Data-driven Approaches to Edge Caching

Guangyu Li, Qiang Shen, Yong Liu
New York University
Brooklyn, New York, USA

Houwei Cao
New York Institute of Technology
New York, New York, USA

Zifa Han, Feng Li, Jin Li
Huawei Technologies,
Nanjing, China

ABSTRACT

Content caching at network edge is a promising solution for serving emerging high-throughput low-delay applications, such as virtual reality, augmented reality and Internet-of-Things. The traditional caching algorithms need to adapt to the edge networking environment since old traffic assumptions may no longer hold. Meanwhile, user/group content interest as a new important element should be considered to improve the caching performance. In this work, we propose two novel caching strategies that mine user/group interests to improve caching performance at network edge. The static user-group interest patterns are handled by the Matrix Factorization method and the temporal content request patterns are handled by the Least-Recently-Used or Nearest-Neighbor algorithms. Through empirical experiments with a large-scale real IPTV user traces, we demonstrate that the proposed caching algorithms outperform the existing caching algorithms and approach the caching performance upper bound in the large cache size regime. Leveraging on offline computation, we can limit the online computation cost and achieve good caching performance in realtime.

CCS CONCEPTS

• **Networks** → *Network experimentation*;

KEYWORDS

Edge Caching, Data-driven

1 INTRODUCTION

The Internet traffic patterns have changed dramatically due to the constantly evolving network applications. The storage and network throughput requirements of today's 4K videos and Virtual Reality games have increased several thousand times over videos streamed over the Internet ten years ago. The network architecture and operation schemes that support these applications should also adapt to the new traffic patterns. In particular, Content Delivery Networks (CDN) face unprecedented challenges to reduce the backbone traffic and deliver high Quality-of-Experience (QoE) to users. As a simple solution to deal with the increasing traffic demand, more and more CDN servers are pushed towards the network edge. Compared with the servers deployed deep inside the core network, distributed CDN servers at network edge are equipped with much lower capacities to handle much smaller user populations. One immediate challenge is that the legacy caching algorithms built in CDN servers may not perform well when the cache capacity and user population scale down at the same time.

More specifically, a core CDN server operates according to the aggregated behaviors of all users served by it. Since such a server normally serves a large number of users, the aggregate content popularity distribution and temporal request patterns are stable. Simple caching algorithms, such as the Least-Recently-Used (LRU) and Least-Frequently-Used (LFU), can exploit such stability to achieve good caching performance. However an edge CDN server only serves a small user population, and operates according to the content popularity distribution and temporal request patterns generated by the local users. Caching on edge server is much more challenging than caching on core server due to the following reasons: 1) local distribution is highly fluctuating and unreliable because of the lower degree of multiplexing; 2) the temporal content request patterns are more sensitive to individual users' content consumption behaviors; 3) edge servers have much less caching and computation resources. For the above reasons, we know that the traditional caching algorithms, such as LRU and LFU, which require reliable content distribution, will not perform well.

For centralized caching, most of the studies assume certain arrival process of the contents, such as the shot noise model [10] and Zipf's distribution [1]. However, in the edge caching problem, the real content 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 caching algorithm. To adapt to the new network traffic environment, 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 of algorithms, e.g., [5, 11, 16]. Although various caching algorithm customizations have been proposed, one important new element—user interest—has not been considered. In the traditional cache architecture, no personal user interest is identified and exploited to improve caching performance. For caching at network edge, the traditional algorithms cannot cope with unreliable content popularity distribution and the highly fluctuating request arrival patterns. This gives rise of our data-driven approach of edge caching algorithms. In our proposed method, we focus on single-stage edge caching, with the "edge" defined at various sizes and hit ratio as objective measurement, while there are works focusing on multi-level caching optimization [6, 13] and other measurement metrics [4].

Our proposed caching algorithms rely on personalized user interest modeling methods. In the research line of user interest modeling and recommendation, lots of methods have been proposed. Some methods, such as matrix factorization [7] and collaborative filtering [15, 17], rely on user-content rating information. Other methods

NEAT '18, August 20, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5907-8/18/08…$15.00
https://doi.org/10.1145/3229574.3229582

employ content analysis and Natural Language Processing (NLP) techniques to solve the problem [8, 12, 14]. However, the existing recommendation algorithms cannot be directly adopted to solve the edge caching problems. 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 user interest profiling, while in caching problem each cache server serves a group of IP addresses and there may be multiple users behind each IP address (for IPv4 with NAT), so we need to profile group interest in practice.

