Proc. of Sixth International Conference on COMmunication Systems and NETworkS (COMSNETS 2014)}, Bangalore, India, January 2014

A Dynamic Popularity-based Partial Caching Scheme for Video on Demand Service in IPTV Networks

Krishna Mohan Agrawal*, T.Venkatesh† and Deep Medhi‡
*Cisco Systems India Private Limited
Bengaluru, INDIA.
Email: kragrawa@cisco.com
†Department of Computer Science and Engineering
Indian Institute of Technology Guwahati, INDIA
Email:t.venkat@iitg.ernet.in
‡Department of Computer Science and Electrical Engineering
University of Missouri Kansas City, USA
Email:dmedhi@umkc.edu

***Abstract*—Caching video objects closer to the users in delivery of on-demand video services in IPTV networks reduces the load on the network and improves the latency in video delivery. Partial caching of the video objects is attractive due to the space constraints of the cache and also due to the fact that some parts of the video might be more popular than the others. However, fixed segment-based caching of videos does not take into account the changing popularity of the segments and the changes in the viewing patterns of the users. In this work, we propose a partial caching strategy that considers the changes in the popularity of the segments over time and the access patterns of the users to compute the utility of the objects in the cache. We also propose to partition the cache to avoid the eviction of the popular objects (those not accessed frequently) by the unpopular ones which are accessed with higher frequency. We measured the popularity distribution and ageing of popularity from two online datasets and use the parameters in simulations. Our simulation results show that the proposed caching scheme improves the byte hit ratio when compared to the LRU caching scheme both for static and dynamic object pools and ageing of popularity.**

I. Introduction

With recent advances in high-speed broadband network technologies, the deployment of Internet Protocol Television (IPTV) services has been increasing all over the world during the past few years. Recent market surveys show that the global consumer Internet video traffic is expected to be 69% of the consumer Internet traffic by 2017. It is also expected that the IPTV traffic alone will be 14% of the consumer Internet video traffic in 2017, up from the current figure of 9% in 2012 [1]. Among the various IPTV services, meeting the demands of the video-on-demand (VOD) service is challenging due to the asynchronous viewing patterns among the users, and the need to support VCR-like functions (play, pause, and seek) [2]. In an IPTV network, VOD services generate a large amount of unicast traffic from the Video Head Office (VHO) to the subscribers, consuming a significant amount of network resources. In such applications caching of the video objects that are viewed frequently, reduces the network traffic, and thereby, the cost of delivering the VOD services. Caching strategies are also important in reducing the latency experienced by the users and minimizing the stalling time during seek operations. In a typical IPTV architecture, the video objects may be stored in caches positioned closer to the subscribers either at the Digital Subscriber line Access Multiplexers (DSLAMs), Central Offices (COs), or the Intermediate Offices (IOs) [3].

This work was done when Mr.Agrawal was a student of IIT Guwahati.

Traditional caching strategies employed for the web objects have been shown to be ineffective for video objects primarily due to the large size of the video objects and the difference in the access patterns of these objects [4]. Further, owing to the large size of the video objects, caching entire videos may not be possible with a limited cache memory. Even if the cache were to store the full video, its utility may not be high if the user quits after watching only a part of the video. Partial caching of videos is also advantageous when the users watch only parts of the video more often than the entire video (also the case with skipped viewing) [5].

Some of the popular partial caching strategies proposed for caching multimedia objects at the proxies include *prefix caching*, which caches the initial part (prefix) of the video to reduce the start-up delay, *sliding-interval caching*, which caches different sizes of the video based on the temporal separation of requests, and *segment-based caching* which caches fixed length segments of the video based on the frequency of their access [4]. All the partial caching strategies have been shown to outperform the full object caching strategies since, the same area of cache could satisfy a larger number of requests. While prefix caching can reduce the start-up delay experienced by the user, it does not reduce the bandwidth consumed if the user plays different parts of the video. Sliding interval caching degenerates to full object caching if the access interval is longer than the playback duration. While a segment-based caching scheme is a generalized version of partial caching that can result in a better performance of the cache, it was shown in [6] that any pre-defined segmentation of the video will not work well if the segmentation does not consider the viewing pattern. All these partial caching strategies use a rigid approach in segmenting the video object and do not adapt to changes in the user's viewing patterns.

