Data-driven Hybrid Caching in Hierarchical Edge Cache Networks

***Abstract*—Hierarchical cache networks are increasingly deployed to facilitate high-throughput and low-latency content delivery to end users. Meanwhile, user content requests present both long-term and short-term variabilities. To achieve the maximal caching gain, it is critical for the caching algorithms to adapt to long-term and short-term content popularity dynamics. However, the traditional reactive caching algorithms, such as LRU and LFU, only respond to the short-term request dynamics, missing the long-term popularity trend. The problem is especially severe in the edge caching setting, due to the limited caching capacity and the reduced degree of user multiplexing. On the other hand, the recently proposed optimal proactive caching solutions only update content caching and routing periodically based on the estimated long-term content popularity. To fill in the gap, we propose a novel hybrid proactive-reactive caching framework. We first develop a fine-grained content popularity estimation method that mines the future content interests of small groups of users based on user content request history data. We then design a sliding-window based mechanism to periodically update content popularity prediction and optimize the content placement and routing to trade-off between minimizing user delays and controlling cache update costs in hierarchical cache networks. We further show that reactive LRU caching algorithm can be seamlessly combined with our proposed proactive caching algorithms to respond to the real time content request dynamics. Finally, we evaluate our algorithms using real world dataset and show that up to 30% caching performance improvement can be achieved over with the state-of-art methods.**

***Index Terms*—edge caching, recommender system, cache placement and routing**

I. Introduction

The Internet traffic is dominated by the delivery of various contents, ranging from web-pages, software packages, to 360 degree videos, etc. The emerging applications, such as Connected Vehicle, Virtual Reality and Augmented Reality, require contents to be delivered with high bandwidth and low latency. A recent trend in Content Delivery Network (CDN) design is to push content servers closer to end users in both wireline and wireless networks [1]–[3]. While pushing content closer to the network edge can facilitate high-bandwidth and low-delay content delivery, edge caching faces new challenges resulted from the limited resource availability and the reduced degree of user multiplexing. More specifically, for financial viability, an edge cache node is only equipped with limited computation, memory, and storage resources. Due to its "shallow" network location, an edge cache normally serves a small user population within the same subnet, e.g, all users connected to the same LTE base station. The aggregate content request patterns for a small user population are more dynamic and more sensitive to individual users' behaviors than those for a large user population. As a result, the traditional caching algorithms, such as the Least-Recently-Used (LRU) and Least-Frequently-Used (LFU), are not expected to work well for edge caching [1]. To address the challenges, hierarchical caching networks are introduced so that content requests missed by the edge caches can be served by upper level caches, which are equipped with more resources and have a wider network coverage. Many recent studies e.g., [4]–[7], have investigated the optimal content placement and routing in cache networks based on the estimated future content popularity. However, such solutions are vulnerable to fine time scale content popularity dynamics that cannot be captured by simple history-based popularity estimations. As a result, the caching performance on edge caches is still far from being optimal.

In this paper, we propose a novel hybrid proactive-reactive caching framework to adapt the caching and routing decisions to the long-term and short-term content popularity trends in edge caching networks. Our main contributions are four-fold:

1) **Group-Interest-Based Popularity Estimation.** To capture the fine-grained content popularity dynamics, we propose to use a recommender system technique, Matrix Factorization, to mine the content interests of small user groups from their content request history data. We demonstrate that MF-based content popularity estimation can significantly improve the estimation accuracy of simple history-based estimation, especially for content with medium and low popularity.
2) **Proactive Optimal Placement and Routing.** To adapt to long-term content popularity evolution, we propose a sliding-window based mechanism to periodically update content popularity prediction and optimize the content placement and routing for hierarchical cache network. We study the trade-off between minimizing user content request delays and controlling network costs for cache updates. We also develop an approximate algorithm to obtain good solutions with low computation cost.
3) **Reactive Realtime Adaption.** Periodical caching updates cannot respond to sudden request pattern changes, such as flash-crowds, in a timely fashion. To address this problem, between two proactive updates, we propose to adopt the reactive LRU algorithm to update a portion of the caches according to the realtime content request patterns. We show that the LRU algorithm can work seamlessly with our proposed proactive caching algorithms in the hierarchical setting.
4) **Experiments driven by Real Traces.** We evaluate the

proposed caching framework and algorithms through extensive experiments driven by video request traces generated by real users. We systematically investigate the performance impact of several key system parameters. We demonstrate that our proposed hybrid caching solutions can bring significant network delay/cost reduction over the state-of-art caching algorithms.

The rest of this paper is structured as following. In Section II, we briefly go through the related work. The caching network model and the overall system framework are introduced in Section III. Matrix-Factorization based content popularity estimation is developed in Section IV. The optimal proactive placement and hybrid caching schemes are presented in Section V. Section VI presents the results of experiments driven by real video request traces. Finally, the paper is concluded by Section VII.

II. Related Work

To meet the new challenges of content delivery, more and more researchers are focusing on edge caching from various angles, such as network architecture design, caching algorithms design, and evaluation of edge caching algorithms, e.g. [1]–[3]. There are also many recent studies on caching network optimization. For example, [6]–[9] studied how to jointly optimize routing and caching in single-level cache networks with different user-cache delays. [4] studied the joint optimization of routing and caching on arbitrary topology. These studies focused more on the optimization formulation and approximate algorithm development by assuming that future content popularity can be accurately estimated. In our work, we first develop a fine-grained content popularity estimation model, and then use the estimated popularity distribution to optimize content placement and routing in "multi-level" hierarchical cache networks, which are more realistic in real world settings.