The rest of this paper is structured as following. In section 2, our proposed MFLRU and KnnDyn caching strategies are presented. In Section 3, we conduct empirical experiments using content request trace of real video streaming users and introduce the baseline algorithms for comparison. In Section 4, we show the performance comparison between our proposed algorithms and the baselines. Results for different settings are also presented. Finally Section 5 concludes the paper.

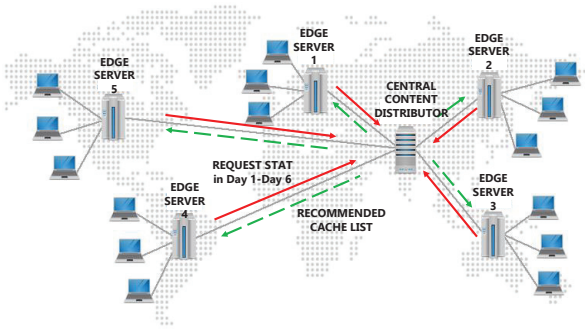

Figure 1: Example of information collection and processing in distributed CDN: the central CDN distributor collects content request statistics in the past 6 days from edge servers, and then generates the recommended cache lists for all edge servers in Day 7.

2 DATA-DRIVEN EDGE CACHING ALGORITHMS

2.1 Distributed CDN Architecture

As illustrated in Figure 1, in a distributed CDN, edge CDN nodes download content from a central content distributor and serve users in local regions. Each edge cache node runs data-driven caching algorithms which exploit not only the statistics of the content popularity and request patterns of its local users, but also such statistics of users from other regions. As will be shown later, richer data from other regions can help better mine user/group content interests to improve caching performance. To facilitate that, the CDN content distributor acts as the information hub. It collects content request statistics of users/groups from all edge CDN servers, and processes the collected information to estimate the potential user/group content interests. Based on the interest estimation, the CDN content distributor sends out the recommended cache list for each edge cache node. In order to reduce the communication and computation overheads, the information collection and interest estimation are only performed at scheduled time instants when the network traffic is light (e.g. midnight). An example for distributed CDN structure is shown in Figure 1.

Formally, the optimal caching problem is to find the best policy controlling the selection of cached contents in each time slot that maximizes the overall hit ratio in a given period. From this perspective, the traditional algorithms such as LRU and LFU are no more than a set of predetermined "rules" such as "always cache the most recently accessed contents" or "always cache the content accessed the most in a period". When the user population handled by a cache server is large, such simple policies can achieve good performance because of the stable content popularity distributions. In edge caching, the user population under each cache server is small, these rules become oversimplified, thus less effective. In real-world network traffic, the request arrival process of a content may be affected by various factors such as user interest and watching behavior, as well as content features such as genre and topic. To introduce our data-driven caching algorithms, we formulate a prediction problem that provides valuable input to caching decision:

Definition 1. *Given all the information obtained globally or locally in all the previous time slots, estimate the probability of request of certain content by a certain user/group in the next time slot.*

In practice, due to user data privacy issue, to what extent we can exploit the network traffic data is a question, and the answer varies under different privacy settings. In our discussion, we assume that as a CDN network operator has access to the necessary information to make predictions, such as user IP address, user id, content id and time stamp of a request, etc.

2.2 MF Enhanced LRU Caching Strategy (MFLRU)

LRU is a popular caching strategy for traditional caching problem because of its fast reaction to new request dynamics with low computational cost. One major shortcoming of using pure LRU method is that it fails to consider request patterns such as user/group similarity within data, while some recommender algorithms [3] can handle this part with medium cost.