In a typical VOD system, the probability of accessing a video (or segment of a video) increases with its popularity.

Some caching strategies have been proposed in the literature taking into account the popularity of the video and not just the recency of its access (which is taken care of by the LRU scheme) or the frequency of its access (which is taken care of by the LFU scheme) [7]. However, it was rightly observed that most of the schemes assume that the popularity of the objects is known apriori and do not consider the change in popularity over time. Additionally, the user in a VOD system has the option to watch a video any number of times and can also decide to play only some parts of the video. Considering this behavior, it can be assumed that some parts of any video have higher popularity than the others [8]. Caching these popular segments would lead to a greater reduction in the bandwidth. Even if the video is partitioned into a fixed number of segments, tracking the changes in the popularity of the segments is an important requirement to design effective partial caching schemes. It was also observed that the popularity of the videos (or segments) can decrease rapidly with time [7]. While it can take some time for a movie, since its induction into the system to become popular, the popularity can also decrease with time. Any partial caching scheme based on popularity should consider the volatile nature of popularity to increase the effectiveness of the cache. Segment-based caching that does not track changes in popularity can lead to diminishing returns over time.

This work is motivated by the observation that neither a frequency-based technique like LFU nor a recency-based technique like LRU can give satisfactory performance for objects with volatile popularity (likelihood of future accesses) [5]. While the existing partial caching schemes fare better than full object caching schemes, the volatile nature of popularity and the viewing pattern should be considered to increase the effectiveness of the cache in a VOD system. In this work, we consider both the frequency and the recency of access for segment caching and propose a method to dynamically compute the utility of a segment stored in the cache with changes in the viewing pattern over time. We also observe that the objects in a VOD system have a wide variation in popularity over time. To protect the previously popular objects from being evicted by objects that see a sudden popularity, we propose to partition the cache into two caches. Each partitioned cache uses a different function to update the utility of segments present in its cache. We also determine the popularity distribution and ageing in popularity based on user ratings from two online datasets.

The rest of the paper is organized as follows. Section II discusses the literature on partial caching schemes for VOD systems. In Section III, we discuss the proposed caching scheme along with the motivation behind the work. Section IV presents the data from an analysis of movie ratings from two data sources and results from simulation of the proposed caching scheme and LRU. Section V concludes the paper.

II. Related Work

Though slightly dated, a good description of different caching schemes for a multimedia streaming proxy is given in [4]. As mentioned in the previous section, the traditional schemes for multimedia such as prefix-based caching, sliding interval caching, and segment-based caching suffer from the rigidity in segmentation and do not consider the volatile nature of the popularity of the video segments. Some of the earlier papers like [9] also addressed the issue of partial caching in multimedia delivery systems. The work in [7] argues the need to consider the volatile nature of the popularity of the videos in a VOD system while caching. It also demonstrates the volatile nature of popularity using a trace collected from a movie rental portal and developed a model to generate requests that considered popularity changes. The proposed caching scheme is shown to outperform both LFU and LRU schemes. However, it does not track the popularity of segments and it is complex to implement when there are dynamic changes in viewing patterns.

The work in [10] considers partitioning a video into segments and uses different segment lengths across the video. The paper assumes that the popularity of the segment is inversely proportional to its length. For variable length segmentation, the paper uses two approaches: *pyramid segmentation* in which the lengths grow exponentially and *skyscraper segmentation* in which the growth is gradual. Though the schemes use variable segmentation, the segmentation approach is rigid and does not consider the changes in the viewing pattern or priority over time. It also uses two caches, one for a fixed number of prefixes to lower the start-up latency and another for the variable-length segments. The first cache uses a simple LRU replacement policy while the second one evicts the segments based on priority.

The work in [6] proposes a scheme called *lazy segmentation* that caches the full object initially and then computes the length of the segment to be cached based on the average playtime of the video and the number of accesses. For each object in the cache, this scheme assigns a utility value based on the number of accesses observed over time and the average duration of the segment played. The segment with the smallest utility value is evicted. This scheme degenerates to the full object caching if the average playtime of the video is almost equal to the full length, or there is no history available. Although proposed in a different context of peer-to-peer systems, a partial caching scheme called *proportional partial caching* stores a small segment initially and then increases the size of the segment cached based on the number of accesses, the average length of the video played, and the current size already in cache [11]. However, this scheme also does not consider volatile popularity and increases the cached segment size very slowly over time.

