PrefCache: Edge Cache Admission With User Preference Learning for Video Content Distribution

Article *in* IEEE Transactions on Circuits and Systems for Video Technology · July 2020

DOI: 10.1109/TCSVT.2020.3006388

3 authors:

Yu Guan
Peking University
15 PUBLICATIONS **86** CITATIONS
SEE PROFILE

Xinggong Zhang
Peking University
69 PUBLICATIONS **834** CITATIONS
SEE PROFILE

Zongming Guo
Yantai Nanshan University
211 PUBLICATIONS **3,523** CITATIONS
SEE PROFILE

PrefCache: Edge Cache Admission with User Preference Learning for Video Content Distribution

Yu Guan, *Peking University,* Xinggong Zhang, *Peking University,* and Zongming Guo, *Peking University*

***Abstract*—With the deployment of video streaming in 4G/5G mobile network, Content Delivery Networks (CDN) are extending to the network edge to provide end-users better Quality of Experience (QoE). However, small cache size and irregular request patterns make it a great challenge for edge caching in video content distribution. Most of the existing cache policies are item-wise, they admit each video object separately, which performs poorly on the network edge due to irregular request patterns.**

We observe that compared with single video objects, users' preferences for video topics are much more constant, thus are easier to be predicted. So we propose PrefCache, a novel cache admission policy based on preference learning, for video content edge caching. PrefCache enables an edge cache to learn users' preferences for videos in real-time. Once receiving a video object, PrefCache decides whether to admit it to the cache by whether it is under users' preference.

We make three contributions in this work. (1) First, we design an information collector, which can proactively collect the preference-related information without any modification of clients and video providers. (2) Second, we propose a tree-structure model to learn and compress users' preferences. (3) Third, to decide which videos should be admitted to the cache in real-time, an explore-and-exploit method is applied. We carried out extensive experiments with 24 hours of trace data from a large commercial video content provider. The experimental results demonstrate that PrefCache can improve hit ratio up to 12%, and save 92% memory / 98% CPU overhead, compared to the state-of-the-art cache policies.

***Index Terms*—Edge caching, cache admission, video content delivery.**

I. Introduction

Over the last few years, network caching has become the dominant technology to improve the performance of video delivery, such as Content Delivery Networks (CDN) and proxy. As a part of Internet infrastructure, it has been quickly pushed from the data center to the network edge. Edge caching, such as 5G macro Based Station (BS) and Femto BS, promises to decrease back-haul overload and provides low-latency applications [7], [40]. A cache policy usually consists of two modules: (1) *cache admission policy* decides whether to admit a new object, and (2) *cache eviction policy* decides which existing object should be evicted when the cache storage is full. As pointed out by previous works [60], compared with the back-bone CDN, edge caching has two new characteristics [40], [6]:

- Small cache storage. Constrained by physical space and power consumption, the storage of an edge cache is much smaller than that of current CDN by several orders of magnitude.

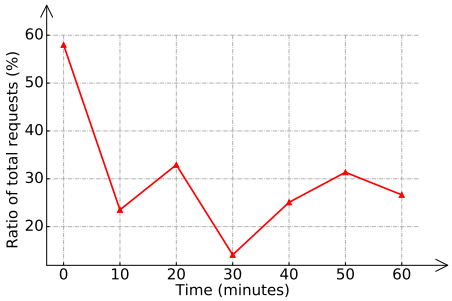


Fig. 1: Based on real request traces (Appendix A), we select the top 10% popular video objects and record their popularity in the following one hour. We measure their popularity by their request ratio of total requests. The result shows that the request arrivals of these videos are highly irregular and random.

- Irregular request pattern. Due to the small coverage region, the content popularity is full of fluctuations and bursts.

Existing cache policies, especially the cache admission part, are mainly designed for CDNs, most of them did not consider these new characteristics of edge caching [30]. In general, they work in an item-wise way: they analyze the request pattern for each single video object (*e.g.,* the time interval between two requests, the request frequency of a video object), and execute cache admission or eviction for each video object separately. Some approaches (*e.g.,* LRU [10], LFU [57], FIFO [9]) directly utilize the request pattern to make cache admission or replacement decisions. Other approaches ([58], [26], [31]) try to learn each video's popularity based on their request pattern, and then admit or replace videos based on their predicted popularity. However, due to the irregular request pattern of video objects on the network edge (Figure 1), both item-wise request pattern analysis and prediction are inaccurate, thus the performance of edge cache is still limited.

Cache admission by users' preference has been validated by many works. Researches [30], [24], [23] show that, although the request pattern of a single video object is highly irregular, the users' preference for video topics (*e.g.,* videos in the same category or the same author) are much more constant, thus is much easier to be predicted. So they try to learn users' preferences for videos. Then, whether a video is admitted to cache or not depends on whether it is under users' preference.

Currently, there are existing approaches of preference modeling in video caching ([30], [19], [24], [23]), but most of them are individual preference learning methods. They rely on a client-side logic to learn each end-user's preference, and

then an edge cache fetches the information from end-users. Two fatal issues limit their deployment in the real-world. First, not all users are willing to share their preference information due to privacy and ethical issue. Second, as far as we know, currently there is no technical support for an edge cache to fetch the preference information from an end-user in the video streaming process.

In this paper, we propose PrefCache, a novel cache admission policy, which utilizes a *common preference* learning model to improve the performance of an edge cache. With PrefCache, an edge cache learns the common preference of all users in its service area, rather than each user's individual preference, thus can work independently from end-users. First, all videos are grouped by some preference-related features (*e.g.,* categories, video authors, duration). Then, PrefCache learns the users' preferences for each group of videos by an unsupervised learning model. Finally, PrefCache admits videos based on whether the groups they belong to are under users' preferences. For the cache eviction part, we directly apply LRU policy due to its simplicity and efficiency.

We make three contributions in the design of PrefCache:

- First, to enable an edge cache to be aware of preference-related information of a given video object, PrefCache sets up a connection to video providers to proactively collect the information, which enables PrefCache to work independently from end-users (§IV).
- Second, to learn users' preference with a power-constrained edge node, PrefCache proposes a tree-structure learning algorithm to highly compress the preference model (§V).
- Third, to efficiently adjust the cache admission when users' preferences change, PrefCache presents a weighted-exploration admission algorithm. Once users' preferences change, PrefCache admits new groups of videos and evicts some old groups. In this process, PrefCache proposes a weighted reward function for each admission choice to avoid some choices that lead to a significant cache replacement. This enables PrefCache to keep high cache performance during the cache admission adjustment (§VI).

The evaluation shows that in the edge video caching scenario, PrefCache improves the cache hit ratio (the ratio between the number of requests being satisfied by cache and the number of all requests) by up to 12%, while requiring 92% less CPU overhead and 98% less memory overhead than state-of-the-art cache policies.

This paper is organized as the following. §II explains the problem of current edge caching and why we need a cache admission based on preference modeling. §III presents an overview of our system PrefCache. §IV presents our user-independent information collector. §V presents our model to learn users' preferences. §VI presents our algorithm to make cache admission decisions in real-time. §VII presents the implementation details. In §VIII we show the evaluation of the proposed PrefCache. We present some related work in §IX. In §X we discuss some limitations of our work. Finally, we conclude this work in §XI.

II. Motivation

A. Video caching on the network edge

At present, video delivery is the dominant traffic in the network [4]. In 2022, more than 82 % of network traffic will be video traffic, and this number will increase to 90 % by the year 2025. In video delivery, reducing network delay is one of the key factors of improving users' quality of experience (*e.g.,* significantly reduce the playback latency [41]). As a result, video caching is pushed from back-bone CDN node to edge network to provide end-users a low-delay video delivery [7], [40].

Compared with back-bone CDN nodes, the storage space of an edge cache is much smaller and the video traffic is highly irregular, which makes existing cache policies designed for CDNs inefficient on the network edge [30].

Pattern-based cache policies such as first in first out (FIFO) [9], least recently used (LRU) [10], least frequently used (LFU) [57] and their variants [51] have been widely deployed in the CDN caching, where there are abundant computing resources and they serve end-users in a large area. These methods directly utilize request patterns (*e.g.,* the time interval between two requests of the same object, the recent number of requests for an object) to decide whether to admit or evict a video object. However, they do not fit the edge network well due to irregular request patterns, and they may suffer major performance degradation caused by frequent cache replacement [30].