In this paper, we adopt the widely used recommender algorithm — Matrix Factorization (MF) [7] and customize it for the edge caching problem. Given a sparse rating matrix $\mathbf{R}^{m\times n} = [r(i,j)]_{m\times n}$ that a set of m users generated over n contents, the goal is to estimate the missing rating (entry) in $\mathbf{R}^{m\times n}$. The MF model assumes that each user has an unknown latent vector $\mathbf{u_i} \in \mathbb{R}^d$ and each content has an unknown latent vector $\mathbf{v_j} \in \mathbb{R}^d$ ($d \ll min(m,n)$). The predicted rating of user i for content j is:

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

$\mathbf{v_j}$ can be interpreted as the latent features of content j, such as genre and topic; $\mathbf{u_i}$ can be interpreted as user i's preference on different features. The problem then can be formulated as finding matrices $\mathbf{U}$ and $\mathbf{V}$ such that $\mathbf{UV}^T$ approximates $\mathbf{R}$ with the least

approximation error:

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

In our caching problem, it's hard to find who are the users behind each IP address, so instead of treating each IP address as a user, we treat a group of IP as a superuser and estimate the superuser's preference to predict the accessing probability of the contents. Formally, given a set of IP addresses: $\mathbf{I} = \{IP_1, IP_2, ...IP_{|I|}\}$, some video contents: $\mathbf{C} = \{c_1, c_2, ...c_{|C|}\}$, we group $\mathbf{I}$ into small groups: $\mathbf{G} = \{g_1, g_2, ..., g_{|G|}\}$, where for certain group i, $g_i = \{IP_{i_1}, IP_{i_2}, ..., IP_{i_{|g_i|}}\}$. All users behind IP addresses in group i are served by the same edge content server. In a predetermined time window T, we define $\mathbf{S}(IP_{i_x}, c_j)$ as the frequency at which any IP address in group i accesses content j in the time window. 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 the ratings. We then use $r'(i,j)$ as the entries in our "group-content rating matrix" $\mathbf{R}$. 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 log of 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.

Assuming that we assign a cache server to each group with cache size of L contents, at any time t, we can use MF to get the estimated access probability ($r'(i,j)$) of any group to any content, then we generate a cache list for time t by selecting the top $|L|$ contents with the highest probability.

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. So we choose to update each group's cache list at scheduled time instants when the network traffic is light (e.g.every midnight), and keep all the cache lists unchanged before the next update. The details about this strategy are shown in Algorithm 1.

MF alone cannot adapt to content request pattern changes between two updates. On the other hand, LRU updates cache list upon each new request. To leverage the advantages of MF and LRU, we merge the cache lists produced by the two strategies by setting a weight α, which controls the proportion that MF-generated cache list takes in the final cache list. For example, α set to 0.4 means 40% of the contents in final cache list come from MF recommender and the rest come from LRU. If there is overlap between MF and LRU results, the system will go down the LRU list to fill in the cache.

Algorithm 1: Matrix Factorization Caching Strategy

Input : sliding time window T, fixed updating schedule $\tau = \{t_1, t_2, ...\}$, request record $Q = \{req_1, req_2, ...\}$, new request req_r=(time stamp,IP,content)
Initialize : cache list for each group $L_i = \emptyset, \forall i = 1, 2, .., |G|$, $\mathbf{R} = (r(i,j))_{|G|\times|C|} = \mathbf{0}_{|G|\times|C|}$

Function MF (**R**)
Input : sparse rating matrix $\mathbf{R} = (r(i,j))_{m\times n}$
Output : latent vectors $\mathbf{u_i}, \mathbf{v_j}$
while *True* **do**
  Collect new request req_r;
  $t = req_r.timestamp$;
  if $t \in \tau$ **then**
    for $i = 1, .., |G|, j = 1, .., |C|$ *and* $r(i,j)$ *is observed* **do**
      $r(i,j) = \sum_{x=1}^{|g_i|} S(IP_{i_x}, c_j) / \sum_{j=1}^{|C|}\sum_{x=1}^{|g_i|} S(IP_{i_x}, c_j)$;
    end
    $\mathbf{u_i}, \mathbf{v_j}$=MF($\mathbf{R}$), $\forall i = 1, .., |G|, \forall j = 1, .., |C|$;
    for $i = 1, .., |G|, j = 1, .., |C|$ *and* $r(i,j)$ *is not observed* **do**
      $r(i,j) = \mathbf{u_i}^T \mathbf{v_j}$;
    end
    L_i = contents with top-$|L_i|$ rating,$\forall i = 1, .., |G|$;
  else
    $Q = Q + \{req_r\}$;
    $Q = Q - \{req_i | \forall req_i \in Q$ that has timestamp $\leq t - T\}$
end