The authors in [8] compute the popularity of segments based on the access pattern and propose to cache only the segments that are popular. However, this distribution is computed at the video server and it is difficult to estimate the parameters of the distribution at run-time at a cache. Recently, the work in [5] addresses the need for partial caching of user generated videos (e.g., YouTube videos) in a cellular network. The work computes the utility of each segment in a cache based on both the recency and frequency of accesses and adaptively computes the sizes of segments to be cached. It is perhaps the closest one to our work as far as the basic approach of computing the utility of segments in the cache is concerned. However, there are a number of differences. We partition the cache into two caches to address the volatility in popularity (which is elaborated further in Section III-B). Due to the overhead involved in handling fragmented memory blocks, we use fixed segments in our work. We compute the utility of each segment dynamically based on the request arrival rate and the average length of the segment played. The formulas we propose are different for each cache.

III. Proposed Caching Scheme

As discussed in Section I, the two main factors that motivated this work are 1) multimedia objects are associated with a popularity (utility) that changes with the time and 2) any segment-based caching scheme should consider the changes in the popularity in an eviction policy. In this section, we first describe the assumptions and motivate the need for a cache partitioning approach followed in this work. Then we propose the cache partitioning scheme and describe the procedure used to compute the utility of segments in a cache based dynamically on both recency and frequency.

A. Assumptions

We assume a hierarchical IPTV network (operated by a single telecom provider) for the delivery of VOD services in which the video objects are located at the VHO and the users are connected to DSLAMs. The network is supported by multiple levels of caches at the DSLAMs, the COs and the IOs. We do not assume any cooperation among the caches in this work. We assume that video objects are partitioned into fixed size segments. Based on conversations with system engineers at Netflix, we found that fixed segmentation with a moderate size is preferred over adaptive segmentation owing to memory management issues. One problem with fixed segmentation is that the segment boundaries may not align with those of heavily-viewed portions, leading to caching inefficiency. To avoid this problem, we measure the utility of a segment based on the number of bytes played from it, thereby capturing the utility of caching the segment. We assume that different segments of the same video might differ in their popularity. This is a reasonable assumption when the user has already watched the video earlier and is determined to watch some parts it again [12]. Changes in the popularity of a video at the segment level are also seen when the user can seek to watch different parts of the video (termed skipped viewing) so that some segments of the video tend to have a higher popularity than the others. In this work, we use a segment as a basic unit of a video object for caching and replacement. Henceforth, we use the words ‘segment’ and ‘object’ in a cache interchangeably. Following [5], a chunk is used as the basic unit of a video and a segment can be viewed as a collection of chunks.

B. Cache Partitioning Scheme

In LRU-based caching, objects that were popular (but not often watched) in recent times, tend to get evicted from the cache. This can lead to the eviction of popular objects when the temporal distribution of the accesses to an object (segment) is not uniform. Similarly, frequency-based caching schemes (LFU) do not perform well when the object pool is dynamic and the popularity of the objects in a cache decreases with time (termed ageing) [5]. Popularity of an object in a cache increases with time if the number of requests made for that object increases. Typically, in a VOD system, the popularity of some objects could decrease with time if the viewers do not see the video often. Thus, it is necessary to consider the last time when the object was accessed along with its popularity when making a cache replacement. The utility of the objects already in a cache should consider both the frequency and recency of accesses to account for the temporal changes in popularity. Typically, a simple popularity-based caching scheme considers the ranking of objects before caching and does not adapt to the ageing popularity. Caching schemes that capture the ageing popularity should compute the utility for objects in the cache continuously, based on both recency and frequency of the accesses. Since this computation becomes complex with an increasing number of segments, the objects in the cache should not be evicted too soon before the computation is effective in reflecting the temporal changes in popularity. New objects with a high popularity inserted in the cache should not evict older objects (with relatively lower popularity) whose utility reflects the temporal distribution of accesses.