Recently, popularity learning approaches have been proposed in video object caching. They utilize the request patterns of video objects to learn their popularity, and then admit popular objects and evict unpopular ones. These methods fall into two categories: (1) popularity distribution learning approaches [25], [49], [20] assume that video popularity subjects to a uniform distribution, and then try to learn this distribution and explore the optimal caching decisions. (2) popularity trend learning approaches [14], [18], [47], [39] recognize that on the network edge, the popularity of each video is changing, but they assume that it is possible to learn the trend of popularity change for each video object. Then videos are admitted or evicted according to their predicted popularity. The above assumptions usually make sense in back-bone CDN caching, but they are not true for edge caching because the popularity of videos on the network edge is changing with high randomness [40], [6]. As a result, the video popularity they learned is inaccurate and their cache performance is still limited.

B. Users' preferences for single video objects and video topics

Among the existing solutions we presented in §II-A, one common design choice is that all those cache policies are item-wise: they analyze the request pattern for each single video object, then execute cache admission or replacement for each video object separately. Therefore, their weaknesses are the same: when the item-wise request pattern can not be well-modeled, their cache performance drops significantly.

Our intuition is that, although the popularity of a single video object can be highly irregular on edge, when we group

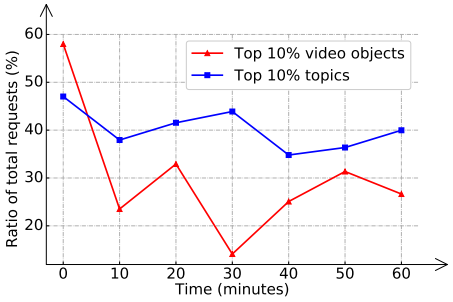


Fig. 2: Temporal popularity change of top 10% videos and top 10% video topics in 60 minutes.

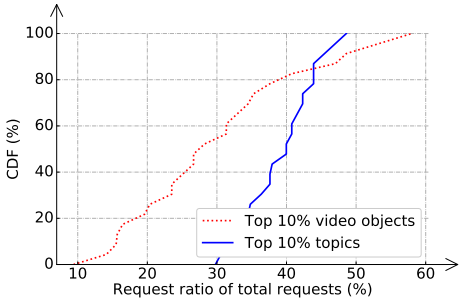


Fig. 3: CDF gram of popularity statistic of top 10% videos and top 10% video topics in 24 hours.

videos of similar topics, users' preference for the video groups is much more constant. For example, a user is watching a TV play one episode by one episode. When one episode is finished, the user may not watch it again, but it is most likely that she will continue watching the next episode. In this scenario, the video popularity of every single video is changing, but the video popularity of this set of TV play keeps constant. As a result, by clustering similar videos into one group, one can build a model to learn the users' preferences of each group of videos. Thus, a video is admitted or not depending on whether the group it belongs to is under users' preference.

To prove our intuition, we present our data analysis for users' preferences on single video objects and video topics. We collect some real video request traces of *Bilibili* [3], one of the biggest video providers in China. The data collection details are presented in Appendix A.

Key Observation: Users' preferences on video topics are much more constant than that on single videos. We compare the popularity persistence of video objects and videos grouped by topics. As shown in Figure 2, we pick 10% most popular videos at a moment. Then, we group videos by topics and then pick 10% most popular topics at the same moment. We record their temporal popularity change in the following 60 minutes. The popularity of the top 10% objects drops dramatically from 58% to only 12.6% in 30 minutes, while the popularity of the top 10% topics keeps much more constant, which is randomly floating between 34.6% and 47.2%.

In addition, we make a CDF gram of their popularity in the following 24 hours. Figure 3 shows that, the number of video requests for the top 10% topics keeps generally stable, which makes up 31% to 48% of total video requests. On the contrary, the video requests for the top 10% videos change dramatically, which makes up 10% to 58% of total video requests. Moreover, they become not-so-popular videos most of the time (their request ratio is below 40% in more than 80% of the time).

Based on the two facts above, we can conclude that the popularity of video topics is much more constant, thus is much easier to be predicted.

Insight: caching videos by preference groups achieves higher cache performance than caching videos by objects! After video grouping by preference-related features (*e.g.,* video topics), preference prediction for a video group is much easier than that for single video objects. Therefore, as a sharp difference with previous work, we want to design a cache policy based on users' preference modeling: we group videos by some preference-related features, learn users' preference of each video group, so as to cache them by groups, rather than by single video objects.

C. Challenges: Enabling an edge cache to learn users' preference

Currently, there are already some existing researches about end-users' preference modeling in cache policy design [30], [24]. However, most of these approaches learn users' individual preferences. They rely on a client-side logic to learn each end-user's preference, and then an edge cache fetches the information from end-users. As a result, the requirement of client-side logic modification and some privacy issues make them difficult to be deployed.

We argue that to enable an edge node to cache videos based on users' preference, it is important to build a preference learning model on the network edge, instead of on the client-side. An edge cache should be able to adaptively learn users' common preferences in its service area without any help from end-users. To achieve this, we face three new technical challenges:

- **How can an edge cache be aware of preference-related information of a video?** In our approach, preference-related information (*e.g.,* topics, video authors) of videos is necessary. On the one hand, when a video request arrives at the edge cache, its preference-related information is used to build our preference model. On the other hand, when a video object arrives, the edge cache decides whether to admit it based on its preference-related information. However, In current video streaming architecture, a video is requested only by HTTP requests. Preference-related information of the requested video is not provided by end-users. As a result, an edge cache should collect this information on its own.
- **How to learn users' preferences in real-time with a power-constrained edge node?** A preference model on the network edge takes inputs from many different users, whose preference may be distributed across a huge feature space. However, unlike a back-bone CDN node, an edge cache is usually power-constrained [40], [6]. How

Video Features			Under users' Preferences?	Admission
Category	Author	Duration		
Music Video	Alice	0-10min	Yes	✓
Music Video	Alice	10-30min	No	✗
Sports Video	Alice	0-10min	No	✗
Music Video	Bob	0-10min	No	✗
Sports Video	Bob	10-30min	No	✗

Fig. 4: An example of PrefCache.

to efficiently learn their preference in real-time among this huge feature space is a challenge.

- **How to efficiently explore the cache admission when users' preference changes?** Although users' preferences do not change with an extremely high frequency, we can not regard it as a constant. Adjusting the cache admission decisions based on the changing users' preference is a classical multi-arm bandit problem that can be solved by explore & exploit process [28]. However, exploring cache admission decisions on an edge cache leads to serious problems: when a cache decides to admit a different set of videos, it needs to first evict existing ones. This leads to serious I/O overhead and degradation of hit ratio during the cache replacement process, which is unacceptable for an edge device. How to minimize the cache replacement in the cache admission exploration is a challenge.

III. Overview

In this paper, we proposed PrefCache, a novel cache admission policy that utilizes users' preference for groups of videos to decide which videos can enter the cache storage.

An example of how PrefCache works: As shown in Figure 4, each video has some preference-related features such as *category* (*e.g.,* movies, cartoons, or sports videos, which represent the topic of a video), *author*, and *duration*. Suppose Alice is a famous singer, her short music videos are preferred by many users. Now a new music video of Alice arrives at an edge node. PrefCache checks its features and finds that this video's *category* is Music, its *author* is Alice, and the *duration* is short. According to PrefCache's preference model, the video group ("Music + 0-10min + Alice") it belongs to is under users' preference. Therefore, this video is admitted to enter the cache. Similarly, when a sports video from Bob arrives at the edge cache, it is not admitted to the cache because it is not preferred by most users.

To address the key challenges listed in §II-C, our system has three design choices (Figure 5):

- **Information collection by *User-Independent Information Collector* (§IV):** In most video providers, each video object is coupled with some tags (*e.g.,* category, author, duration) in its corresponding video information page, and these tags are highly related to users' preference. Therefore, we build a request listener on edge cache. Once receiving a video request, the edge node creates a paralleled request to fetch these tags from the video provider, instead of collecting them from users.

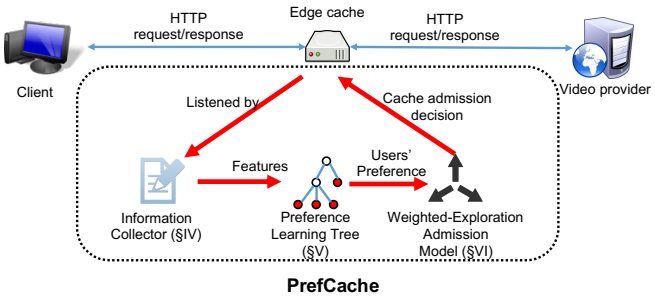


Fig. 5: System overview of PrefCache.

