_{i \in N} f(GainRatio(A_i)) \tag{7}$$

4.2.2. Admission threshold

Based on the steps and equations above, we obtain the prediction result of the future access frequency category f_t. It is essential for designing self-adaptive admission thresholds to consider not only the influencing factors of access frequency and concentration but also the size of the storage space and hit rate conditions. If the storage surplus size is larger than the current access content size in a period, we

Algorithm 2 *Crates*

input: A new content c_{new} is access, with size s_{new}; the surplus area in dynamic cache: r_s; each contents size: s_i; Core users accesses numbers sequence in time t: x_1^t; whole users accesses numbers sequence in time t: x_2^t.

1: **while** $r_s < s_{new}$ **do**
2: Calculate $cov(x_1^t, x_2^t)$ by equation (12) and obtain E
3: Get c_i by value v_t from E by equation (14)
4: $r_s = r_s + s_i$
5: remove c_i
6: **end while**
7: $r_s = r_s - s_{new}$
8: Storage c_{new} in dynamic cache.

can assume that all access contents are necessary for caching. We can assume that when the hit rate is close to 1 and more of the access contents are hit, these contents may be valuable and must be cached. Based on the above two assumptions, we design the formula for calculating the access threshold as shown in Eq. (8). If s_t is less than S, r_i^t is equal to 0. S represents the size of the entire storage space. s_t is the total size of the access contents for the current period by computing Eq. (10). When the current access content $rn_{c_{new}}$ has a history of being accessed more frequently than r_i^t, it is allowed to cache. The admission decision process is shown in the function Admission Thresholds of Algorithm 1.

$$r_i^t = \begin{cases} 0 & \text{if } S \geq s_t \\ [\frac{e^y - e^{-y}}{e^y + e^{-y}}] & \text{otherwise} \end{cases} \tag{8}$$

$$y = (1 - h_t) * f_t \tag{9}$$

$$s_t = \sum_{i=1}^{m} s_i \tag{10}$$

Combining the above two parts of predicting future access frequency categories and self-adaptive admission thresholds, we obtain the admission algorithm, named *AutoFre*, for access frequency-variability, as shown in algorithm 1.

4.3. Eviction algorithm: Crates

Another challenge is how to capture the relationship between popular content and specific user groups, and to improve the hit rate based on the relationship in edge servers. Through analysis above in Section 3.2, the specific user group with many features plays an important part during the low-frequency period. In the eviction algorithm, the principal component analysis determines the relationship between specific users and popular content, and the eviction content is chosen based on that relationship value in Section 4.3.1. On this basis, we propose a popular content protection mechanism in Section 4.3.2.

4.3.1. Principal component analysis

To describe specific user groups clearly, we denote the users (whole users) with high access frequency and broader distribution on the time dimension as *core users* U, and treat others as common users. It is well known that many users access the same content, which will become popular. The popularity of the content is gauged by the number of its accesses. On this basis, the relationship between popular content and core users can transform into the relationship between the access content number of core users and whole users, respectively. Therefore, we measure the relationship between popular content and core users by the content access number $X^t = [x_1^t, x_2^t]$, where x_1^t is an m-dimension vector of length m as a sliding window from core users in time t, is presented as

$$x_i^t = [r_1^{t-m*\Delta t}, r_2^{t-(m-1)*\Delta t}, \ldots, r_m^{t-\Delta t}], \tag{11}$$

each term represents the total number of user accesses per unit of time Δt. x_2^t is the same structure as x_1^t and represents the access number of whole users. We obtain the relationship between the access content number of core users and whole users' by

$$cov(x_1^t, x_2^t) = \frac{\sum_{i=1}^{n} (x_{1i}^t - \bar{x}_1^t)(x_{2i}^t - \bar{x}_2^t)}{n}, \tag{12}$$

where $\bar{x}_1^t$ and $\bar{x}_2^t$ are computed by

$$\bar{x}_i^t = \frac{\sum_{i=1}^{n} x_i^t}{n}. \tag{13}$$

The eigenvector with the largest eigenvalue in $cov(x_1^t, x_2^t)$ is the relationship value between the contents of core users and whole users. The larger an element in eigenvector E^t is the closer the relationship between popular content of core users and whole users is. We choose the immense value in the eigenvector E^t as the relationship between popular contents and core users in time t. Eviction content v_t is the lowest value of E cached in the dynamic cache area. Suppose the total size of evicting contents and the surplus area is smaller than the access one. In that case, we choose the smallest one in the remaining contents until the cache size of evicts is larger than the access content size. And then we pop these contents finally. (Line 1–8 in algorithm 2)