For centralized caching, most of the previous studies on content popularity assume certain arrival process of the content requests, such as the shot noise model [10] and Zipf's distribution [11]. However, in the edge caching problem, the real content popularity distribution may not follow the assumed distributions based on many users. The content distribution for a small group of users is more dependent on the users' personal interests. This motivates us to propose a user/group interest based content popularity estimation method. In the research line of user interest modeling and recommendation, lots of methods have been proposed. Some methods, such as matrix factorization [12] and collaborative filtering [13], [14], rely on user-content rating information. Other methods employ content analysis and Natural Language Processing techniques to solve the problem [15]–[17]. However, the existing recommendation algorithms cannot be directly adopted to solve the edge caching problem. Recommendation algorithms only predict a user's interest to a content, and do not predict when the user will likely to consume the content, which is very important for caching decision. Additionally, the traditional recommendation algorithms focus on modeling individual user interest profiles, while in the caching problem each cache server serves a group of IP addresses and there may be multiple users behind each IP address, so we need to profile the content interests of a group of users for the caching purpose.

It has been shown in [18] that for a single cache that, given a routing policy, static caching achieves the minimum expected delay with a fixed number of contents stored in network. However in practice nowadays, new contents constantly emerge and some of them can suddenly become viral and attract flash-crowds of users. Static caching alone cannot handle such content popularity dynamics. We will show in our work, by combining periodical proactive cache update with realtime reactive caching adjustment, one can handle both long-term and short-term content popularity dynamics. We use the average network delay for user requests and content updates as our main evaluation metric. There are other caching studies that use other metrics, such as file download latency and retention [19]–[21]. Our framework and algorithms are open to adopt those metrics. Finally, we reported some preliminary results of MF-based popularity estimation for a single edge cache node in a recent workshop paper.[1] This paper is much broader than the workshop paper by focusing on hybrid caching for hierarchical cache networks.

III. System Overview

A. Hierarchical Edge Cache Network

We consider a hierarchical network of caches with 'multi-level' topology. Users who generate requests are connected to one or more edge caches at the bottom level of the network. A request for a content that is not cached at the edge level is recursively forwarded up along the hierarchy until it is served by either the closest cache storing the requested content or the back-end server storing all the contents. We further assume that a content request will be forwarded along a predetermined single (e.g. shortest) path from an edge cache to the back-end server. Figure 1 shows a toy example. The caching network consists of six caches and N users. The caches are placed at two different levels: c_1, c_2 at the upper level and $c_3, \cdots, c_6$ at the lower (edge) level, cache capacities are set to $C_1, \cdots, C_6$ respectively. Content requests are generated by users $\{i_1, ...i_N\}$. The whole catalog consists of K contents $\{j_1, ...j_K\}$ of the same size. The content request arrival rate of user i is λ_i, and for each request of user i the probability of accessing content j is p_{ij}.

The delay incurred by a user's content request depends on where it is served: the shortest delay if served by the edge cache, longer delay if served by a cache at a higher level, and the longest delay if served by the back-end server. For the two-level hierarchy in Figure 1, the delay incurred by user i when downloading content from the back-end server, upper-level cache m' and lower-level cache m are denoted as $d_i^b, d_{im'}^u$ and d_{im}^l respectively. As is the case in reality, we assume that $d_i^b > d_{im'}^u > d_{im}^l$. We also focus on a congestion-insensitive setting where the delays are not dependent on the loads of

[1] We omit the citation to maintain the anonymity of this paper.

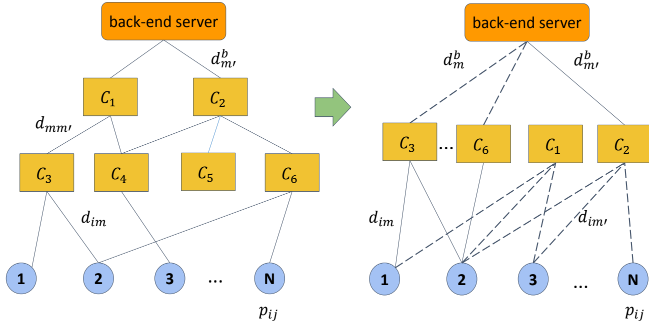

Fig. 1: Multi-level Hierarchical Cache Network Topology

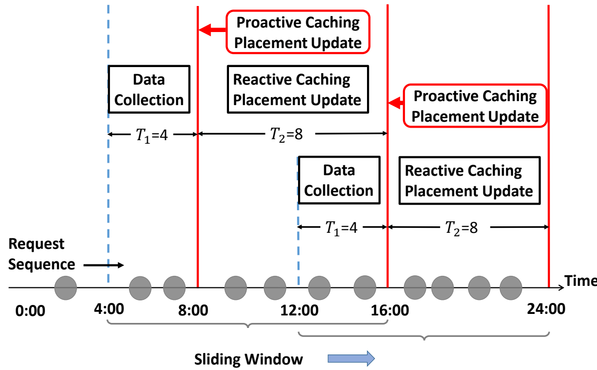

Fig. 2: Periodical Popularity Estimation and Caching Update

cache and back-end servers.[2] Our design goal is to jointly determine the content placement policy (i.e. which contents are stored on which cache(s)) and request routing policy (i.e. where a request is served) to minimize the overall delays of all users.

B. Data-driven Proactive Caching with Periodical Updates

One caching solution is to proactively load contents into caches based on the predicted content popularity. To deal with content popularity dynamics, one can periodically update the popularity estimation and adjust the preloaded content accordingly. We consider an online proactive caching framework as shown in Figure 2. Let T_1 be the time interval used to collect past content request history data. After sufficient data are collected, the content popularity for all users will be estimated and content placements on all caches and content routing for all users will be adjusted. The content placement and routing will remain static for a subsequent interval of T_2. At the end of T_2, the next popularity estimation and content placement/routing updates will be conducted based on the data collected during the previous time interval of T_1. The timing of periodical updates is controlled by T_1 and T_2. Specifically, T_1 controls the "freshness" of data used for the estimation: adopting smaller T_1 makes the estimation more sensitive to the recent demand dynamics, but less stable. T_2 controls the cache update frequency: adopting larger T_2 can reduce the frequency and cost of cache updates, but at the risk that the estimated popularity may not match well with the actual user content interests in a longer future time window.