Considering the aforementioned issues with caching schemes that ignore the volatile nature of popularity, we propose to divide a cache into two parts. $Cache_1$ (primary cache) is used to cache a segment that is accessed for the first time (on a typical cache miss) and $Cache_2$ (secondary cache) is used to store segments whose utility is already determined to be high (i.e., popular over a longer period of time). We use independent functions to determine the utility of segments in these two partitions because the eviction of segments from them should be handled differently. A segment that is accessed for the first time (or which is not in the cache) is cached in the primary cache. The utility of each segment in the primary cache is updated based on the recency and frequency of accesses made to it. Once the utility of a segment crosses a threshold (a user-defined parameter), it is moved to the secondary cache. In this way, all segments in $Cache_2$ have a larger utility, determined over a longer period of time, when compared to the short-term popularity of segments in $Cache_1$. The replacement of segments in both the partitions is handled independently according to the *smallest-utility-first* policy.

C. Dynamic Caching Scheme

A request is identified by an object ID, starting chunk ID, and the time at which the request arrives. Whenever a request for an object is made, the object is fetched in a segment-wise fashion. Depending on the state of segments (whether cached or not), our caching algorithm makes a decision as explained below.

Caching in $Cache_1$: If a requested segment is neither present in $Cache_1$ nor in $Cache_2$, then it will be cached in $Cache_1$. That is, $Cache_1$ is used to store segments whose utility is yet to be determined. Clearly, our approach is aggressive for segments whose utility is yet to be determined. Subsequently, as the number of requests for the segment increases, we determine its utility based on the average length of the segment played and the recency of the request.

Segments in $Cache_1$ are evicted based on the utility value determined by a utility function. Since the segment is inducted into the cache without any prior information about its popularity, the utility must be proportional to the average number of chunks played in the segment, and inversely proportional to the total number of chunks in the segment and its last access time.

We determine the utility of segments in $Cache_1$ as follows:

$$Utility_1 = \frac{N_{chunks\ played} * F_{recency}}{N_{chunks\ in\ segment} * N_{request}}, \qquad (1)$$

where

$$F_{recency} = \frac{1}{1 + \left(\frac{T_{current} - T_{last\ accessed}}{\beta}\right)}. \qquad (2)$$

Here, $N_{request}$ denotes the number of requests, $N_{chunks\ played}$ denotes the number of chunks played, $N_{chunks\ in\ segment}$ denotes the number of chunks in a segment, $F_{recency}$ denotes the recency factor, $T_{current}$ denotes the time when the request arrives, and $T_{last\ accessed}$ denotes the time when the segment was last accessed.

The utility value based on Eq.(1) decreases as the segment ages. This ensures that older segments are evicted from the cache. β is used to smoothen the recency factor, $F_{recency}$, based on the last access time, $T_{last\ accessed}$. This ensures that the utility of the segments does not change by a large amount in a short duration, which otherwise leads to a false prediction of the popularity of segments.

While the segment with the least utility is evicted, we do not evict a segment being played to avoid a cache miss. If all the segments in $Cache_1$ are being played and a new request arrives for the segment currently not in the cache, then the new request is dropped as there is no segment to replace. When a segment already in $Cache_1$ is requested, the data related to that segment such as, $N_{request}$, $N_{chunks\ played}$, and $T_{last\ accessed}$ are updated.

Promotion to $Cache_2$: For each request made for a segment in $Cache_1$, its utility is checked; if this crosses a predefined threshold (*THRESHOLD*), then the segment is marked for promotion to $Cache_2$. The marked segment is moved to $Cache_2$. If necessary, another segment in $Cache_2$ is replaced based on the utility values. $Cache_2$ is used to keep segments with a utility value that is greater than the *THRESHOLD*. A different function is used to compute the utility of segments in $Cache_2$ because the popularity of segments is already determined before promotion. In $Cache_2$, instead of using $F_{recency}$ while calculating the utility, we determine the probability of the next request for the segment. This gives a better prediction for the popularity of a segment based on the inter-arrival time, $1/\lambda$, of requests for that segment. As the popularity of a segment decreases, the inter-arrival time between the requests for the segment increases. The utility function for $Cache_2$ is given by

$$Utility_2 = \frac{N_{chunks\ played} * P_{next\ request}}{N_{chunks\ in\ segment} * N_{request}}. \qquad (3)$$

Here, $P_{next\ request}$ denotes the probability of the next request, given by

$$P_{next\ request} = \frac{1/\lambda}{\max\{1/\lambda, T_{since\ last\ request}\}}. \qquad (4)$$

where $T_{since\ last\ request}$ denotes the time since the last request.