2.3 KNN-based Dynamic Caching Strategy (KnnDyn)

Due to the high computation cost of MF, MFLRU can only update user/group interest estimation at large time scales, e.g. hours, while the user/group interest and similarity between them can change at smaller time scales. Failure to capture the changes in user/group interests in a timely fashion may undermine the performance of the data-driven caching algorithms. We now resort to a low-cost collaborative filtering approach, namely K-Nearest-Neighor (KNN), to keep track of the fast changes in user/group interest. In this section, we propose KNN-based Dynamic Caching Strategy (KnnDyn).

In this approach, we use the real-time content request information of groups that are similar to the target group to guide the caching operation. There are two parts in this approach. First part is generating group similarity. This part requires the central server to collect the content request history from every edge server. Then we calculate the similarity matrix offline by using the Jaccard similarity based on history data, and the similarity between group i and group j is Sim_{ij}. The second part is computing the access score for each content currently in the cache. This part requires each edge server keeps a log of recently requested contents to be sent to other edge servers upon request. When an edge server serving group i needs to update the scores of contents currently in its cache, the server will collect the latest request logs from edge servers serving the K groups that are the most "similar" to group i (i.e., the K-Nearest-Neighbors of group i in terms of content interest) to obtain a recent

request matrix. Given the recent request matrix $\mathbf{R}^{M\times K}$, where M is the number of latest requests for a group, and K is the number of similar groups, the score for a content c in group i is:

$$Score(i,c) = \sum_{j=1}^{K} A(c,j)Sim_{ij} \tag{5}$$

where $A(c,i)$ is the occurrence of content c in the latest M requests for group i. After we calculate the scores of all contents in the cache, we sort them in the decreasing order of the score. Whenever there is a cache miss, the algorithm will evict the content with the lowest score. In practice, due to the overhead of computing the score and updating cache, we can not afford updating the whole cache for each new request. So we choose to update the cache whenever the agent receive a batch of requests. We name this part of the algorithm "KNN-based Dynamic Caching Strategy", as shown in Algorithm 2.

Algorithm 2: KNN-based Caching Strategy

Initialize: cache list for each group $L_i = \emptyset, \forall i = 1, 2, .., |G|$, request counter $cnt = 0$, batch size B, cache size CS.
while *True* **do**
  Collect the new request req_r ;
  $cnt = cnt + 1$;
  if $req_r \notin L_i$ **then**
    $L_i.append(req_r)$;
    $c_{min} = argmin_c(Score(c,i))$;
    $L_i.pop(c_{min})$;
  if $cnt == B$ **then**
    get the new recent request matrix $\mathbf{R}^{M\times K}$;
    $Score(i,c) = \sum_{j=1}^{K} A(c,j)Sim_{ij}, \forall c \in L_i$;
end

3 EXPERIMENTAL SETTINGS

3.1 Content Requests Trace

We use a real world content request trace collected by a major company providing video-on-demand service over the Interent in China. The provided contents range from TV shows and movies to live news and sports programs. The trace includes the sequence of requests generated from June 2014 to Septemer 2014 and from IPs located in several provinces in China. Each request also includes rich information (shown in table 1). In table 2 we show the basic statistics of the trace. From the trace we know in real world video-on-demand system, requests can be very sparse and there can be more than one user behind every IP address.

Meta Information	Example
IP Address	192.168.1.1
Request Time	2014.1.1.1.111111
Content Name	"Gone with the Wind.mp4"
Valid Watch Time	10000000ms
Watch Time	20000000ms
Location	Shanghai

Table 1: Information for Each Request in the Trace

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 2: Basic Statistics of the Trace

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 device, it may be hard to identify who are users behind each IP address. For this reason, we aggregate all requests of a group of IPs sharing a common prefix at certain length, treat the group as one super user, and use the number of accesses of one content by this group in a time window as the rating. By defining the prefix length k, we can group IP addresses into small groups such that IPs sharing the same first k bits are grouped together. 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 are served by the same CDN edge server[1] For this reason, we can assign a cache list to each of the IP groups. The longer the IP prefix length, the fewer the IPs falling into each group. 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 group if we want to push the CDN server close to edge, and a short prefix if we want to place the CDN deep in network core.