[2]Our framework and algorithms can be extended to cover congestion-dependent delay. For the clarity of presentation, we only focus on congestion-insensitive delay in this paper.

C. Hybrid Proactive and Reactive Caching

One drawback of proactive periodical update is that it only updates caches once at the end of each T_2 interval, therefore cannot respond to content popularity dynamics in realtime. The problem can be severe when T_2 is large. On the other hand, the traditional caching strategies, such as Least-Recently-Used (LRU) and Least-Frequently-Used (LFU), don't proactively estimate future content popularity and don't preload content. Instead, they reactively update the cached contents after serving each request. As a result, they tightly follow the content popularity in realtime. To take the advantages of both proactive and reactive caching, we will augment our estimation-based proactive caching with LRU to adapt to realtime content popularity changes between two periodical updates. More details will be given out in Section V-D.

IV. Group-Interest-based Popularity Estimation

To achieve good performance in proactive caching, it is crucial to accurately estimate the content popularity in future. One naive approach is to take empirical content popularity distribution in T_1 and simply use it as popularity estimation in T_2. Such an approach was adopted by previous studies, e.g. [8], for its simplicity. However, the estimation accuracy is poor when the user population served by each cache server is small (which is the case for edge caching). In order to achieve more accurate popularity estimation, we should dig deeper into the user request data and generate more fine-grained estimation.

Specifically, we adopt the widely used recommender system algorithm — Matrix Factorization (MF) [22] and customize it for the edge caching problem. Given a sparse rating matrix $\vec{R}^{m\times n} = [r(i,j)]_{m\times n}$ that a set of m users generated over n contents, the goal is to estimate the missing rating (entry) in $\vec{R}^{m\times n}$. The MF model assumes that each user has an unknown latent vector $\vec{u_i} \in \mathbb{R}^d$ and each content has an unknown latent vector $\vec{v_j} \in \mathbb{R}^d$ $(d \ll min(m,n))$. $\vec{v_j}$ can be interpreted as the latent features of content j, such as genre and topic; $\vec{u_i}$ can be interpreted as user i's preference on different features. The predicted rating of user i for content j is:

$$r(i,j) = u_{i_1}v_{j_1} + u_{i_2}v_{j_2} + ... + u_{i_d}v_{j_d} = \vec{u_i}^T\vec{v_j}. \quad (1)$$

The problem then can be formulated as finding matrices $\vec{U}$ and $\vec{V}$ such that $\vec{U}\vec{V}^T$ approximates $\vec{R}$ with the least approximation error:

$$\begin{aligned} \underset{\vec{U},\vec{V}}{\text{minimize}} \quad & \|\vec{R} - \vec{U}\vec{V}^T\| \\ \text{subject to} \quad & \vec{U},\vec{V} \geq 0. \end{aligned} \quad (2)$$

In our caching problem, we treat a group of users behind a set of IP addresses as a superuser and predict each

superuser's content accessing probability. Formally, given a set of IP addresses: $\vec{I} = \{IP_1, IP_2, ...IP_{|I|}\}$, and K content files: $\vec{f} = \{f_1, f_2, ...f_{|K|}\}$, we partition $\vec{I}$ into small groups: $\vec{G} = \{g_1, g_2, ..., g_{|G|}\}$, where for a certain group i, $g_i = \{IP_{i_1}, IP_{i_2}, ..., IP_{i_{|g_i|}}\}$. All users behind IP addresses in group i are assumed to be served by the same edge content server. In content request data collection window T_1, we define $\vec{S}(IP_{i_x}, c_j)$ as the frequency at which any IP address in group i accesses content j. To calculate the preference of group i over content j, we use:

$$r'(i,j) = \frac{\sum_{x=1}^{|g_i|} S(IP_{i_x}, c_j)}{\sum_{j=1}^{|C|} \sum_{x=1}^{|g_i|} S(IP_{i_x}, c_j)}, \tag{3}$$

where $r'(i,j)$ is the observed rating of group i to content j. We then use $r'(i,j)$ as the entries in our "group-content rating matrix" $\vec{R}$. Notice that we use a normalized score ($r'(i,j) \in [0,1]$) instead of the absolute frequency to measure the preference to avoid any large groups or popular contents dominating the rating matrix.

By recording the access logs of all users in the group to all the contents, we can use MF to estimate the future preference for those contents that have not been accessed by this group. In practice, we choose the Root Mean Squared Error (RMSE) as our objective function in the training phase of MF:

$$RMSE = \sqrt{\frac{1}{N} \sum_{(i,j)} (r(i,j) - r'(i,j))^2}, \tag{4}$$

where N is the number of data points. In order to minimize the RMSE, we use the algorithm in [23]. The running time of matrix factorization over a $10^3 \times 10^5$ rating matrix is on the order of minutes. In practice, it is not scalable to generate rating matrix and MF results in real time. According to our periodical update framework Figure 2, we only run MF to update $\{r'(i,j)\}$ every T_2 period.

V. Data-driven Caching Placement and Routing

Similar to [5], we model the cache placement problem as follows: let x_{jm} to be the binary cache placement variable, $x_{jm} = 1$ if content j is to be placed in cache m and $x_{jm} = 0$ otherwise. Let r_{ijm} to be the binary routing variable, $r_{ijm} = 1$ if the request for content j from user i is routed to cache m and $r_{ijm} = 0$ otherwise. We use l_{im} to denote the connectivity between user i and cache m, $l_{im} = 1$ if i and m are linked and 0 otherwise. In the context of hierarchical cache network where only the adjacent levels can be connected and there is no peering link between caches at the same level, a user can potentially download content from his connected edge cache nodes and their ancestor cache nodes towards the back-end server, following the pre-determined single (shortest) path. We can therefore flatten the multi-level cache network into a single-level network and create virtual links between each user and all cache nodes that he can potentially access. Each virtual link is weighted by the delay between the user and the cache node. For the example in Figure 1, the upper-level caches c_1, c_2 are moved to the lower-level, and virtual links with new delays d_{i1}, d_{i2} are added between c_1, c_2 and users who can access them. Our goal is jointly optimizing the content placement and routing strategy to minimize the overall network delay.