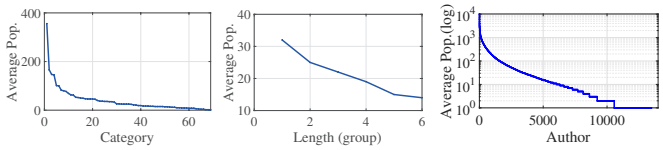
- **Preference modeling by *Preference Learning Tree* (§V):** Inspired by decision tree [50], we design a tree-structured model to learn users' preferences with high efficiency. Different from classical decision tree approaches, we design a dynamic node pruning and branching process, which enables an efficient online model compression.
- **Real-time cache admission by *Weighted-Exploration Admission Model* (§VI):** We present an explore & exploit method which dynamically admits the videos under users' preference. Different from traditional solutions, we design a new reward function to limit some exploration choices which cause a significant change of cache admission decisions, thus reduces the cache replacement in the learning process.

IV. PrefCache: User-Independent Information Collector

To admit videos based on users' preferences, the first problem is: given a video request or a video object, how can an edge node find its preference-related features. Based on current video streaming architectures (*e.g.,* DASH [54], HLS [1]), we can only get a video's technical information from their media description files, such as resolution, duration, available bitrate, the number of soundtracks. However, preference-related features (*e.g.,* author, topic) are not available.

To collect these preference-related features, existing approaches build an information collector to communicate with end-users. When end users send a video request, it also provides the necessary features coupled with the video (*e.g.,* category) and users (*e.g.,* age, gender). However, this requires the modification of client-side logic. They are difficult to be deployed in the real world due to compatibility and privacy problems [24].

Our intuition is that users' preferences can be well-expressed only by information about videos, which can be fetched from video providers. It provides PrefCache an opportunity to work independently from end-users. In this section, we show that there are available video features on the video provider, which can be utilized to learn users' preferences (§IV-A). Then we present our process for an edge cache to proactively collect them (§IV-B).



(a) Categories. (b) Duration. (c) Authors.

Fig. 6: Users' preferences for videos grouped by different features. We apply popularity (Pop. for short in the figures) to represent users' average preferences to videos in a group, which is calculated by the number of requests to videos in the group divided by the number of different videos in the group.

Fig. 7: A piece of HTML file from a video information page on Youtube.

A. Available preference-related features on video providers

For almost all video providers, given a video, there are some tags on the video page, representing some features of the video. For example, in Youtube [5], each video has information like uploading date, author, duration, times of view in history, the number of comments, subscribers, likes, and dislikes on the video information page. In Bilibili [3], other than the above tags, there are some more tags that can reflect the category of the video, such as games, education, music.

Figure 6 shows that these tags are highly related to users' preferences. We use the same dataset as that in §II in this analysis. In Figure 6(a), we group all video objects by categories. We can see that in the most popular category, a video object has 356 requests in our traces on average, while in the least popular category, a video has only two requests on average. Also, the video author is an important factor of users' preferences. The most preferred video author has over 10^4 requests for each video, while the most unpreferred author has only one. Similar conclusions can be derived from the result of video duration.

Given a video request, we can find these preference-related features in the following steps:

- First, there is a simple relationship between the URL of the video object and the video information page. For example, on Youtube, a video has the following URL:

  https://youtu.be/qbrKzpwZSRo

  With a simple URL transformation, its corresponding information page (an HTML file) can be fetched by requesting a similar URL:

  https://www.youtube.com/watch?v=qbrKzpwZSRo
- Second, on the video information page, the HTML file, all the preference-related features can be found. With a keyword extraction, it is possible to get all of them, as shown in Figure 7.

B. Process flow of information collector

With the above insights, we design our information collector which fetches the preference-related features from the remote video provider. It works in the following process:

- A listener is set in the edge cache. Once the edge cache receives a video request, the listener generates a copy of the requested URL.
- Based on a string transformation, a request sender generates the URL of the corresponding video information page, and then sends the HTTP request for this video page to the video provider.
- When the video page is back, a keyword extractor gets all the video features related to this video.

The design of our information collector has two benefits. (1) Since the size of a video information page (usually less than 100 KB, we put a sample in [2]) is much smaller than that of a video object (usually 100 MB to 3000 MB), this method does not produce more network traffics. (2) Our method is transparent to both end-users and video providers. For end-users, they request a video just by sending HTTP requests, which is the same as they request a video without our architecture. For a video provider, it needs nothing more than a web server which can response HTTP requests. Therefore, our method is light-weight and naturally compatible with existing video streaming architectures.

V. PrefCache: Real-time preference learning by Preference Learning Tree

In this section, we present the design of our Preference Learning Tree (PLT). PLT utilizes some content features collected by Information Collector (§IV) to represent users' preference for videos (we call them preference-related features), and learns which features are under users' preference. First, we introduce how PLT learns users' preferences (§V-A). Then, we introduce our method to compress the huge feature space (§V-B).

A. Learn user preference by PLT

PLT proposes a tree model to group similar videos by preference-related features, and learn users' preferences for each group of videos. PLT utilizes three features: (1) category, (2) author, and (3) video duration. An example of PLT is shown in Figure 8. PLT is a tree with four levels. Level 0 (root) represents the video service provider. Level 1, 2, and 3 group videos with different categories, authors, and durations. In this tree, each node represents a feature combination. Especially, each *leaf node* (we mark them in red in Figure 8) represents a group of videos.

For each node i (which corresponds to a feature combination), three key values are maintained as the node information: (1) the number of requests for videos under this feature combination in current time-slot t, denoted by $f_{i,t}$, (2) the number of cache-hit requests (the requested video is locally stored in the edge cache) for videos under this feature combination in current time-slot t, denoted by $h_{i,t}$, and (3) average hit ratio of all videos under this feature combination in current time-slot t, denoted by $r_{i,t}$.

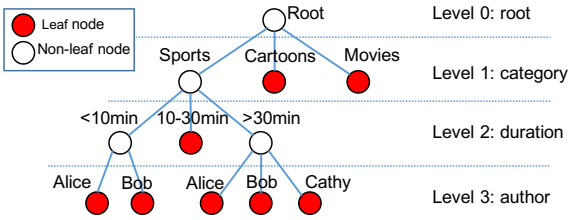


Fig. 8: Structure of PLT.

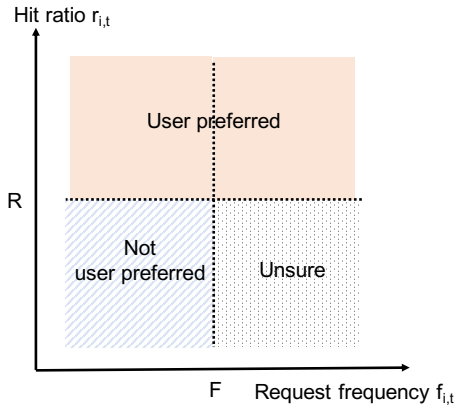


Fig. 9: Definition of whether a group of videos is under users' preference.

For each node i, the above information is updated using a moving average algorithm [21]. When a new time-slot $t+1$ arrives, the value of $f_{i,t+1}$ is updated as the weighted average of two parts: the previous request frequency $f_{i,t}$, and newly increased requests in the new time-slot $\Delta f_{i,t+1}$:

$$f_{i,t+1} = \alpha f_{i,t} + \beta \Delta f_{i,t+1} \tag{1}$$

Similarly, we update the value of $h_{i,t+1}$ by:

$$h_{i,t+1} = \alpha h_{i,t} + \beta \Delta h_{i,t+1} \tag{2}$$

Then, $r_{i,t+1}$ can be computed by the ratio of cache-hit requests to all requests:

$$r_{i,t+1} = \frac{h_{i,t+1}}{f_{i,t+1}} \tag{3}$$

We notice that when computing $f_{i,t+1}$ and $h_{i,t+1}$, there exists a trade-off between sensitivity and smoothness when users' preferences change, and this trade-off depends on the value of α and β. When the value of α is larger, the value of $f_{i,t+1}$ and $h_{i,t+1}$ react slowly when the users' preferences are actually changed. When the value of β is larger, they are more sensitive to preference change, but it may suffer from bursts and random fluctuations. In this paper, α is set to 0.33, and β is set to 0.67, which is a common setting to keep a balance of sensitivity and smoothness. The duration of each time-slot is defined as 30 minutes.