Based on prefix-based grouping, we now compare how the content popularity distribution generated by users in the same group is different from the global content popularity distribution generated by all users. To calculate the distance between global and local content popularity distribution, we select the top-50 most popular contents globally. We then calcuate the access frequency vector for those contents in the whole network (prefix=0), as well as for each local network (prefix length =1~20). At each prefix length, we calculate the distance as the average Euclidean distance between global and local access frequency vectors. Figure 2 shows that the distance increases significantly as prefix length increases (group size decreases).

3.2 Baselines

To demonstrate that our proposed methods can truly improve the caching hit ratio, we compare the performance against two widely-used methods:

- *Least-Recently-Uses (LRU)* [9]: Based on this strategy, the cache always discards the least recently used items first. If

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

Table 3: Hit Ratios (%) at Different Prefix Lengths and Different Cache Sizes

Cache Sizes (Content #)	Prefix Length=16				Prefix Length=18				Prefix Length=20			
	LRU	MFLRU	KnnDyn	FiF	LRU	MFLRU	KnnDyn	FiF	LRU	MFLRU	KnnDyn	FiF
50	10.12	11.49	**11.61**	23.99	12.05	13.05	**13.48**	20.43	14.71	15.41	**15.48**	23.99
100	12.80	14.95	**16.46**	28.92	15.2	16.77	**17.78**	25.09	17.58	**19.19**	18.74	26.25
200	16.67	20.55	**21.73**	34.24	19.13	**22.43**	22.19	28.61	20.93	**24.52**	22.24	28.82
500	24.01	**30.00**	29.68	41.25	25.70	**32.03**	29.76	31.96	27.22	**33.29**	28.95	33.67
1000	31.34	**38.39**	36.91	46.22	32.72	**39.94**	36.70	41.30	33.57	**41.00**	35.86	38.70

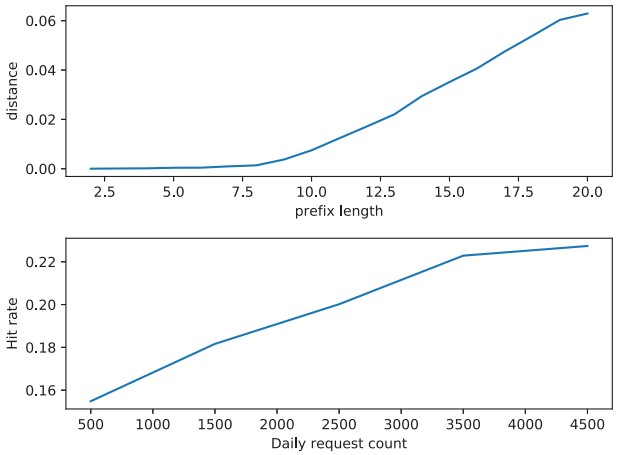

Figure 2: Upper: Difference between local content distribution generated by edge users and the global content distribution generated by all users when IP address prefix length increases. Lower: LRU hit ratios for groups with increasing request numbers

the content is popular at this moment, users have higher probability to request it. Thus the more popular the content is, the more likely the content will stay in the cache. Typically, the popularity of a new content increase dramatically at the beginning and achieve peak in very short time [10]. LRU is suitable to handle new and popular contents.

- *Farthest-in-Future (FiF)* [2]: Given the oracle of arrival time instants of all future content requests, the Farthest-in-Future algorithm evicts the item whose next request arrival time is the farthest among all items in the cache. FiF is the optimal reactive caching algorithm that minimizes the number of cache misses. Although the assumption of FiF is not realistic, it can be used to generate performance upper bound of the cache algorithms.

To test the performance of LRU, we randomly choose 1,500 groups with different sizes (measured by daily request count) and execute LRU algorithm to calculate its hit ratio when the cache size is set to be proportional to group size. As illustrated in the lower figure of Figure 2, LRU works very well for large groups, but its performance for smaller groups with few requests is much worse. This demonstrates the need for novel caching algorithms at network edge.