We define the overall delay function as $F(\mathbf{x}, \mathbf{r})$ which depends on the routing and placement strategies $\mathbf{x}, \mathbf{r}$. The problem can then be written as:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{r}} \quad & F(\mathbf{x}, \mathbf{r}) \\ & r_{ijm} \le l_{im}, \quad r_{ijm} \le x_{jm}, \quad \forall i, j, m \\ & \sum_m r_{ijm} \le 1, \quad \forall i, j \\ & \sum_j x_{jm} \le C_m, \ \forall m \\ \text{s.t.} \quad & r_{ijm} \in \{0,1\}, \ \forall i, j, m, \quad x_{jm} \in \{0,1\}, \ \forall j, m, \end{aligned} \tag{5}$$

where the first constraint guarantees the connectivity between user i and cache m, and the content availability on cache m, the second constraint says each user will choose at most one cache to download content, the third constraint models the limited capacity on each cache.

In proactive caching, the caches update the stored contents to adapt to new content request patterns. Downloading content from the back-end server to caches will incur extra delay and network bandwidth cost. For the two scenarios where the downloading delay/cost for cache updates can or cannot be ignored, we introduce two models *i) basic model* and *ii) update-cost-aware model* with two different objective F functions in the following sections.

A. Basic Model

First we consider the scenario where the downloading cost for cache updates can be ignored. Given the fixed routing and placement policy, $\mathbf{x}$ and $\mathbf{r}$, the overall delay function F in this case can expressed as:

$$F(\mathbf{x}, \mathbf{r}) = \sum_i \sum_j \lambda_i p_{ij} \left[\sum_m r_{ijm} d_{im} + \left(1 - \sum_m r_{ijm}\right) d_i^b \right], \tag{6}$$

where the first term is download delay from a selected cache, the second term is the delay when the content has to be downloaded from the back-end server.

Note that there may exist multiple paths from user i to an upper-level cache m' through different lower-level edge caches. For the optimal routing policy without considering congestion delay on caches, there is no incentive to choose the path with larger delay, so we can define the virtual link delay $d_{im'}$ as the minimum delay between user i and upper-level cache m'. For the two-level hierarchy in Figure 1, the delay on the virtual link between user i and upper-level cache m' can be calculated as:

$$d_{im'} = \min_m (d_{im} + d_{mm'}), \tag{7}$$

where m is chosen from all edge caches that user i is connected to in the original topology.

B. Update-cost-aware Model

Next we consider the scenario where the downloading cost for cache updates cannot be ignored. In proactive caching, the downloading cost is proportional to the difference between the new cache list and the current cache list from the last update. The cost for downloading one content from the back-end server is proportional to the distance between the cache and back-end server. We define the content downloading delay for cache m to be d_m^d. In periodical proactive caching, there is only one update in each T_2 interval. Let t be some time slot and $\mathbf{x^t}$ be the placement policy at t. Then the aggregated downloading delay for update at time slot t can be written as:

$$\mathcal{D}^t(\mathbf{x^t}, \mathbf{x^{t-1}}) = \sum_m \sum_j \left(x_{jm}^t - x_{jm}^{t-1}\right)^+ d_m^d, \quad (8)$$

where $(x)^+ = x$ if $x > 0$, and 0 otherwise.

Note that, instead of downloading from the back-end server, a cache can download content j from its closest ancestor cache node that stores j to reduce download delay. For the two-level hierarchy, the downloading delay d_m^d when content j is requested for the lower-level cache m can be reduced to $d_{mm'}$ if there exist an upper-level cache m' that stores j and is connected to m. That is:

$$d_m^d(j) = \begin{cases} \min\limits_{m' \in M'(j,m)} d_{mm'}, & \text{if } M'(j,m) \neq \emptyset \\ \min\limits_{m'} d_{mm'} + d_{m'}^d, & \text{otherwise} \end{cases}$$

where $M'(j,m) \triangleq \{m' | j \in \text{cache } m' \text{ and } l_{mm'} = 1\}$ is the set of upper level caches that are connected to m and have content j. Given the downloading delay for each update, the overall delay F^t at time t in update-cost-aware model can be calculated as a weighted sum:

$$F^t(\mathbf{x}, \mathbf{r}) = F(\mathbf{x}, \mathbf{r}) + \eta \mathcal{D}^t, \quad (9)$$

where F is calculated in (6), and η is the weight of cache content update cost in the overall objective function.

C. Approximate Solution

Using techniques similar to [5], the problem described in Section V-A can be shown to be NP-complete, and approximate solution with a performance factor of $(1 - 1/e)$ can be developed. Due to the space limit, we skip the presentation here. However that solution ignores the downloading cost for each update, which is not acceptable when updates are frequent i.e., T_2 is small. In this section, we will develop an approximate algorithm for the update-cost-aware model in Section V-B, which shares the NP-completeness as the basic model in Section V-A.