$$v_t = f(min(E)) \tag{14}$$

4.3.2. Popular contents protection mechanism

We can solve the problem by getting the relationship between core users and popular material to identify popular content. In addition, we find that there is still a tiny amount of popular content in each period that can easily be obtained by historical access frequency. To reduce the computation, these easily obtained contents cannot be used as the selected set for evicting contents. As a result, we suggest a strategy to protect those popular contents.

The main idea of the mechanism is to divide cache resources into dynamic and static cache areas dynamically. The existing intersection in popular content sets between two continuous unit times can regard as the future popular content. The dynamic cache area stores the access contents in real-time. The static cache area stores the intersection of popular contents without caching replacement. Since the intersection changes over time, we dynamically adjust the size of the static cache S_s^t by

$$S_s^t = \sum_{i=1}^{k^t} size(q_i^t), \tag{15}$$

where k^t is the number of protection popular contents in time $t-1$ and is calculated by Eq. (16). q_i^t presents the ith popular content in time $t-1$, where q_i^t is one of the protection popular content set q^t can be obtain in Eq. (17).

$$k^t = num(H_{t-1} \cap H_{t-2}) * a^t \tag{16}$$

$$q^t = Top_{k^t}(Access(H_{t-1} \cap H_{t-2})) \tag{17}$$

H_{t-1} and H_{t-2} are popular content sets from two continuous unit times.

Due to the content of the request being time-varying, the dynamic and static cache regions need to be automatically generated. So we design a coefficient a^t to implement it. It is generated based on hit rates r_s^{t-1} and r_d^{t-1} from dynamic and static cache areas. The coefficient a^t is calculated by

$$a^t = \frac{a^{t-1} * r_s^{t-1}}{a^{t-1} * r_s^t + (1 - a^{t-1}) * r_d^{t-1}}. \tag{18}$$

In other words, when a cache region has a higher hit rate, more areas are allocated to that region in the next update time. This approach can dynamically provide an appropriate static cache size S_s^t between two continuous unit times. Dynamic cache size S_d^t is the total size of the cache minus the static cache size S_s^t.

In the introduction above, we can gain a metric to measure the content's value by the relationship between popular content and core users and propose a protection mechanism to ensure the hit rate for a low access period. In other periods, we can choose the existing eviction policy and combine the protection popular contents mechanism as our eviction algorithm, which is called *Crates* and it is shown in algorithm 2. It is worth mentioning that, because the low-frequency access period is variable, we determine if it is in the low-frequency access period by predicting the access frequency category.

Algorithm 3 *Chameleon*

input: A new content c_{new} is access, with size s_{new}, user id u_{new} and time t_{new}; C contents $\{c_1, ..., c_i\}$ are already stored in the cache area, each size is s_i; Core users accesses numbers sequence x_1 and wholes' x_2 are cache.

if Size(historical traces)<Size(sliding window) **then**
  caching replacement by LRU.
else
  if Size(historical traces)=Size(sliding window) **then**
    Calculate the static cache size by equation (15)
    Calculate the intersection q_t by equation (17)
    cache q_t
  end if
  if c_{new} NOT in C **then**
    a = decision c_{new} whether admission or not by algorithm 1
    if $a = 1$ **then**
      remove eviction contents and caching c_{new} by algorithm 2
    end if
  end if
end if
$x_2^{t-\delta t} = x_2^{t-\delta t} + 1$
if u_{new} in U **then**
  $x_1^{t-\delta t} = x_1^{t-\delta t} + 1$
end if
if $t_{old} + \delta t = t_{new}$ **then**
  remove $x_1^{t-m*\delta t}$
  remove $x_2^{t-m*\delta t}$
  Calculate the static cache size by equation (15)
  Calculate the intersection q_t by equation (17)
  cache q_t
end if

4.4. Overall design: Chameleon

By integrating the admission (*AutoFre*) and eviction (*Crates*) algorithms, we obtain our cache replacement algorithm *Chameleon*. It mainly includes four phases: Initialization, Admission Decision, Eviction Decision, and Updating Information as follows:

- *Initialization.* When the length of access content information is inconsistent with the sliding window size, the cache replacement technique is LRU at first. When the size matches, we divided storage resources into dynamic and static cache areas and cached the intersection of popular contents between two continuous unit times in the static cache area based on the protection mechanism above. (Line 1–8 in algorithm 3)
- *Admission Decision.* With uninterrupted access, content statuses are different. When the status is miss, the future frequency level is gained. Based on the future access frequency predicted, the algorithm 1 decides whether allow the access content or not to cache. (Line 9–10 in algorithm 3)
- *Eviction Decision.* If the access content is admitted to the cache storage, the eviction content is selected and removed by algorithm 2. (Line 11–13 in algorithm 3)
- *Updating Information.* Access information is updated, and the contents and cache size of the static cache area through the sliding window periodically. (Line 16–26 in algorithm 3)

5. Evaluation

In this section, we evaluate our approach *Chameleon* by real traces from ChuangCache, show the results of applying *Chameleon* on them versus the existing representative policies, and analyze the results.

5.1. Experiment setting

Algorithms: We compare with three algorithms including *SecondHit-LFU*, *SecondHit-LRU*, and *SecondHit-Crates*.

- *SecondHit-LFU*: The admission policy is *SecondHit*, and the eviction policy is LFU. LFU removes the content object which has been cached for the smallest access frequency.
- *SecondHit-LRU*: The admission policy is *SecondHit*, and the eviction policy is LRU, which removes the least accessed content in the cache.
- *SecondHit-Crates*: The admission policy is *SecondHit*, and the eviction policy is *Crates*.

DataSet: The traces are from ChuangCache in China with 9,839,213 accesses in two days. Each trace item contains the timestamps, anonymized user ID, content ID, server ID, access size, URL, et al. We then deploy and evaluate *Chameleon* by comparing it with representative algorithms.

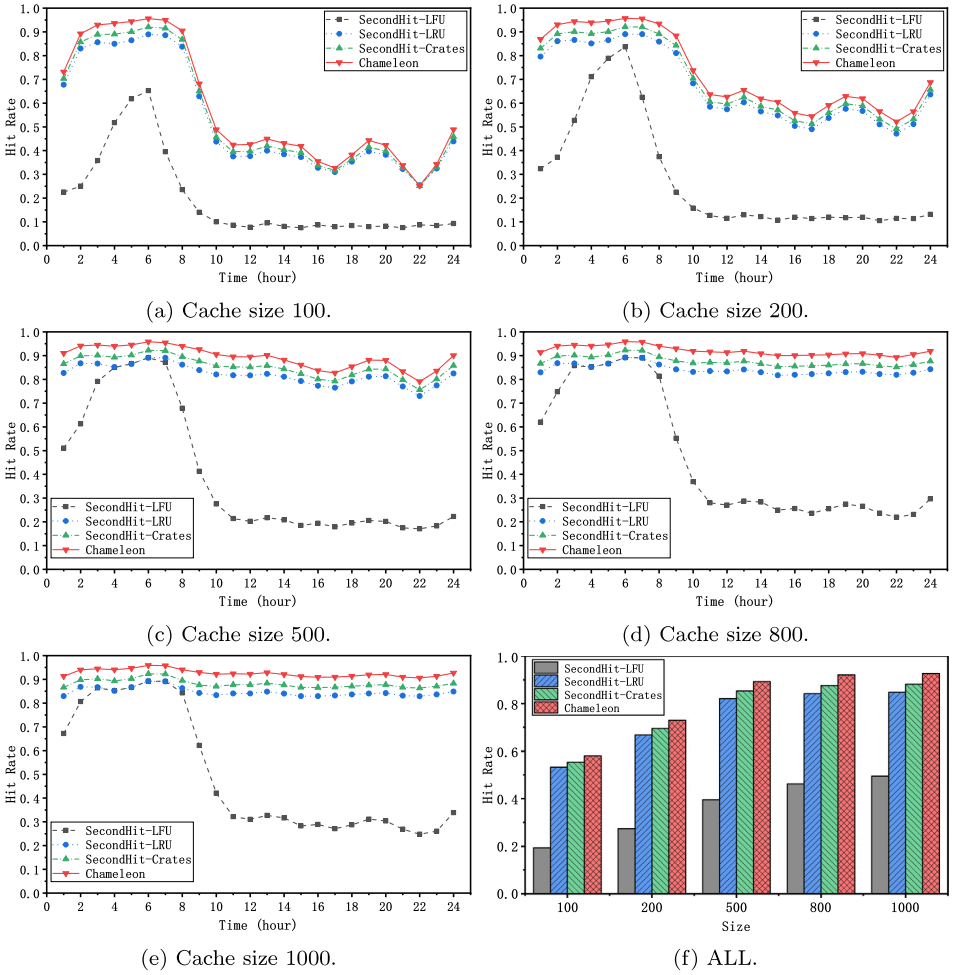
Parameter Settings: The sliding window size was set to 1 min, and δt was 1 s. Each access content size was 1. The core users were obtained from the previous day's traces in the same application.