Finally, given the above information of a *leaf node* i in PLT (which corresponds to a video group), we can judge whether videos in this group are under users' preference (Figure 9). Generally, there are three cases:

- **User-preferred:** $r_{i,t} \geq R$**.** A high hit ratio indicates that the same videos are requested by users many times, thus it is reasonable to believe that most videos in this group are preferred by users.
- **Not user-preferred:** $r_{i,t} < R$ **and** $f_{i,t} < F$**.** Low hit ratio and request frequency indicate that few people are interested in videos in this group.
- **Unsure:** $r_{i,t} < R$ **and** $f_{i,t} \geq F$**.** In this case, videos in this group are requested many times but the overall hit ratio is limited. This indicates that in this group, some videos are really preferred by users, but others are not. Therefore, users' preference for this group of videos is unknown.

where R and F are two constants that denote the threshold of users' preference. The parameter settings of these two constants will be discussed in §VIII-E. Moreover, we further discuss these three cases in §X.

B. PLT branching and pruning

In the real-world deployment on a power-constrained edge node, the huge number of feature combinations make the size of PLT unscalable. According to our investigation, *Bilibili* has 69 different categories and more than 1.5×10^8 active authors in 2017. To express all the feature combinations, PLT needs more than 10^{10} nodes. Maintaining and updating the information (*e.g.,* cache hit ratio, request frequency) for such a number of nodes leads to huge overhead. Therefore, we need to provide this tree model the ability to compress the huge number of feature combinations.

Fortunately, we do not need to branch all nodes in PLT to level 3: PLT can be designed as an incomplete tree. For example, suppose node "Sports + 10-30min" is under users' preference according to our definition in §V-A, then all videos with this feature combination are considered user-preferred videos. Therefore, "Sports + 10-30min" already expresses users' preferences accurately. We do not need to branch this node to the author level since it does not matter whether the video comes from Alive or Bob.

We present our real-time PLT compression algorithm, which only branches a node to the next level *when users' preference is unsure*, which corresponding to the third case we listed in §V-A. In the beginning, the root is the only node in PLT. PLT compression consists of only two rules: (a) *node branching* and (b) *node pruning*.

- *Node branching:* For a node i in PLT, if the users' preference on it is unsure (the third case in §V-A) and the node i does not have child nodes, then i is expanded to the next level.
- *Node pruning:* For a node i in PLT, if the users' preference on it is known (first two cases in §V-A) and i has child nodes, then all child nodes of i are deleted.

These two rules are executed for each node in PLT periodically. In this way, PLT compresses the original 10^{10} video groups to only 800 to 1900 video groups (see our discussion in §VIII-E).

VI. PrefCache: Real-time cache admission by Weighted-Exploration Admission Model

In PrefCache's cache admission phase, given the users' current preference for each group of videos we learned by PLT, PrefCache is able to admit video objects under user-preferred groups. However, although users' preference does not change dramatically, we can not regard it as a constant. Directly admitting videos under users' current preferences may lead to cache performance degradation [33]. Moreover, to accurately compute users' preference for all groups of videos, we need to admit them all into the cache to update the information (*e.g.,* hit ratio) of corresponding nodes in PLT.

In this section, we present how existing approaches solve the problem, as well as issues they have (§VI-A). Then we present our Weighted-Exploration Admission Model (WEAM) to explore the potential users' preference in real-time (§VI-B).

A. Cache replacement problem in user preference exploration

In our cache admission scenario, given N video groups, we need to choose K out of N groups and admit them to the cache. Since users' preference for video groups are changing, the cache admission choices should be adjusted in real-time. This has been well-modeled as the classical multi-armed bandit problem [8], where each group is an arm of the bandit and the cache hit ratio is the reward. In its basic form, top-K groups are selected by an explore-exploit process, where a fraction of groups is chosen to explore different options and the rest exploit the best decision. To optimize the performance in a dynamic network environment, some advanced methods like **UCB1** [12] are proposed which does not require explicitly specifying the fraction of samples for exploration. Instead, it transparently combines both the exploration as well as the exploitation decisions.

Unfortunately, these classical methods lead to serious cache replacement problem during the exploration. This usually happens *when the exploration process decides to choose a video group that contains a huge number of video objects*. For example, once PrefCache decides to explore the "movie" group, the cache will admit all movie videos. Naturally, the group "movie" includes a lot of different video objects, and not all movies are under users' preference. After a short period of time, PLT will find that not all movie videos are under users' preference, and PrefCache will make another admission decision. In this process, most video objects in the movie category are first admitted and then evicted, leading to a serious cache replacement.

Our insight to solve the problem is that, in the exploration phase, we should give priority to video groups which contain a smaller number of different video objects. In PLT tree, a node of high depth (*e.g.,* "movie + Alice + 10-30min" is a level-3 node) usually contains fewer different videos than a node of low depth (*e.g.,* "movie" is a level-1 node). As a result, the depth in PLT can be regarded as an estimation of the number of video objects included in a group. We can utilize this information to weight the priority of each video group in the explore & exploit process.

B. Weighted-Exploration Admission Model

We present Weighted-Exploration Admission Model (WEAM), a new solution for the multi-arm bandit problem, especially for the edge caching scenario. WEAM computes a reward for each video group. Values of rewards are weighted by not only the expected hit ratio of videos in the group which represents users' preference, but also the depth of the corresponding node in PLT which represents the number of videos in the group. Then, WEAM decides whether to admit a video group to the cache according to its reward.

WEAM is executed at the beginning of each time-slot. It has two main steps:

1. Reward computation for each video group. Given a leaf node i (corresponding to a video group) in PLT, $h_{i,t}$ is the recent hit ratio of videos under this group (we defined in §V), then we define l_i as the depth of node i in PLT, and define c_i as the number of times i has been admitted in history. Then we define the reward function $F(h_{i,t}, l_i, c_i)$ of i as the following:

$$F(h_{i,t}, l_i, c_i) = \frac{\log(h_{i,t} + \delta_1)\log(l_i + \delta_2)}{\log(c_i + \delta_3)} \tag{4}$$

where δ_1, δ_2 and δ_3 are three constant offsets. Notice that if a leaf node i has never been chosen before, PrefCache sets its hit ratio h_i to its parent's hit ratio by default. Based on this reward function, WEAM prefers to admit videos group with higher hit ratio, fewer times of being admitted before, and higher depth in PLT. Compared with classic explore & exploit process which only considers each group's hit ratio and the times of being chosen before, in WEAM, depth of a node in PLT is considered as a new factor of the reward function. For example, "sports + <10min + Alice" has a higher priority for exploration than "movies". This prevents WEAM from admitting a huge group of videos at one time, thus prevent the potential cache replacement.

2. Admitting video groups with the highest reward. WEAM chooses K video groups *with the highest rewards defined* in equation (4), and admit them to enter the cache.

The duration of each time-slot is set to 30 minutes, the same as that of PLT (§V).

VII. Implementation: PrefCache workflow

We implement PrefCache based on C++ without requiring any extensions. Our implementation is publicly available with open access at its GitHub repository [2].

PrefCache is implemented as a finite-state machine which has four stages (Figure 10):

Stage 0: Initialization. In the initialization phase, PrefCache needs to set up the initial settings for both PLT and WEAM. For PLT, the root is the only node of the tree, representing the only one group which contains all videos of the video provider. Node information like request frequency, times of cache hit are all set to zero. For WEAM, it will admit the group contains all videos to enter the cache.

Stage 1: PLT update. After initialization, PLT checks each node in the tree with the branching rule and the pruning rule (§V). Then PLT updates the set of all leaf nodes, each of them representing a group of videos.

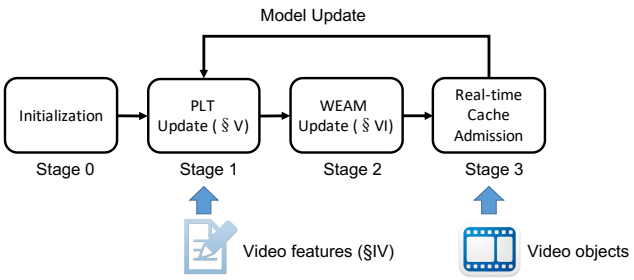


Fig. 10: Four-stage workflow of PrefCache.