Algorithm 1 is developed based on the following procedure: first the algorithm starts with empty caches and initialize the delay for user i to access content j to the delay for accessing the back-end server d_i^b. Then at each step the algorithm greedily finds a position in all caches that maximize the delay reduction of $\sum_i \lambda_i p_{ij}(d_{ij} - \min(d_{ij}, d_{im}))$ minus the download cost of $\eta d_m^d(j)$ if the content is not currently cached

Algorithm 1: Approximate solution for update-cost-aware model at time t

```
   Initialize: S = {S_1, S_2, ...S_M};
               S_m = {1, 2...K}, ∀m = 1...M;
               d_ij = d_i^b, ∀i, j;
1  while S ≠ ∅ do
2      G_jm ← [0]_{K×M} ;
3      for S_m ∈ S, m = 1...M do
4          for j = 1...K do
5              g ← 0 ;
6              for i=1...N do
7                  g = g + λ_i p_ij (d_ij − min(d_ij, d_im))
8              end
9              if x_jm^{t−1} == 0 then
10                 G[j][m] ← g − η d_m^d(j)
11             else
12                 G[j][m] ← g
13         end
14     end
15     (j*, m*) ← argmax_{j,m} G_jm ;
16     if K − |S_m| < C_m then
17         S_m ← S_m − j* ;
18     else
19         S ← S − S_m
20     d_ij* ← min(d_ij*, d_im*), ∀i
21 end
22 Caching: obtain new cache placement {x_jm^t, ∀j, m} ;
23 Routing: each user downloads content j from the nearest
   cache storing j.
```

($x_{jm}^{t-1} == 0$) based on the previous solution. The procedure iterates until all caches are filled.

After Algorithm 1 completes, we get the new cache placement $\{x_{jm}^t\}$ for time t. To implement the new placement, on cache m, one only needs to download content j with $x_{jm}^t = 1$ and $x_{jm}^{t-1} = 0$, and remove content with $x_{jm}^t = 0$ and $x_{jm}^{t-1} = 1$. To facilitate a lower-level cache downloading missing content from an upper-level cache, we can update caches in a *bottom-up* fashion: first update the bottom level caches according to the new placement, download missing content from a upper level cache if possible; move one level up, repeat the step until caches at all levels are updated.

D. Hybrid Caching: MF+LRU

To adapt to the content request dynamics between two cache updates in a timely fashion, we combine the periodical proactive caching placement with a reactive caching strategy Least Recently Used (LRU) [24]. LRU is a widely used reactive strategy that simply evicts the least recently used content all the time. We divide the capacity of each cache into two portions: one is used to store the suggested contents from proactive placement, the other one is used to store the suggested contents from LRU. Between any two cache updates, the "proactive" part of the cache remains unchanged,

while the “LRU” part is constantly updated according to its replacement rule. When a request is routed to a cache according to the proactive placement and routing solution, if there is a hit in the proactive cache portion, the content will be served directly; if it is a miss in the proactive portion,[3] the LRU portion will be checked, if hits, it will be served by LRU, if LRU also misses, the content request will be routed back towards the back-end server along the shortest path in the cache network. When the back-end server sends the content to the user, all caches along the shortest path check whether it already has a copy of the content in its proactive portion, if yes, no action required; if not, it will add a copy of the content into its LRU portion and evict another content according to the LRU rule. For a content request that is routed to the back-end server according to the proactive routing solution, it will be processed in a similar way as in the LRU miss case. We control the sizes of the two portions by a weight α. For example, if α is set to 0.6, 60% of the contents in the cache are pre-loaded by proactive placement at each update instant, and the rest are constantly updated by LRU.

VI. Performance Evaluation

In this section, we present performance evaluation results from experiments driven by real world user content request dataset. Our goal is to evaluate: *1) how well the MF-generated popularity helps with reducing the overall network delay? 2) whether combining reactive and proactive caching strategies will outperform the pure proactive strategy? 3) how the update-cost-aware model controls the cost of cache updates?*

A. Video Request Dataset

We use a real world content request trace collected by a major Over-The-Top (OTT) video streaming service provider in China. The served contents range from TV shows, movies, to live news and sports programs. The trace includes the sequence of video content requests generated by IPs located in several provinces in China from June 2014 to September 2014. In Table I, we show the basic statistics of the trace. From the trace we know in real world video streaming systems content requests are highly time-varying and can be sparse at times.

Statistics	Value
Avg. requests	1,335,488.0/day
Avg. # of Unique IP addresses	215,908.6/day
Avg. # of Unique Contents	83,859.4/day
# of groups under prefix 16	544
# of groups under prefix 18	1,307
# of groups under prefix 20	4,068

TABLE I: Basic Statistics of the Trace

[3]Since the proactive cache portion is smaller than the cache size in the original calculation, some content that are supposed to be cached in the pure proactive solution cannot be cached in the hybrid solution.

B. Experiment Setup

As discussed in section IV, we customize the widely used recommender algorithm “Matrix Factorization” (MF) to achieve better content popularity estimation. For Matrix Factorization algorithm to work in caching problem, we first need to construct a rating matrix that represents users’ preferences on each content. However in practice, due to NAT and multiple users sharing the same OTT device, it may be hard to identify who are the users behind each IP address. Instead of estimating content access probability for each user, we estimate content accessing probability for a group of users. Specifically, we group all IPs sharing a common prefix at certain length k and treat each group as one super user, and use the number of accesses of one content from all IPs in this group in a time window as the group’s content rating. Since we don’t have access to each IP’s geographic location, for simplicity, we further assume that the IPs sharing with the same prefix form a subnet and are served by the same CDN edge server.[4] For this reason, we can assign a cache list to each subnet. The longer the IP prefix length, the fewer the IPs falling into each subnet. As a result, the requested content distribution fluctuates more, and is more sensitive to individual users’ personal interests. When determining the network location of a CDN server, we set a long prefix for each subnet if we want to push the CDN server close to the edge, and a short prefix if we want to place the CDN server deep in network core. In our experiments, we choose prefix 16 for the upper-level caches and prefix 18 for the lower-level caches. All the IPs sharing a prefix 20 are grouped and treated as one superuser for the corresponding prefix 18 cache. In our dataset, we have 554 upper-level caches, 1,307 lower-level caches and 4,068 superusers.