5.2. Performance comparison

We conduct a series of experiments with different cache sizes to show the overall hit rate performance in Fig. 6. Fig. 6(a), 6(b), 6(c), 6(d), and 6(e) show that hit rates of each hour with cache size 100, 200, 500, 800 and 1000. Fig. 6 presents the average hit rate of 4 algorithms in different cache sizes. When cache size is small, the hit rate of our algorithm (red line) has a similar performance in high access frequency–time (such as the 17th and 22nd hour) but is superior to other times compared to the baselines. In particular, in Figs. 6(a) and 6(b), the frequency during low-frequency access is significantly higher than that during high-frequency periods. As the size of the cache grows, so does the hit rate. With the enlargement of the cache size, the performance of our algorithm becomes more prominent because more content can be cached in edge servers, so the hit rate increase as expected. The hit rate of our algorithm reaches about 98% during the 7th hour in Fig. 6(e), and the improvement is nearly 8%. From these figures, we can see that *Chameleon* exceeds the other three methods.

5.3. Result analysis

Fig. 7(a), 7(b), 7(c), 7(d), and 7(e) show that replacement rate of each hour with cache size 100, 200, 500, 800 and 1000. Fig. 7(f) presents the average of replacement rate from 4 algorithms in different cache size. When the cache size is 100, the replacement rate is higher than other cache sizes because the hit rate being lower than others. Our algorithm's replacement rate (red line) is nearly the *SecondHit-Crates*'s in some time, such as the 7th hour, but its hit rate is higher than the *SecondHit-Crates*'s. That means, sometimes the *SecondHit* is



(a) Cache size 100. (b) Cache size 200.

(c) Cache size 500. (d) Cache size 800.

(e) Cache size 1000. (f) ALL.

Fig. 6. [Performance comparison] The hit rate in different cache size. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

not fit as admission to improve the hit rate. As the cache size grows, the replacement rate of our algorithm gradually pulls away from the second eviction algorithm. It demonstrates that our algorithm can automatically adjust the cache access threshold as the hit rate and cache size grows. As you can see from the above graph, our algorithm increases access to some extent, but the overall hit rate is improved.

In summary, it is necessary to consider the time-varying characteristics of access frequency in different periods to improve the cache hit rate of edge servers in the cache replacement strategy.

6. Conclusion

To address the problem of low hit rate in ever-changing frequency scenarios, we analyze the real trace from ChuangCache. Based on the analysis, we propose a self-adaptive admission threshold in continue time with various access frequencies, including the decision tree to predict the access frequency category to support the admission policy. Then, we adopt the principal component analysis algorithm to analyze the relationship between popular content and core users for choosing the eviction content. Lastly, we propose a popular contents protection mechanism, which caches the popular contents from history in the static cache area, and cache & evict contents in real-time by the relationship between popular contents and core users in the surplus area. Through a series of experiments using real application trace data, we demonstrate that *Chameleon* reaches about 98% in caching hit rate and outperforms the *SecondHit-Crates* by 8%.