4 PERFORMANCE RESULTS

In this section, we present the perfomance comparison between our data-driven caching algorithms and the two baselines.

4.1 Optimal MFLRU Combining Weight

For MFLRU we need to choose the proportion α that MF results take in the final cache list. In order to choose the optimal α, we separate our data into three sets: training set, validation set and testing set. Then we get the user/group interest based on the training set, choose the proportion that achieves best performance in the validation set and get the final results in testing set. Figure 3 shows the effect of different proportion α on hit ratio. From the figure we can see that the best MF proportion increases as the cache size increases. For cache size=50, 100, the best MF proportion is around 0.2, while for cache size=500, 1000, the proportion is around 0.8. From this point on, we use $\alpha = 0.6$ for all MFLRU experiments.

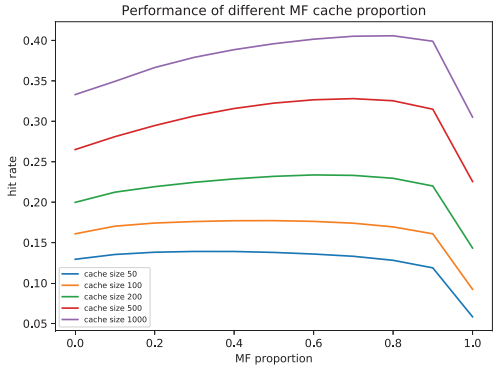

Figure 3: Impact of MFLRU combining weight on hit ratios.

4.2 Impact of Network Location of Cache Server

In our presumed setting, the network depth of a cache server is determined by the prefix length of the subnet it serves. In Table 3 we show the hit ratios generated by LRU, MFLRU, KnnDyn and FiF at the prefix length =16,18,20 for different cache sizes that range from 50 contents to 1,000 contents. We can make several observations from this table:

Table 4: Hit Ratios (%) at Different Group Sizes and Different Cache Sizes

Cache Sizes (Content #)	Small Groups			Medium Groups			Large Groups		
	LRU	MFLRU	FiF	LRU	MFLRU	FiF	LRU	MFLRU	FiF
50	24.8	**26.77**	25.83	17.73	**18.52**	25.57	12.05	**13.04**	20.43
100	27	**30.93**	27.61	21.87	**22.85**	27.2	15.2	**16.69**	25.09
200	31.03	**34.78**	31.52	24.76	**28.45**	29.52	19.13	**22.44**	28.61
500	—	—	—	31.56	**37.11**	34.22	25.71	**32.04**	31.96
1000	—	—	—	37.43	**44.12**	39.01	32.72	**39.95**	41.3

- Our data-driven approaches (MFLRU, KnnDyn) outperform LRU at all prefix lengths and cache sizes, and when the cache size is small (e.g., cache size = 50), KnnDyn slightly outperforms MFLRU. The reason is that KnnDyn is good at capturing the smaller timescale changes in user/group interests, thus performs better in predicting the most popular contents. MFLRU tends to estimate the long-term stable interests of users/groups and tries to cover all contents including those less accessed contents, thus MFLRU benefits from the increased cache size to accommodate less popular contents.
- MFLRU can approximate the optimal caching strategy (FiF) at all prefix lengths when the cache size increases to 1,000, while there are big gaps between LRU and FiF. When prefix legnth = 20, the group sizes (the number of requests from a group) become so small that FiF performance begins to drop, interestingly, MFLRU (41.00%) can even outperform FiF (38.70%). This is because MFLRU is not a pure reactive caching scheme, it proactively loads content at midnight. Content preloading does not consume traffic during day time, but can help increase hit ratio during day time.
- When the prefix length is fixed, MFLRU gains more improvement at larger cache sizes. This is because MF exploits more content request data to better predict content access probability than LRU. Consequently, the ranking list generated by MF better reflects the relative importance to keep contents in the cache, which in turn leads to higher caching hit ratio. The performance improvement is more pronounced when the cache size is large, because the accuracies for contents farther down the ranking list also matter.