We divide the dataset along time and use the T_1, T_2 parameters discussed in Section III-B as the history data collection interval and proactive caching list holding time. For example, for $T_1 = 8, T_2 = 2$, the proactive caching update times in one day are: $8:00, 10:00, 12:00, ...24:00$, and the data collection interval for the above updates would be $(0:00 \sim 8:00), (2:00 \sim 10:00), ...(16:00 \sim 24:00)$. We test the performance of our algorithms with different cache capacity setting in each $/16$ network, consisting of one upper-level cache ($/16$), four lower-level caches ($/18$), and maximally sixteen superusers ($/20$). We set the total capacities of all five caches to be $10\%, 30\%, 50\%, 70\%$ of the size of the catalog observed. In reality, the upper-level caches normally have larger capacity than the lower-level caches. So in our experiments, we test several situations where the upper-level cache capacity is $4, 8, 16, 32, 64, 128$ times of the lower-level cache capacity. We set the user downloading delays from the back-end server as $d_i^b = 50$, $\forall i$, from the lower-level caches as $d_{im} = 5.5$, $\forall i, m$, from the upper-level caches as $d_{im'} = 11$, $\forall i, m'$.

[4]In reality, IP grouping method can be complex, but our algorithm works with any other grouping method.

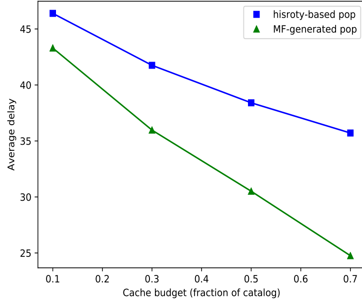

Fig. 3: Caching performance at different cache sizes. Upper-level cache size is four times of each lower-level cache. $T_1 = 2$ hours, $T_2 = 2$ hours.

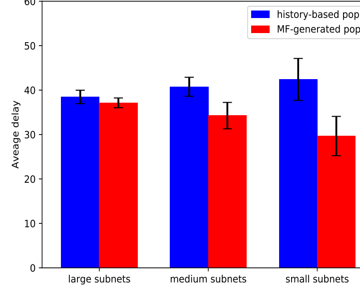

Fig. 4: Caching performance for different subnet sizes. Upper-level cache size is 4 times of lower-level cache. $T_1 = 2$ hours, $T_2 = 2$ hours.

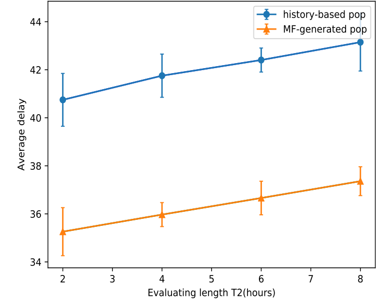

Fig. 5: Caching performance at different update frequencies: T_1 is fixed at 2 hours, while T_2 is varied from 2 to 8 hours.

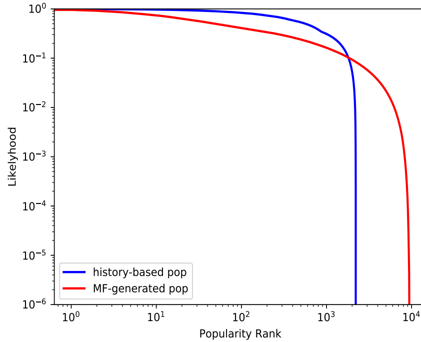

Fig. 6: Difference between history-based and MF-generated content popularity estimation: CCDF of content access likelihood as a function of content rank

C. MF-generated vs. History-based Popularity Estimation

Based on the prefix-based grouping, we now compare the history-based empirical content popularity distribution collected in T_1 period with the content popularity generated by MF. Figure 6 shows the difference for one of the /20 superuser. Due to the capability of inferring the unobserved ratings of superusers, compared with the history-based content popularity, MF-generated content popularity has similar but smoother popularity decay for those popular contents. Meanwhile, for those unpopular (cold) contents, MF-generated popularity has reasonably accurate estimation, while history-based popularity estimation assigns zero probability to contents not observed in the T_1 period.

Figure 3 compares the network delay performance between feeding history-based popularity and MF-generated popularity into the greedy algorithm 1 with different cache sizes. It can be seen that using MF-generated popularity can reduce the average content request delay at all cache sizes. The performance gap between using MF and history-based popularity increases as the total cache size increases.

Figure 4 compares the performance for $/16$ IP subnets with different sizes. We label a subnet whose total content request number within $T_2 = 2$ hours falls into 50K $\sim$ 100K, 5K $\sim$ 10K and 0.5K $\sim$ 1K as a large, medium, or small IP subnets, respectively. We then sample several subnets from each of the three categories out of the dataset and test the performance of our algorithms for subnets in each category. It can be shown that as the subnet size decreases, caching driven by history-based popularity estimation performs worse (delay increases), while caching driven by MF-generated popularity performs better. This is because MF algorithm can more accurately predict the content accessing probability of each superuser and adapt better to the more fluctuated and unstable observed content distribution as subnet size decreases.

The above results demonstrated that MF-generated popularity estimation can truly help with reducing the network delay. In this experiment we will show how caches at both levels perform. Table II shows the numbers of hit requests at each level. It can be seen that for history-based popularity estimation the upper level hit numbers are significantly less than the lower level hit numbers. One explanation could be the lower-level caches have already taken care of the most popular contents, the contents escaped from the lower level to the upper level are the cold contents that are harder to be predicted by the simple history-based popularity estimation. To the contrary, MF-based popularity estimation works better with cold contents. Another observation is that when the cache size is large, the MF method could outperform the history-based method by up to 30%. This is because MF method can rank the cold contents which can be cached to reduce delay when the cache size is large.