CRediT authorship contribution statement

Pengmiao Li: Research algorithm development, System architecture design, Analysis and/or interpretation of data, Validation, Writing – original draft, Writing – review & editing. **Yuchao Zhang:** Conception and design of study, Writing – review & editing, Funding acquisition. **Wendong Wang:** Conceptualization, Supervision, Funding acquisition. **Weiliang Meng:** Acquisition of data. **Yi Zheng:** Acquisition of data. **Ke Xu:** Writing – review & editing. **Zhili Zhang:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62172054, BUPT-ChuangCache Joint Laboratoryunder B2020009, the Key Project of Beijing Natural Science Foundation under M21030, the NSFC under Grant 62072047, and the National Key R&D Program of China under Grant 2019YFB1802603. The work of Pengmiao Li was supported in part by the BUPT Excellent Ph.D. Students Foundation under CX2019134.

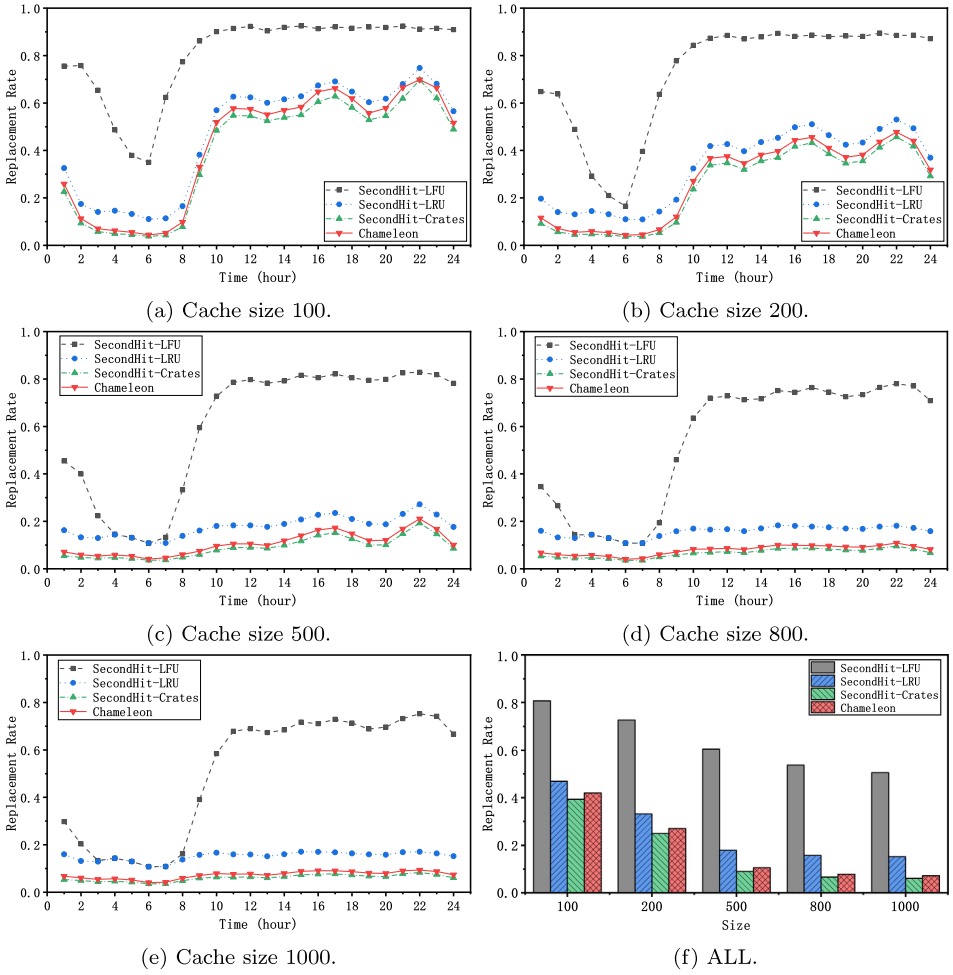


Fig. 7. [Result analysis] The replacement rate in different cache size. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

References

[1] We Are Social& Hootsuite, 2022. https://datareportal.com/reports/digital-2022-global-overview-report.

[2] Cisco, 2018. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[3] Nicola Blefari-Melazzi, Giuseppe Bianchi, Alberto Caponi, Andrea Detti, A general, tractable and accurate model for a cascade of LRU caches, IEEE Commun. Lett. 18 (5) (2014) 877–880, http://dx.doi.org/10.1109/LCOMM.2014.031414.132727.