Stage 2: WEAM update. Once PLT finishes an update, WEAM will update its admission decision. Given all groups of videos provided by PLT, WEAM chooses K of them. All videos under these nodes will be admitted to the cache, others are not. (§VI)

Stage 3: Cache admission while waiting for the next time-slot. Finally, we provide real-time cache admission to each video object, by simply checking if its feature combination is chosen by PrefCache. These operations are simple and can be executed very fast. When the next time-slot arrives, PrefCache goes back to Stage 1 to update PLT (and then goes to Stage 2 to update WEAM). In practice, the duration of one time-slot is set to 30 minutes.

VIII. Performance Evaluation

In this section, we show that:

- In simulated request traces, PrefCache improves the cache hit ratio by 1.4% to 12%.
- In real request traces, PrefCache improves the cache hit ratio by 2.5% to 11.6%.
- PrefCache requires 92% less computational and 98% less memory overhead than state-of-the-art learning-based cache algorithms.
- The parameter settings of PrefCache can be different to adapt to different scenarios. And we discuss the principle of edge caching according to our parameter optimization.

A. Methodology

We build a test-bed to evaluate the performance of an edge cache with different admission strategies. In our scenario, all requests are video requests, and they will be all sent to this edge node. If a cache hit happens, the request will be satisfied immediately. If a cache miss happens, the request will be forwarded to the video provider, and the file is sent back with the same path, passing by the edge caching node. The edge cache node decides whether to admit the video object based on its admission policy.

Baselines: We compare PrefCache with two caching policies: (1) Second hit [44], a frequency-based caching policy, which admits a video object when the video arrives at the cache at the second time. (2) RL-Cache [33], a state-of-the-art deep-learning-based cache policy, which is proposed in 2019[1]. This approach applies a reinforcement learning algorithm to learn the probability of cache admission to each video object based on the recent request history. All the above methods apply classic LRU as the eviction policy. To make a fair comparison, we apply LRU as the eviction policy for PrefCache as well. To illustrate our potential gain, we also evaluate the performance of pure LRU policy (which admits all video objects to enter the cache, only running an LRU replacement) as a reference point. Notice that we mention several cache policies based on individual preference learning. The performance of these algorithms highly depends on users' private information (like gender, age). However, as far as we know, there is no technical way to fetch this information by an edge cache. So they are not evaluated in this comparison.

Simulated request traces: Six simulated request traces are generated following Zipf distribution: $p_i = \frac{c}{i^\alpha}$, where p_i is the popularity of the ith most popular video, c is a constant. We set the value of α to six different values $(0.2, 0.4, 0.6, 0.8, 1.0, 1.2)$ for the corresponding six traces. Each trace contains 10^7 requests to 10^5 different videos.

Real request traces: We collected 24 hours of HTTP request traces from *Bilibili* (including 1.5×10^6 requests to over 2.5×10^4 different videos). Details of the request collection are presented in §A. The request traces are available at [2].

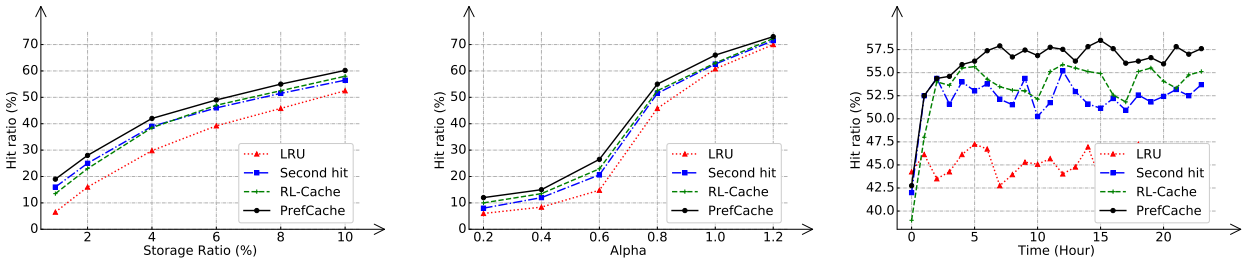
Default experiment settings: In our evaluation, we test PrefCache's performance under different scenarios, including different *storage ratio* (the ratio of cache storage and the total size of all videos) and *request distribution* (Zipf distribution with different α). In each experiment, we change one dimension of setting and keep another dimension as the default setting. The default storage ratio is 8%. In our simulated experiments, the default request distribution is a Zipf distribution with $\alpha = 0.8$.

PrefCache parameter settings: We present the parameter settings for §VIII-B, §VIII-C, and §VIII-D here. For the PLT training phase (§V-A), suppose F_0 is the request frequency of all videos in the current time-slot, we set $F = 0.01F_0$, and set $R = 60\%$. For the parameter K in the admission phase (§VI-B), basically when the value increases, more videos will be admitted to the cache but more cache replacements will happen. The value of K is optimal when the admitted videos can fulfill the cache space without causing frequent cache replacement. We set $K = 10\%$ according to our practice. As for the formula (4) in §VI-B, we set $\delta_1 = 0.1$, $\delta_2 = 1$ and $\delta_3 = 1$. All the above settings will be discussed in §VIII-E.

B. Hit-ratio under simulated trace

Hit ratio v.s. cache storage ratio: Figure 11(a) shows the performance comparison under different storage ratio (the ratio of cache storage and the total size of all videos). PrefCache

[1]RL-Cache is an algorithm which requires extremely powerful devices (*e.g.*, the authors evaluate its performance based on a computer with a 16-cores CPU and a 3584-cores GPU). It is not practical to work on an edge or ad-hoc device. Fortunately, we find that their training process can be highly optimized. We re-implement their algorithm and apply K-means [27] clustering model instead of back-propagation [35] model in the parameter training step. This significantly reduces the device requirement and get even better performance than its original implementation. Our implementation can be found in [2].

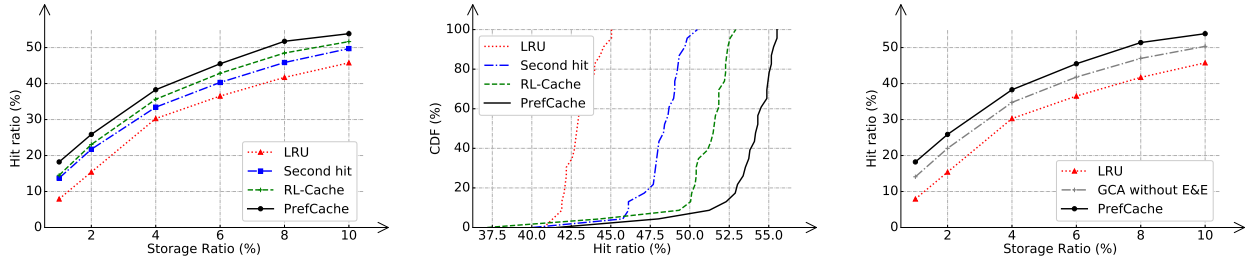


(a) Hit ratio v.s. cache storage ratio under simulated request traces (Video requests follow a Zipf distribution with $\alpha = 0.8$).

(b) Hit ratio v.s. different density of the request distribution. We adjust the request distribution by modifying the Zipf parameter α (Cache storage ratio = 8%).

(c) The 24-hour hit ratio under our simulated request traces. (Video requests follow a Zipf distribution with $\alpha = 0.8$, cache storage ratio = 8%)

Fig. 11: Performance comparison of LRU, Second Hit, RL-Cache and PrefCache under simulated request traces.



(a) Hit ratio v.s. cache storage ratio under the real request traces.

(b) The CDF of hit ratio of 24-hour real request traces. (cache storage ratio = 8%)

(c) A component-wise analysis of PrefCache's improvement (cache storage ratio = 8%).

Fig. 12: Performance comparison of LRU, Second Hit, RL-Cache and PrefCache under real request traces.

performs consistently better than other cache policies, improving the average cache hit ratio by 2.4% compared with RL-Cache, by 2.5% compared with Second Hit and by 10.4% compared with LRU.

Hit ratio v.s. the density of popularity distribution: Figure 11(b) shows the performance comparison under different request distribution. Under a Zipf distribution $p_i = \frac{c}{i^\alpha}$ (we defined in §VIII-A), when the alpha is large (a dense request distribution), these four algorithms perform roughly the same. PrefCache out-performs other admission policies when the request distribution is sparse (*e.g.,* $\alpha \in [0.2, 0.8]$), just like the real-world request distribution on the network edge [40], [6]. The performance of RL-Cache is on par with that of Second Hit, and LRU is the worst.