Figure 5 shows the performance impact of the update frequency controlled by parameter T_2. The results show that the performance of two popularity estimation methods degrade almost at the same pace as the updating frequency decreases. This is because, if not updated frequently enough, the estimated popularity gets outdated and cannot keep track of the current content popularity. Figure 7 shows the caching

TABLE II: Hit Number at Different Level and Different Cache Sizes ($T_1 = 2, T_2 = 2$)

Total Capacities (% of catalog)	Popularity Est. Method	Back-end Server Download Times	Hit # in lower-level	Hit # in upper-level	Average Delay (time units)	Gain(%)
0.1	history-based	1735.66	102.38	58.72	46.39069255	—
	MF-generated	1597.27	188.72	110.77	43.29484489	6.673424098
0.3	history-based	1525.66	212.44	158.66	41.75366414	—
	MF-generated	1264.5	354.94	277.33	35.9705183	13.85063074
0.5	history-based	1371.16	271.22	254.38	38.40648791	—
	MF-generated	1012.05	447	437.72	30.51288243	20.55279175
0.7	history-based	1244	299.22	353.55	35.71058167	—
	MF-generated	742.33	518.66	635.77	24.75937915	30.66654758

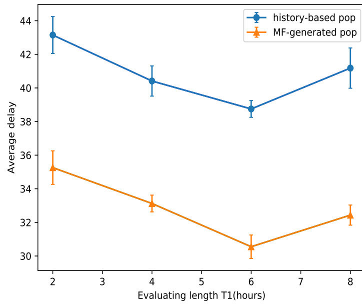

Fig. 7: Caching performance at different update frequencies: $T_2 = 8$, while T_1 is varied from 2 to 8 hours.

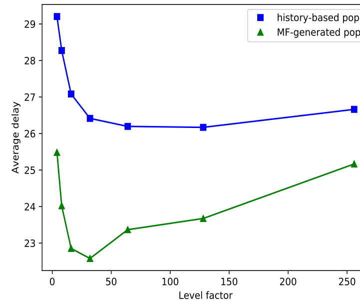

Fig. 8: Caching performance under different ratios of upper cache size over lower cache size. $T_1 = 6, T_2 = 2$.

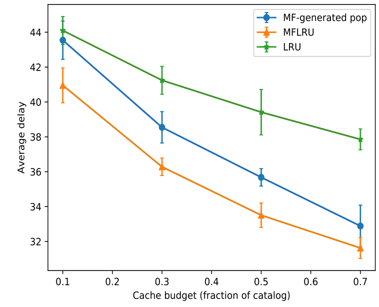

Fig. 9: Caching performance of MF, MFLRU and LRU with large update interval: $T_1 = 24$, $T_2 = 24$.

performances when the data collection interval is set to $T_1 = 2, 4, 6, 8$ while T_2 is fixed at 8. It shows that the system achieve the best performance when $T_1 = 6$. This suggests that the history data collection window has to be carefully chosen: too small a window does not collect enough data for accurate prediction and too large a window will introduce estimation interference from old content request history.

Within a $/16$ network, the cache capacity distribution among upper and lower level caches will have big impact on the caching placement, content routing and the final delay performance. We define the "level factor" as the ratio between the size of an upper-level cache and the size of a lower-level cache. Figure 8 shows how the level factor affects caching performance within a particular /16 subnet. The optimal level factor is 32 which means that with the total cache capacity of 18,980 files (70% of the total unique files) in this case, one may put very few contents on the edge caches (450 files/cache) and put most of the content in the upper level caches (14,426 files/cache) and still get the best performance.

D. Performance of Hybrid Solution – MFLRU

The periodical proactive caching may not perform well if the content request patterns change dramatically between two updates. Such dramatic changes could be due to either large group size or long time intervals between two updates. Figure 9 shows the results for $T_1 = 24, T_2 = 24$. Since the time interval between two cache updates is large (24 hours), the MF-based proactive caching algorithm has bad performance while the hybrid MFLRU algorithm performs much better. To understand how LRU helps with reducing delay in our network model, we list the hit numbers for MF, MFLRU and LRU algorithms at both levels in Table III. It can be seen that although LRU performs worse than MF at the bottom level, it significantly outperforms MF at the upper-level. The reason is that the upper-level cache handles mostly the cold contents, while most of the popular contents are handled by the lower-level caches. Since it is hard to predict the popularity of cold content, the proactive strategies perform worse than the reactive strategies. That is the reason why the LRU perform much better in the upper-level. By tuning the size ratio between the MF and LRU portions in MFLRU algorithm, we can achieve shorter average delay than only giving all capacities to MF. In addition, to achieve the best performance, we can use different MF/LRU size ratios at upper and lower levels. For example, in the lower-level we can set the MF portion large to exploit its ability to estimate the popularity for the most popular and medium popular contents. In the upper-level, we set the LRU portion large to exploit its ability to adapt to the cold contents. Due to the limited space, we only present results from experiment where MF proportions at all levels are 0.6.