[4] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, Chong-Sang Kim, On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies, in: Daniel A. Menascé, Carey Williamson (Eds.), Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Atlanta, Georgia, USA, May 1-4, 1999, ACM, 1999, pp. 134–143, http://dx.doi.org/10.1145/301453.301487.

[5] Martin F. Arlitt, Ludmila Cherkasova, John Dilley, Rich Friedrich, Tai Jin, Evaluating content management techniques for Web proxy caches, SIGMETRICS Perform. Eval. Rev. 27 (4) (2000) 3–11, http://dx.doi.org/10.1145/346000.346003.

[6] Akamai, 2022. https://www.akamai.com/.

[7] Mac Dowell Maggs, Ramesh K. Sitaraman, Algorithmic nuggets in content delivery, ACM SIGCOMM (2015).

[8] ChuangCache, 2022. https://www.chuangcache.com/.

[9] Gil Einziger, Roy Friedman, Ben Manes, TinyLFU: A highly efficient cache admission policy, ACM Trans. Storage 13 (4) (2017) 35:1–35:31, http://dx.doi.org/10.1145/3149371.

[10] Daniel S. Berger, Ramesh K. Sitaraman, Mor Harchol-Balter, AdaptSize: Orchestrating the hot object memory cache in a content delivery network, in: Aditya Akella, Jon Howell (Eds.), 14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017, USENIX Association, 2017, pp. 483–498, URL https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/berger.

[11] Nicaise Choungmo Fofack, Philippe Nain, Giovanni Neglia, Don Towsley, Analysis of TTL-based cache networks, in: Bruno Gaujal, Alain Jean-Marie, Eduard A. Jorswieck, Alexandre Seuret (Eds.), 6th International ICST Conference on Performance Evaluation Methodologies and Tools, Cargese, Corsica, France, October 9-12, 2012, ICST/IEEE, 2012, pp. 1–10, http://dx.doi.org/10.4108/valuetools.2012.250250.

[12] L. Cherkasova, Improving WWW proxies performance with greedy-dual-size-frequency caching policy, Hewlett-Packard Laboratories (1998).

[13] Suoheng Li, Jie Xu, Mihaela van der Schaar, Weiping Li, Popularity-driven content caching, in: 35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016, IEEE, 2016, pp. 1–9, http://dx.doi.org/10.1109/INFOCOM.2016.7524381.

[14] Qiang Li, Wennian Shi, Yong Xiao, Xiaohu Ge, Ashish Pandharipande, Content size-aware edge caching: A size-weighted popularity-based approach, in: IEEE Global Communications Conference, GLOBECOM 2018, Abu Dhabi, United Arab Emirates, December 9-13, 2018, IEEE, 2018, pp. 206–212, http://dx.doi.org/10.1109/GLOCOM.2018.8647794.

[15] Peng Yang, Ning Zhang, Shan Zhang, Li Yu, Junshan Zhang, Xuemin Shen, Content popularity prediction towards location-aware mobile edge caching, IEEE Trans. Multim. 21 (4) (2019) 915–929, http://dx.doi.org/10.1109/TMM.2018.2870521.

[16] Tongyu Zong, Chen Li, Yuanyuan Lei, Guangyu Li, Houwei Cao, Yong Liu, Cocktail edge caching: Ride dynamic trends of content popularity with ensemble learning, in: 40th IEEE Conference on Computer Communications, INFOCOM 2021, Vancouver, BC, Canada, May 10-13, 2021, IEEE, 2021, pp. 1–10, http://dx.doi.org/10.1109/INFOCOM42981.2021.9488910.

[17] Giuseppe Vietri, Liana V. Rodriguez, Wendy A. Martinez, Steven Lyons, Jason Liu, Raju Rangaswami, Ming Zhao, Giri Narasimhan, Driving cache replacement with ML-based lecar, in: Ashvin Goel, Nisha Talagala (Eds.), 10th USENIX

Workshop on Hot Topics in Storage and File Systems, HotStorage 2018, Boston, MA, USA, July 9-10, 2018, USENIX Association, 2018, URL https://www.usenix.org/conference/hotstorage18/presentation/vietri.

[18] Zhengxin Yu, Jia Hu, Geyong Min, Zi Wang, Wang Miao, Shancang Li, Privacy-preserving federated deep learning for cooperative hierarchical caching in fog computing, IEEE Internet Things J. (2021) 1, http://dx.doi.org/10.1109/JIOT.2021.3081480.