Hit ratio v.s. time: Figure 11(c) compares the cache hit ratio under the 24-hour simulated request traces. Among all algorithms, LRU performs similarly all the time and the other three algorithms have a very short learning process. This is because a short period is required for Second hit to warm up the BloomFilter table [52], for RL-Cache to learn the parameters of cache admission, and for PrefCache to learn the users' preference. After the learning process, all admission policies arrive at a stable performance level.

We also notice that given an arrival order of object requests, it is possible to simulate different request arriving rates. Fortunately, all cache policies we compared are event-driven, not time-driven. Therefore, a cache admission or a cache eviction only happens when a new video object arrives. As a result, when the order of request arrival is fixed, the behavior of a cache is fixed, regardless of whether all requests arrive within 1 minute or 10 minutes. Therefore, basically the arriving rate does not influence the hit ratio.

C. Hit-ratio under real trace

Hit ratio v.s. cache storage ratio: Figure 12(a) shows the performance comparison under different cache storage. Compared with the result in Figure 11(a), the hit ratio for all the compared algorithms becomes slightly lower. The reason is that on the network edge, the request pattern becomes irregular and dynamic. So the video popularity is more difficult for algorithms to learn.

Hit ratio v.s. time: Figure 12(b) shows the CDF gram of the hit ratio under 24-hour real-world request traces. PrefCache reaches 2.5% higher hit ratio than RL-Cache, 6% higher than Second Hit and 11.6% higher than LRU. We also notice that RL-Cache performs better than Second hit (which is on par with Second hit in our simulated traces) because the users' request frequency becomes more dynamic, traditional frequency-based admission approaches are not able to accurately estimate the popularity of videos. According to the evaluation in [33], RL-Cache has outstanding performance in web content caching scenario, but its performance gain is relatively smaller in video caching scenario, compared with AdmitAll and Second Hit, especially in small cache storage. This result is consistent with our evaluation.

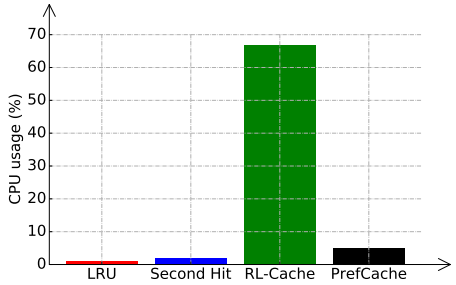


Fig. 13: Comparison of average CPU occupancy when running cache admission policies.

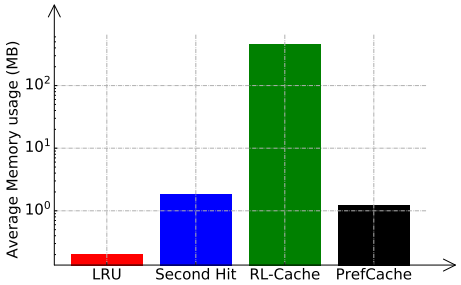


Fig. 14: Comparison of average memory usage when running cache admission policies.

Component-wise improvement: Figure 12(c) runs a component-wise analysis to evaluate the contribution of each technique in PrefCache by adding one of them at a time to the LRU baseline (with LRU eviction policy). Conceptually, one can get to PrefCache from the baseline in two steps.

First, we add our proposed user preference modeling (§V), and directly admitting the top $K\%$ video groups (without the explore & exploit process we present in §VI). This improves the cache hit ratio by 4.8% to 7.1%.

Then, we add our proposed explore & exploit based cache admission policy (§VI). This further improves the cache hit ratio by 3.0% to 4.5%.

D. System Overhead

Next, we examine the overheads of LRU, Second Hit, RL-Cache, and PrefCache. Here, we use a laptop to perform as an edge cache (MacOS Mojave version 10.14.5, 2.7 GHz Intel Core i5 CPU, 16 GB 1867 MHz DDR3 Memory, Intel Iris Graphics 6100 1536 MB).

CPU overhead: Figure 13 shows the CPU overhead of four algorithms. LRU is the most efficient one because it does not need to do any extra computation other than admitting all videos and executing LRU replacement. Second Hit requires a real-time update to the BloomFilter table. PrefCache is a learning-based algorithm but it proposes a light-weight PLT and WEAM training process. RL-Cache is a Monte-Carlo-based algorithm [46] which requires a lot of sampling and computation.

Memory overhead: Figure 14 shows the memory overhead of four algorithms. LRU only needs to record the recency of each video object in the cache. Second Hit requires a BloomFilter table to record the request frequency for each video object (regardless of whether it is currently in the cache or not). PrefCache's storage overhead mainly comes from PLT. Since PLT compress the feature combinations to a limited number of leaf nodes, it does not require much memory to store the model. RL-Cache requires significantly higher memory allocation for its admission model.

From the above overhead evaluation, we can see that although PrefCache is a learning-based cache policy, its computation and storage overhead is just on par with classic algorithms. It is much more deployable on the network edge than existing learning-based algorithms.

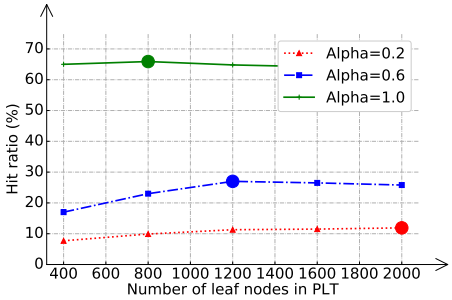


Fig. 15: The optimal number of leaf nodes in PLT (which can achieve the highest cache hit ratio) under Zipf distribution with different values of α. We highlight the optimal number of leaf nodes in each case. Cache storage ratio $= 8\%$

E. Parameter analysis of PrefCache

Now, as the final part of our evaluation, we present how parameters of PrefCache influence the cache performance.

The optimal number of leaf nodes in PLT: As is shown in §V, the number of leaf nodes in PLT can be controlled by two parameters: R and F. By decreasing the value of F and increasing the value of R, the node branching can be more aggressive and the node pruning can be more defensive, thus the number of leaf nodes is increased. Similarly, adapting these parameters in another direction, we can decrease the number of leaf nodes. A PLT with inadequate leaf nodes is not enough to express users' preference, but too many leaf nodes may lead to over-fitting of users' preference. Both situations degrade the performance of PrefCache.

Figure 15 shows the optimal number of leaf nodes with different request distributions. A dense request distribution indicates that users' preference falls in a limit number of videos, so their preference is easy to be modeled. On the contrary, a sparse request distribution indicates that users' preference is difficult to model, which requires a large model size.

The order of PLT levels: In all the performance evaluation parts (§VIII-B, §VIII-C, and §VIII-D), PLT is structured as the following order: level 1 - category, level 2 - duration and level 3- authors. Now we also try all the other orders.

Table I shows the number of leaf nodes and the hit ratio under different orders of video grouping. We keep the value of R and F as default (§VIII-A). The result shows that the order generally does not matter with the cache performance,

L1	L2	L3	Leaf nodes	Hit ratio (%)
Category	Duration	Author	832	52.4
Category	Author	Duration	905	52.3
Duration	Category	Author	982	52.3
Duration	Author	Category	1839	51.8
Author	Category	Duration	840	52.4
Author	Duration	Category	1788	51.9

TABLE I: The number of leaf nodes and hit ratio of different PLT structures.

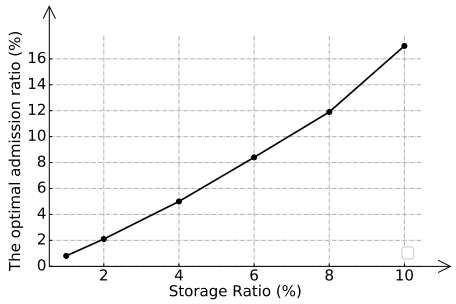


Fig. 16: The optimal admission ratio K (admitting K video groups into the cache) under different cache storage. (Video requests follow a Zipf distribution with $\alpha = 0.8$)

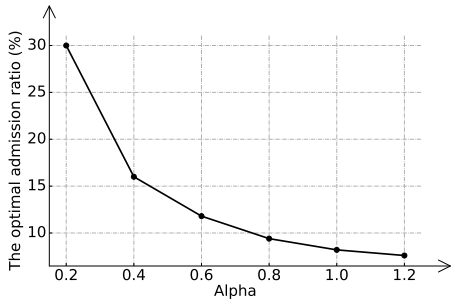


Fig. 17: The optimal admission ratio K (admitting K video groups into the cache) under request distribution of different values of α. (Cache storage ratio = 8%)

but some orders require more memory (*e.g.,* to store more leaf nodes) than other orders.