4.3 Impact of Group Sizes

As mentioned previously, we measure the group sizes with the average number of requests per day by each group. When group size is small, LRU method may find it more difficult to follow the highly fluctuating arrival patterns of the content requests, while the MF method may be advantageous to exploit group interest patterns. To identify groups with different group sizes, we first draw the CDF of the group sizes and label the groups whose sizes are ranked in the bottom 33% as "small", the groups whose sizes are ranked in 33%-67% as "medium" and the groups whose sizes are ranked in the top 33% as "large". We then compare the hit ratios of these three types of groups. From Table 4, we can see that MFLRU makes more improvement when the group size is small. For example, when cache size=100, the relative gain of MFLRU over LRU for "small group" is around 15% ((30.93-27)/27), while the gain for "medium group" is around 4.5% ((22.85-21.87)/21.87).

5 CONCLUSION

In this paper, we first identified the new challenges of edging caching, then we proposed two novel data-driven caching algorithms that mine user/group interests to improve caching hit ratio. In our algorithms, the static user-group interest patterns are handled by the Matrix Factorization method and the temporal content request patterns are handled by the Least-Recently-Used or Nearest-Neighbor algorithms. Through empirical experiments with a large-scale real IPTV user traces, we demonstrated that the proposed caching algorithms outperform the widely used LRU algorithm and approach the caching performance upper bound in the large cache size regime. Our algorithms have controlled communication and computation overheads, can be adopted by edge caching nodes in distributed CDN.

REFERENCES

[1] Lada A Adamic and Bernardo A Huberman. 2002. Zipf's law and the Internet. *Glottometrics* 3, 1 (2002), 143–150.

[2] Laszlo A. Belady. 1966. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal* 5, 2 (1966), 78–101.

[3] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. 2013. Recommender systems survey. *Knowledge-based systems* 46 (2013), 109–132.

[4] Giovanna Carofiglio, Leonce Mekinda, and Luca Muscariello. 2016. 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, 5.

[5] Ali Dabirmoghaddam, Maziar Mirzazad Barijough, and JJ Garcia-Luna-Aceves. 2014. 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, 47–56.

[6] Stratis Ioannidis and Edmund Yeh. 2017. Jointly optimal routing and caching for arbitrary network topologies. *arXiv preprint arXiv:1708.05999* (2017).

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

[8] Ahmad A Kardan and Mahnaz Ebrahimi. 2013. A novel approach to hybrid recommendation systems based on association rules mining for content recommendation in asynchronous discussion groups. *Information Sciences* 219 (2013), 93–110.

[9] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. 2001. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE transactions on Computers* 50, 12 (2001), 1352–1361.

[10] Emilio Leonardi and Giovanni Luca Torrisi. 2015. Least recently used caches under the shot noise model. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2281–2289.

[11] Dong Liu, Binqiang Chen, Chenyang Yang, and Andreas F Molisch. 2016. Caching at the wireless edge: design aspects, challenges, and future directions. *IEEE Communications Magazine* 54, 9 (2016), 22–28.

[12] Zhongqi Lu, Zhicheng Dou, Jianxun Lian, Xing Xie, and Qiang Yang. 2015. Content-Based Collaborative Filtering for News Topic Recommendation.. In

AAAI. 217–223.

[13] Konstantinos Poularakis and Leandros Tassiulas. 2016. On the complexity of optimal content placement in hierarchical caching networks. *IEEE Transactions on Communications* 64, 5 (2016), 2092–2103.

[14] Nirmala Pudota, Antonina Dattolo, Andrea Baruzzo, Felice Ferrara, and Carlo Tasso. 2010. Automatic keyphrase extraction and ontology mining for content-based tag recommendation. *International Journal of Intelligent Systems* 25, 12 (2010), 1158–1186.

[15] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*. ACM, 285–295.

[16] Chun Yuan, Yu Chen, and Zheng Zhang. 2004. Evaluation of edge caching/off loading for dynamic content delivery. *IEEE Transactions on Knowledge and Data Engineering* 16, 11 (2004), 1411–1423.

[17] Zhi-Dan Zhao and Ming-Sheng Shang. 2010. User-based collaborative-filtering recommendation algorithms on hadoop. In *Knowledge Discovery and Data Mining, 2010. WKDD'10. Third International Conference on*. IEEE, 478–481.