[19] Zhenyu Song, Daniel S. Berger, Kai Li, Wyatt Lloyd, Learning relaxed belady for content distribution network caching, in: Ranjita Bhagwan, George Porter (Eds.), 17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020, USENIX Association, 2020, pp. 529–544, URL https://www.usenix.org/conference/nsdi20/presentation/song.

[20] Ge Ma, Zhi Wang, Miao Zhang, Jiahui Ye, Minghua Chen, Wenwu Zhu, Understanding performance of edge content caching for mobile video streaming, IEEE J. Sel. Areas Commun. 35 (5) (2017) 1076–1089, http://dx.doi.org/10.1109/JSAC.2017.2680958.

[21] Yuchao Zhang, Pengmiao Li, Zhili Zhang, Bo Bai, Gong Zhang, Wendong Wang, Bo Lian, Ke Xu, AutoSight: Distributed edge caching in short video network, IEEE Netw. 34 (3) (2020) 194–199, http://dx.doi.org/10.1109/MNET.001.1900345.

[22] Marisa A. Vasconcelos, Jussara M. Almeida, Marcos André Gonçalves, Predicting the popularity of micro-reviews: A foursquare case study, Inform. Sci. 325 (2015) 355–374, http://dx.doi.org/10.1016/j.ins.2015.07.001.

[23] Zhiyi Tan, Wen Hu, Ya Zhang, Hao Ding, Online popularity prediction of video segments: Towards more efficient content delivery networks, in: 2019 IEEE Global Communications Conference, GLOBECOM 2019, Waikoloa, HI, USA, December 9-13, 2019, IEEE, 2019, pp. 1–6, http://dx.doi.org/10.1109/GLOBECOM38437.2019.9013162.

[24] Yuchao Zhang, Pengmiao Li, Zhili Zhang, Chaorui Zhang, Wendong Wang, Yishuang Ning, Bo Lian, GraphInf: A GCN-based popularity prediction system for short video networks, in: Wei-Shinn Ku, Yasuhiko Kanemasa, Mohamed Adel Serhani, Liang-Jie Zhang (Eds.), Web Services - ICWS 2020 - 27th International Conference, Held As Part of the Services Conference Federation, SCF 2020, Honolulu, HI, USA, September 18-20, 2020, Proceedings, in: Lecture Notes in Computer Science, 12406, Springer, 2020, pp. 61–76, http://dx.doi.org/10.1007/978-3-030-59618-7_5.

[25] Jie Liang, Dali Zhu, Haitao Liu, Heng Ping, Ting Li, Hangsheng Zhang, Liru Geng, Yinlong Liu, Multi-head attention based popularity prediction caching in social content-centric networking with mobile edge computing, IEEE Commun. Lett. 25 (2) (2021) 508–512, http://dx.doi.org/10.1109/LCOMM.2020.3030329.

[26] Saran Tarnoi, Wuttipong Kumwilaisak, Vorapong Suppakitpaisarn, Kensuke Fukuda, Yusheng Ji, Adaptive probabilistic caching technique for caching networks with dynamic content popularity, Comput. Commun. 139 (2019) 1–15, http://dx.doi.org/10.1016/j.comcom.2019.03.001.

[27] Zhengxin Yu, Jia Hu, Geyong Min, Zhiwei Zhao, Wang Miao, M. Shamim Hossain, Mobility-aware proactive edge caching for connected vehicles using federated learning, IEEE Trans. Intell. Transp. Syst. 22 (8) (2021) 5341–5351, http://dx.doi.org/10.1109/TITS.2020.3017474.

[28] Pengmiao Li, Yuchao Zhang, Huahai Zhang Wendong Wang, Ke Xu Zhili Zhang, CRATES : A cache replacement algorithm for access frequency-low period in edge server, in: 17th International Conference on Mobility, Sensing and Networking, MSN 2021, Exeter, UK, December 13-15, 2021, IEEE, 2021.

[29] Anis Cherfi, Kaouther Nouira, Ahmed Ferchichi, Very fast c4.5 decision tree algorithm, Appl. Artif. Intell. 32 (2) (2018) 119–137, http://dx.doi.org/10.1080/08839514.2018.1447479.