Furthermore, to determine the correct utility of a segment in a cache, we propose to keep each segment in $Cache_1$ for a minimum time that is proportional to the segment size whenever it enters the system for the first time. This is particularly beneficial when the request arrival rate is very high. The minimum time for which a segment is kept in $Cache_1$ is the maximum of either the playback time of the segment or the playback time of the current request.

Algorithm 1 Dynamic Caching Scheme

```
for each request for an object; construct request for corresponding $Segment$
If $Segment$ is cached in $Cache_1$
Serve the $Segment$ and update utility of the $Segment$.
elseif $Segment$ is cached in $Cache_1$
Serve the $Segment$; update $Utility_1$
If $Utility_1$ is greater than $THRESHOLD$
promote the $Segment$ to $Cache_2$ replacing a segment (if needed).
Else Cache the $Segment$ in $Cache_1$ evicting a segment based on $Utility_1$.
Initialize/update utility of all segments in Cache.
```

IV. Performance Evaluation

We simulated the proposed caching scheme using a trace-driven simulation. We developed a traffic generator and a cache simulator in C++. A tree topology that typically depicts the hierarchical caching architecture in IPTV networks is used for simulations. The link capacities are not important as we do not consider network congestion and only study the performance of a stand-alone cache using the byte hit ratio. Based on our initial experimentations, we set $\beta = 10$ in Eq.(2) in our work. We study the impact of the popularity distribution and the ageing of the popularity on the performance of our caching scheme. When ageing is introduced, the popularity of objects and the object pool is changed over time. A video library of 20,000 objects is considered from the datasets discussed below since this collection represents movies with a reasonably large number of user ratings, and for which the popularity distribution is modeled as the MZipf distribution with the parameters estimated from real traces.

It was observed in [13] that the popularity in streaming videos deviates from Zipf and Zipf-like distributions. The deviation is due to the immutable nature of video objects and the fetch at-most-once behavior of users. The Zipf-Mandelbrot (MZipf) distribution, which uses a shift parameter q, in addition to the shape parameter α (used by the Zipf-like distribution) has been shown to model such a behavior and is a good fit for the popularity distribution of video objects [11]. The shift parameter of MZipf can also explain the flattened head observed in the log-log frequency plot of YouTube viewing patterns [14].

To obtain the values for the parameters of the MZipf distribution and to understand the changes in popularity, we first study movie ratings from two sites IMDb and `www.filmtipset.se`. While IMDb data is publicly accessible, the data from filmtipset.se is obtained from the site administrator. Both the datasets show that the movie ratings follow MZipf distribution (albeit with different parameters) and there is also evidence to support the claim that popularity changes with time (see the discussion below).

A. *Analysis of data from IMDb*

For each movie title in the IMDb database, we collected user ratings, number of voters, playtime, genre and the release date. We studied the popularity distribution of movies in IMDb. In IMDb, a user can rate a movie on a scale of 1 to 10. We first filter the movies rated by the users for analysis, as not all the movies are rated by the users. Out of a total of 277,935 titles, only 135,009 movie titles were rated by users. The popularity of a movie is calculated as the product of an average rating of the movie and the number of users rating the movie. Table I shows the popularity distribution of movies based on the number of users rating a movie. We find that the popularity distribution of movies rated by at least 1,000 users in IMDb follows MZipf distribution with α and q approximately equal to 0.95 and 58, respectively.

Minimum number of voters	Number of movies	Popularity distribution
100	47,658	MZipf(0.9980, 69.8404)
500	21,265	MZipf(0.9661, 63.1332)
1,000	14,816	MZipf(0.9427, 58.4949)
5,000	6,201	MZipf(0.8528, 42.5598)

TABLE I: Statistics for the movie ratings from IMDb

Minimum number of voters	Number of movies	Popularity distribution
1	12,830	MZipf(1.113, 21.436)
1,000	1,716	MZipf(0.9712, 14.0201)
5,000	792	MZipf(0.7859, 6.5406)

TABLE II: Statistics for movies released in 2009-2011

We see from the above table that the number of movies, which are rated by more than 1,000 users, is considerably low. This analysis suggests that the unpopular movies do not get rated by as many users as a popular movie gets rated. This also suggests that the utility of an object in a cache should be evaluated based on continuous requests and static ratings cannot be used for cache replacement.