Admission ratio: In the admission phase (§VI), videos under top K video groups are admitted to enter the cache. The selection of the value K depends on both cache storage and the density of popularity distribution.

Figure 16 shows the relationship between cache storage and the optimal setting of K. The result indicates that when the cache storage becomes larger, the optimal value of K increases. This is because a cache with larger storage is capable to satisfy the preference of more users. An additional observation is that the growth of K is a little faster than the growth of the cache storage. When the cache storage increases, it begins to admit some not-so-popular videos. Our PLT's preference modeling to these videos is of less accuracy, so it admits more videos to be robust.

Figure 17 shows the relationship between the value of α and the optimal setting of K. The result indicates that when the video request distribution becomes sparse, the optimal value of K increases. This is because, in this situation, users' preference falls in more different video groups, rather than focusing on only a few top video groups.

What we learned from our parameter optimization: According to our parameter settings for PrefCache, we get the following insights.

- Different levels of model complexity are required by different request distribution. Generally, when users' preferences focus on a few popular videos, a simple model performs better than a complex model. A complex model has advantages usually when the users' preference is on more topics. As a result, in the future design of cache policies, analyzing the request distribution can be useful for designing an appropriate cache algorithm for a specific scenario.
- Users' preferences can be modeled in different ways. As shown in Table I, different orders of video grouping do not matter to the overall cache performance. Although some of them require more memory, the difference is not significant. This indicates that there should be methods other than our proposed PLT to model user preference on an edge cache. How to describe users' preference remains an open problem.

IX. Related Work

Video Caching Network: Since video traffic becomes the dominant traffic in the network, an increasing number of researches focus on optimizing cache policies specifically for video delivery. [43], [30], [11], [37] discussed some key factors which influence cache performance (user mobility, content popularity, cache capacity, cache strategy, geographic regions, and scenarios). [55], [38], [56] surveyed how to design a cache policy that can improve the quality of adaptive video streaming. Most of them are based on CDN caches. Compared with these works, we focus on designing a cache policy for video objects on the network edge.

Cache Admission policy: A complete cache policy usually consists of two modules: cache admission and cache replacement. Cache admission for general content items is well studied by researchers since 1994. They utilize item-wise request patterns to admit content objects in different ways. (1) frequency-based cache admission (2Q [32], TLFU [22] and BloomFilter [44]) admit objects based on their request frequency, and (2) size-based cache admission (Threshold [6], LRU-S [53], AdaptSize [17]) admit objects based on their size. Their difference is mainly on the implementation and their cache tuning methods ([45], [36], [29], [59], [13], [16]). Compared with these approaches, PrefCache has two differences in design choices. First, PrefCache is optimized for video caching, not for general content object caching. Second, all the above methods are based on item-wise request patterns. PrefCache learns users' preference for video groups rather than individual video objects.

Content-based recommendation: Content-related features have been widely utilized in recommendation systems [48], [34], [42], [15]. In these works, an algorithm first analyzes the content-related features and learns the user's preference of them, then it recommends some content objects that the

user may be interested in, or proactively requests content objects and stores them in the cache on the client-side. If a user requests an object which is pre-cached, the transmission delay is much lower. However, in order to lower the delay, the recommendation system needs much higher bandwidth consumption and large cache storage to proactively request content objects. It does not work on an edge caching system because of the constrained bandwidth and cache resources. Our work does not focus on how to request videos more than what the user wants, but on how to block some unpopular videos to save the cache space, which is fundamentally different from the above works.

X. Limitations

PrefCache only considers static preference-related Features: In PrefCache, all preference-related features are static features (*e.g.,* category, author, duration). In other words, once the video is generated and uploaded to the video provider, its features do not change. However, there do exist some other preference-related features which are dynamically changing for a given video object. For example, the number of "likes" and "dislikes" (it will increase during the time), how long is the time since it is uploaded (it also increases with the time passes by). We believe these features are also helpful to learn users' preferences for videos, so we do not claim that our model can provide optimal preference modeling. Moreover, we will try to model these dynamic features in our future work.

The side effect of the information collector: As stated in §IV, PrefCache requires an Information Collector to download metadata (*e.g.,* preference-related features) from the video provider. Although the bandwidth consumption is negligible (the feature information is plain text, which is about 10^4 smaller than the video object itself), the technical complexity is increased. As far as we know, the URL relationship of video objects and video information pages is not the same for all video providers. Moreover, a video provider may update its URL naming rules for its video objects an information pages. In this situation, PrefCache needs to be manually updated to know the new URL relationship between video objects and their information pages.

Limitations of preference learning by PLT: As described in §V-A, PLT judges whether a group of videos is under users' preferences by three heuristic rules. There is no guarantee that following these rules can always find users' preferences accurately. For example, under a video group, both hit ratio and request frequency is high, thus our PLT concludes that this group of videos is under users' preferences. However, a counter-example is that there is a single video object with extremely high popularity, and it increases the average hit ratio and request frequency of the whole group, although other videos under this group are not preferred by users.

Fortunately, our final goal is to improve the cache performance, not to accurately find all users' preferences. Our evaluation results show that mis-accepting all videos in such a video group does not significantly lower the cache performance, and we do not claim our algorithm is an optimal method to do this. We leave the refinement of our algorithm to our future work.

Source	www.bilibili.com
Trace duration (h)	24
Total requests	1.5×10^6
Videos being requested	2.5×10^4
Video categories	69
Video authors	1.3×10^4
Video duration (min)	1-189

TABLE II: Dataset summary

XI. Conclusion

In this paper, we revisit cache policies at the network edge. Due to the irregular request distribution on the network edge, current item-wise cache policies fail to predict the popularity of videos, leading to a limited cache performance.

We find that compared with single video objects, the users' preference for video topics is much more constant. We propose PrefCache, which admits videos by users' preference. Video objects are divided into groups by preference-related features, and a tree-structure learning algorithm is proposed to learn users' preferences. Moreover, an explore-and-exploit method is applied to do cache admission in real-time. Experimental results show the high performance of the proposed PrefCache.

Appendix A
Data collection

Real HTTP request traces are collected from *Bilibili* [3], one of the biggest video providers in China. We ran TCPdump on two main routers in a large Internet Service Provider (ISP) in China. Once a request arrives at the router, TCPdump will listen and record it. Since we are aware of the network topology of this ISP, we can figure out which edge router a packet comes from according to its source and destination IP address. Then we filtered all recorded requests to pick up all HTTP requests for video contents of Bilibili (by checking the protocol in TCP header and the URL in HTTP header). Moreover, to collect preference-related information of these videos, we also wrote a crawler program to download the HTML files of video information pages coupled with these videos. We collected 1.5×10^6 requests to over 2.5×10^4 different videos. The data summary is presented in Table I and the dataset we collected can be found in [2].

References

[1] https://developer.apple.com/streaming/.
[2] https://github.com/CACAproject/CACAproject.
[3] https://www.bilibili.com.
[4] https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.
[5] https://www.youtube.com.
[6] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox, "Caching proxies: Limitations and potentials," *Hewlett Packard*, 1995.
[7] C. Affiliates, "The zettabyte era: Trends and analysis," 2014.
[8] D. Agarwal, B.-C. Chen, and P. Elango, "Explore/exploit schemes for web content optimization," in *2009 Ninth IEEE International Conference on Data Mining*. IEEE, 2009, pp. 1–10.
[9] C. Aggarwal, J. L. Wolf, and P. S. Yu, "Caching on the world wide web," *IEEE Transactions on Knowledge and data Engineering*, vol. 11, no. 1, pp. 94–107, 1999.
[10] H. Ahlehagh and S. Dey, "Video caching in radio access network: Impact on delay and capacity," in *2012 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2012, pp. 2276–2281.