E. Impact of Cache Update Cost

We calculate the average user delay and the average cache update delay for both the *Basic* model and the *Update-cost-aware* model when they handle the same number of user requests. We set $\eta = 1$. We can see from Figure 10 that the average user delay increases as the update interval T_2

TABLE III: Hit Numbers at Different Levels for MF, MFLRU and LRU ($T_1 = 24, T_2 = 24$)

Method	# of Downloads from Server	Hit # in lower-level	Hit # in upper-level	Gain(%)
MF	9523	1489	149	—
MFLRU	8789	1533	839	6.28
LRU	9574	722	865	-1.37

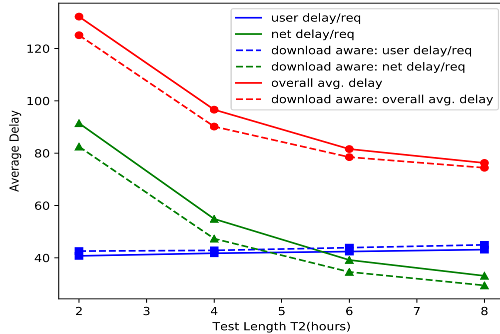

Fig. 10: Compare User Delay and Cache Update Cost between *Basic* and *Update-cost-aware* Models

increases, while the average cache update cost per user request decreases as T_2 increases. This is because the cache update cost is shared between all user requests between T_2. After we consider the cache update cost (dashed lines), the cache update cost decreases considerably while user delay increases are almost negligible. This suggests that our update-cost-aware algorithm can effectively control the cache update cost at the price of minor user delay increases. This user delay and cache update cost trade-off can be further fine tuned by changing the weight η. We will explore it further in our future work.

VII. Conclusion

In this paper, we developed a novel data-driven hybrid caching framework for hierarchical edge cache networks. It consists of fine-grained content popularity estimation obtained by mining the content interests of small user groups from user content access history data, periodical proactive optimization of content placement and routing to trade off between minimizing user delays and controlling content update costs, and LRU-based adaption to realtime content request patterns between two updates. Through extensive experiments driven by video request traces of real users, we demonstrate that our hybrid solution can adapt to long-term and short-term content popularity dynamics and outperform the state-of-art caching solutions by up to 30%.

References

[1] 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.

[2] C. Yuan, Y. Chen, and Z. Zhang, "Evaluation of edge caching/off loading for dynamic content delivery," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 11, pp. 1411–1423, 2004.

[3] A. Dabirmoghaddam, M. M. Barijough, and J. Garcia-Luna-Aceves, "Understanding optimal caching and opportunistic caching at the edge of information-centric networks," in *Proceedings of the 1st ACM conference on information-centric networking*. ACM, 2014, pp. 47–56.

[4] S. Ioannidis and E. Yeh, "Jointly optimal routing and caching for arbitrary network topologies," *arXiv preprint arXiv:1708.05999*, 2017.

[5] M. Dehghan, A. Seetharam, B. Jiang, T. He, T. Salonidis, J. Kurose, D. Towsley, and R. Sitaraman, "On the complexity of optimal routing and content caching in heterogeneous networks," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 936–944.

[6] K. Poularakis, G. Iosifidis, V. Sourlas, and L. Tassiulas, "Exploiting caching and multicast for 5g wireless networks," *IEEE Transactions on Wireless Communications*, vol. 15, no. 4, pp. 2995–3007, 2016.

[7] C. Yang, Y. Yao, Z. Chen, and B. Xia, "Analysis on cache-enabled wireless heterogeneous networks," *IEEE Transactions on Wireless Communications*, vol. 15, no. 1, pp. 131–145, 2016.

[8] K. Poularakis and L. Tassiulas, "On the complexity of optimal content placement in hierarchical caching networks," *IEEE Transactions on Communications*, vol. 64, no. 5, pp. 2092–2103, 2016.

[9] M. Gregori, J. Gómez-Vilardebó, J. Matamoros, and D. Gündüz, "Wireless content caching for small cell and d2d networks," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 5, pp. 1222–1234, 2016.

[10] E. Leonardi and G. L. Torrisi, "Least recently used caches under the shot noise model," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 2281–2289.

[11] L. A. Adamic and B. A. Huberman, "Zipf's law and the internet." *Glottometrics*, vol. 3, no. 1, pp. 143–150, 2002.

[12] M. Jamali and M. Ester, "A matrix factorization technique with trust propagation for recommendation in social networks," in *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 2010, pp. 135–142.

[13] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001, pp. 285–295.

[14] Z.-D. Zhao and M.-S. Shang, "User-based collaborative-filtering recommendation algorithms on hadoop," in *Knowledge Discovery and Data Mining, 2010. WKDD'10. Third International Conference on*. IEEE, 2010, pp. 478–481.

[15] A. A. Kardan and M. Ebrahimi, "A novel approach to hybrid recommendation systems based on association rules mining for content recommendation in asynchronous discussion groups," *Information Sciences*, vol. 219, pp. 93–110, 2013.

[16] Z. Lu, Z. Dou, J. Lian, X. Xie, and Q. Yang, "Content-based collaborative filtering for news topic recommendation." in *AAAI*, 2015, pp. 217–223.

[17] N. Pudota, A. Dattolo, A. Baruzzo, F. Ferrara, and C. Tasso, "Automatic keyphrase extraction and ontology mining for content-based tag recommendation," *International Journal of Intelligent Systems*, vol. 25, no. 12, pp. 1158–1186, 2010.

[18] Z. Liu, P. Nain, N. Niclausse, and D. Towsley, "Static caching of web servers," in *Multimedia Computing and Networking 1998*, vol. 3310. International Society for Optics and Photonics, 1997, pp. 179–191.

[19] J. Li, T. K. Phan, W. Chai, D. Tuncer, G. Pavlou, D. Griffin, and M. Rio, "Dr-cache: Distributed resilient caching with latency guarantees," in *IEEE INFOCOM*. IEEE INFOCOM, 2018.

[20] G. Carofiglio, L. Mekinda, and L. Muscariello, "Analysis of latency-aware caching strategies in information-centric networking," in *Proceedings of the 1st Workshop on Content Caching and Delivery in Wireless Networks*. ACM, 2016, p. 5.

[21] S. Shukla and A. A. Abouzeid, "Proactive retention aware caching," in *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*. IEEE, 2017, pp. 1–9.

[22] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, no. 8, pp. 30–37, 2009.

[23] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in neural information processing systems*, 2001, pp. 556–562.

[24] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, "Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies," *IEEE transactions on Computers*, no. 12, pp. 1352–1361, 2001.