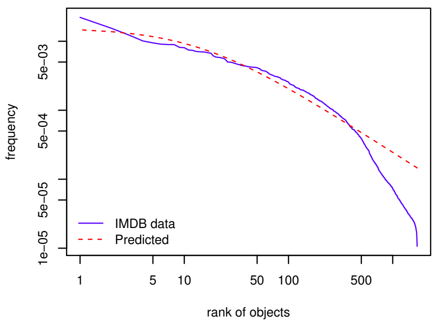

Fig. 1: Popularity distribution of movies released during 2009 and 2011 (rated by at least 1,000 users)

Since users frequently access recent movies, we focus on the popularity distribution of recently released movies. Table II summarizes the popularity distribution of all movies released from 2009 to 2011. We find that the popularity distribution of movies released from 2009 to 2011 in IMDb (rated by at least 1000 users) follows MZipf distribution with α and q equal to

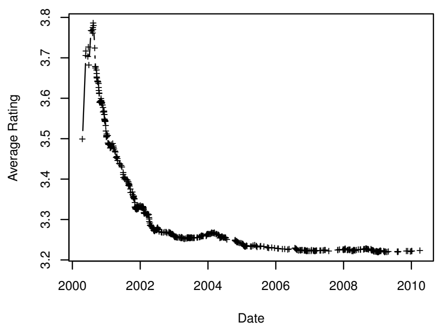

Fig. 2: The average rating of an arbitrary movie measured over time

0.97 and 14.02, respectively. Fig. 1 shows the log-log plot of the popularity distribution of movies released from 2009 to 2011 that had at least 1,000 voters along with the predicted popularity distribution. Note that of the 12,830 movies released between 2009 and 2011, only 1,716 movies were rated by at least 1,000 users, which is quite low.

B. *Analysis of data from www.filmtipset.se*

We have analyzed the data from *www.filmtipset.se* that contain ratings for 91,576 movies. The data contain the ratings given by every user on a scale of 1 to 5 for each movie. It also contains other information related to the genre, actors, and director of a movie. Fig 2 shows the evolution of an average rating of an arbitrary movie from the time it was released. We see that the average rating (which also represents the popularity) of a movie increases initially and then decreases over time before stabilizing to a constant value after a large number of users rated the movie. The evolution of rating over time is not measurable with the ratings on IMDb. We use this evolution of popularity as an indication for the volatile nature of the popularity of movies.

In order to measure the popularity distribution of the movies, we sorted the list of movies based on the number of users who rated the movie. Movies rated by at least 100 users are considered for the analysis. Fig. 3 shows the log-log plot of the number of movies vs the number of users rating them. We observed that there were a large number of movies rated by a few users and the number of movies rated by more than 1,000 users was quite small. This is because unpopular movies do not get rated by many users, while a popular movie gets rated by many users. The popularity of a movie is determined as the total sum of ratings given by voters. Table III summarizes statistics of our analysis. The total number of movies rated by at least 100 voters was 32,774. Even for this data, we found that the popularity follows MZipf distribution. Fig. 4 shows the log-log plot of the popularity distribution of movies rated by at least 100 voters along with the predicted MZipf distribution with $q = 286.5$ and $\alpha = 0.5862$.

Our analysis confirms that using a Zipf-like distribution to model the popularity of movies or video objects would result in a significant error. In the log-log scale, the Zipf-like distribution appears as a straight line, which can reasonably fit most of the popularity distribution except the left-most

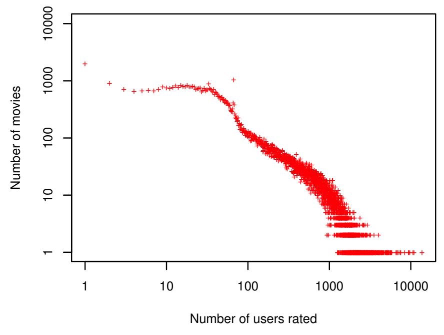

Fig. 3: Number of movies rated by the users

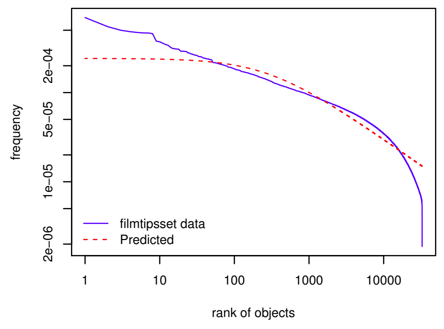

Fig. 4: Popularity distribution of movies rated by at least 100 users on *www.filmtipset.se*

part. A Zipf-like distribution would largely overestimate the popularity of objects with higher ranks. This region is the most important for caching as the highly ranked objects are the potential candidates for caching.