[11] ——, "Video-aware scheduling and caching in the radio access network," *IEEE/ACM Transactions on Networking (TON)*, vol. 22, no. 5, pp. 1444–1462, 2014.
[12] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.
[13] S. Bansal and D. S. Modha, "Car: clock with adaptive replacement," in *Usenix Conference on File and Storage Technologies*, 2004, pp. 14–14.
[14] E. Baştuğ, M. Bennis, and M. Debbah, "A transfer learning approach for cache-enabled wireless networks," in *2015 13th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*. IEEE, 2015, pp. 161–166.
[15] C. Basu, H. Hirsh, and W. Cohen, "Recommendation as classification: Using social and content-based information in recommendation," in *Fifteenth National/tenth Conference on Artificial Intelligence/innovative Applications of Artificial Intelligence*, 1998.
[16] D. S. Berger, S. Henningsen, F. Ciucu, and J. B. Schmitt, "Maximizing cache hit ratios by variance reduction," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 2, pp. 57–59, 2015.
[17] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, "Adaptsize: Orchestrating the hot object memory cache in a content delivery network," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 483–498.
[18] B. Bharath, K. G. Nagananda, and H. V. Poor, "A learning-based approach to caching in heterogenous small cell networks," *IEEE Transactions on Communications*, vol. 64, no. 4, pp. 1674–1686, 2016.
[19] B. Chen and C. Yang, "Caching policy for cache-enabled d2d communications by learning user preference," *IEEE Transactions on Communications*, vol. 66, no. 12, pp. 6586–6601, 2018.
[20] Y. Cui, D. Jiang, and Y. Wu, "Analysis and optimization of caching and multicasting in large-scale cache-enabled wireless networks," *IEEE Transactions on Wireless Communications*, vol. 15, no. 7, pp. 5101–5112, 2016.
[21] J. Durbin, "Efficient estimation of parameters in moving-average models," *Biometrika*, vol. 46, no. 3/4, pp. 306–316, 1959.
[22] G. Einziger and R. Friedman, "Tinylfu: A highly efficient cache admission policy," in *Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, 2014, pp. 146–153.
[23] Y. Guan, X. Zhang, and Z. Guo, "Caca: Learning-based content-aware cache admission for video content in edge caching," in *Proceedings of the 27th ACM International Conference on Multimedia*, 2019, pp. 456–464.
[24] Y. Guo, L. Duan, and R. Zhang, "Cooperative local caching under heterogeneous file preferences," *IEEE Transactions on Communications*, vol. 65, no. 1, pp. 444–457, 2016.
[25] K. Hamidouche, W. Saad, and M. Debbah, "Many-to-many matching games for proactive social-caching in wireless small cell networks," in *2014 12th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*. IEEE, 2014, pp. 569–574.
[26] S. He, H. Tian, and X. Lyu, "Edge popularity prediction based on social-driven propagation dynamics," *IEEE Communications Letters*, vol. 21, no. 5, pp. 1027–1030, 2017.
[27] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.
[28] J. Jiang, V. Sekar, H. Milner, D. Shepherd, I. Stoica, and H. Zhang, "Cfa: a practical prediction system for video qoe optimization," in *Usenix Conference on Networked Systems Design and Implementation*, 2016, pp. 137–150.
[29] S. Jiang and X. Zhang, "Lirs: an efficient low inter-reference recency set replacement policy to improve buffer cache performance," in *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2002, pp. 31–42.
[30] Y. Jiang, M. Ma, M. Bennis, F.-C. Zheng, and X. You, "User preference learning-based edge caching for fog radio access network," *IEEE Transactions on Communications*, vol. 67, no. 2, pp. 1268–1283, 2018.
[31] Y. Jiang, M. Ma, M. Bennis, F. Zheng, and X. You, "A novel caching policy with content popularity prediction and user preference learning in fog-ran," in *2017 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2017, pp. 1–6.
[32] T. Johnson and D. Shasha, "2q: A low overhead high performance buffer management replacement algorithm," in *International Conference on Very Large Data Bases*, 1994, pp. 439–450.
[33] V. Kirilin, A. Sundarrajan, S. Gorinsky, and R. K. Sitaraman, "Rl-cache: Learning-based cache admission for content delivery," in *Proceedings of the 2019 Workshop on Network Meets AI & ML*, 2019, pp. 57–63.
[34] C. Koch, G. Krupii, and D. Hausheer, "Proactive caching of music videos based on audio features, mood, and genre," in *ACM Multimedia Systems Conference*, 2017.
[35] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in neural information processing systems*, 1990, pp. 396–404.
[36] D. Lee, J. Choi, J. H. Kim, S. H. Noh, L. M. Sang, Y. Cho, and S. K. Chong, "On the existence of a spectrum of policies that subsumes the least recently used (lru) and least frequently used (lfu) policies," in *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 1999, pp. 134–143.
[37] M.-C. Lee, A. F. Molisch, N. Sastry, and A. Raman, "Individual preference probability modeling for video content in wireless caching networks," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–7.
[38] C. Li, L. Toni, J. Zou, H. Xiong, and P. Frossard, "Qoe-driven mobile edge caching placement for adaptive video streaming," *IEEE Transactions on Multimedia*, vol. 20, no. 4, pp. 965–984, 2017.
[39] S. Li, J. Xu, M. van der Schaar, and W. Li, "Trend-aware video caching through online learning," *IEEE Transactions on Multimedia*, vol. 18, no. 12, pp. 2503–2516, 2016.
[40] D. Liu, B. Chen, C. Yang, and A. F. Molisch, "Caching at the wireless edge: design aspects, challenges, and future directions," *IEEE Communications Magazine*, vol. 54, no. 9, pp. 22–28, 2016.
[41] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang, "A case for a coordinated internet video control plane," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, 2012, pp. 359–370.
[42] P. Lops, M. De Gemmis, and G. Semeraro, "Content-based recommender systems: State of the art and trends," in *Recommender systems handbook*. Springer, 2011, pp. 73–105.
[43] G. Ma, Z. Wang, M. Zhang, J. Ye, M. Chen, and W. Zhu, "Understanding performance of edge content caching for mobile video streaming," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 5, pp. 1076–1089, 2017.
[44] B. M. Maggs and R. K. Sitaraman, "Algorithmic nuggets in content delivery," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 3, pp. 52–66, 2015.
[45] N. Megiddo and D. S. Modha, "Arc: a self-tuning, low overhead replacement cache," in *Usenix Conference on File and Storage Technologies*, 2003, pp. 9–9.
[46] N. Metropolis and S. Ulam, "The monte carlo method," *Journal of the American statistical association*, vol. 44, no. 247, pp. 335–341, 1949.
[47] S. Müller, O. Atan, M. van der Schaar, and A. Klein, "Context-aware proactive content caching with service differentiation in wireless networks," *IEEE Transactions on Wireless Communications*, vol. 16, no. 2, pp. 1024–1036, 2016.
[48] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," in *The adaptive web*. Springer, 2007, pp. 325–341.
[49] K. Poularakis, G. Iosifidis, and L. Tassiulas, "Approximation algorithms for mobile data caching in small cell networks," *IEEE Transactions on Communications*, vol. 62, no. 10, pp. 3665–3677, 2014.
[50] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.
[51] M. Z. Shafiq, A. X. Liu, and A. R. Khakpour, "Revisiting caching in content delivery networks," in *The 2014 ACM international conference on Measurement and modeling of computer systems*, 2014, pp. 567–568.
[52] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, "Fast hash table lookup using extended bloom filter: an aid to network processing," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, pp. 181–192, 2005.
[53] D. Starobinski and D. Tse, *Probabilistic methods for web caching*. Elsevier Science Publishers B. V., 2001.
[54] T. Stockhammer, "Dynamic adaptive streaming over http: Standards and design principles," in *Proceedings of the Second Annual ACM SIGMM Conference on Multimedia Systems, MMSys 2011, Santa Clara, CA, USA, February 23-25, 2011*, 2011.
[55] Z. Su, Q. Xu, F. Hou, Q. Yang, and Q. Qi, "Edge caching for layered video contents in mobile social networks," *IEEE Transactions on Multimedia*, vol. 19, no. 10, pp. 2210–2221, 2017.
[56] T. X. Tran, P. Pandey, A. Hajisami, and D. Pompili, "Collaborative multi-bitrate video caching and processing in mobile-edge computing networks," in *2017 13th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*. IEEE, 2017, pp. 165–172.

[57] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. Leung, "Cache in the air: Exploiting content caching and delivery techniques for 5g systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 131–139, 2014.
[58] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, and X. S. Shen, "Content popularity prediction towards location-aware mobile edge caching," *IEEE Transactions on Multimedia*, vol. 21, no. 4, pp. 915–929, 2018.
[59] Y. Zhou, J. Philbin, and K. Li, "The multi-queue replacement algorithm for second level buffer caches," in *General Track: 2001 Usenix Technical Conference*, 2001, pp. 91–104.
[60] H. Zhu, Y. Cao, W. Wang, T. Jiang, and S. Jin, "Deep reinforcement learning for mobile edge caching: Review, new features, and open issues," *IEEE Network*, vol. 32, no. 6, pp. 50–57, 2018.