Minimum number of voters	Number of movies	Popularity distribution
100	32,774	MZipf(0.5862, 286.5)
200	24,485	MZipf(0.4463, 46.1245)
500	12,869	MZipf(0.3511, 3.6603)

TABLE III: Statistics for movies rated by users on filmtipset.se

C. Simulation of the Proposed Caching Scheme

We simulated the proposed caching scheme with MZipf distribution for object popularity using several values of q and α. We also studied the scheme with ageing popularity, in which the popularity of objects in the object pool degraded with time and new popular objects were inserted frequently into the object pool. We found the performance of our proposed algorithm to be much better than LRU. Here, we present results for the MZipf distribution with $q = 20$ and α ranging from 0.5 to 0.9. The request arrival is assumed to follow Poisson distribution with a mean inter-arrival time of 120 seconds. Video objects are 100 minutes long (based on the average movie length observed in the datasets) with a playback rate of 2 Mbps, which amounts to an approximate size of 1.5 GB for each movie. All requests were initially for the starting point of a movie, but they ended at different times. The time

Parameter	Default value
Simulation Time	24 days
Cache warm-up time	2 days
Ratio of $Cache_2$ and Total cache	0.1
Average request arrival time	120 seconds
Popularity distribution	Mzipf (0.5, 20)
Segment size in uniform segmentation	10 mins playtime
Threshold	0.45

TABLE IV: Default Values for Simulation

when a user quit is assumed to follow a trimodal distribution with a probability of 0.4, 0.2, and 0.4 for a playback time of 0 to 20 minutes, 20 to 80 minutes, and 80 to 100 minutes, respectively [12]. The segment size is assumed to be equivalent to 10 minutes of playback time. All the results were computed with a 95% confidence interval. These are not showed in the figures as the interval was very small, within 4% of the mean values plotted.

Table IV gives the set of default parameters used in simulation unless otherwise specified. We used byte-hit ratio, defined as the total number of bytes served to the total number of bytes requested as the cache performance metric. A higher byte-hit ratio indicates a reduction in the core network load.

1) Simulation with static object pool: For this case, the set of videos remains constant, i.e., the ranking of objects is constant during the simulation. Figs. 5 and 6 show the performance of the proposed scheme compared with LRU for $\alpha = 0.5$ and $\alpha = 0.9$, respectively. We notice that the proposed scheme performs much better then LRU when the value of α is low. The performance of LRU approaches that of our proposed algorithm, as the value of α increases. This suggests that our proposed algorithm gives better results than LRU when the requests are spread over a wide range of popular objects, than when they are concentrated over a small subset of popular objects. This is because of the partitioning of the cache, which requires computing the utility of each set in a different manner.

2) Simulation with dynamic object pool: We now study the performance of the proposed caching scheme when the set of videos changes and/or the popularity of videos changes with time. We first explore how ageing of popularity affects the caching scheme. In this case, a limited number of objects are added periodically to the top 100 list, with an effect that some objects move down the ranking within a short duration of time. This is to mimic the scenario wherein the movies, which were popular at a certain time, are no longer popular at another time while new movies become instantly popular. The number of objects in the object pool remained the same and no new object was inserted into the object pool during simulation. Fig. 7 shows the performance of our proposed algorithm and LRU in this case. We observed that our algorithm performed better than LRU and the increase in performance was around 10% at all times.

Next, we explored the effect of introducing new objects into the system and the ageing popularity of existing objects. To simulate this case, for each day in the simulation, the ranks of all the objects were decreased by X, which may be interpreted as a measure for the rate of ageing. In order to maintain the overall popularity distribution, the bottom X objects with rank from $N - X$ to N were removed while new objects with rank

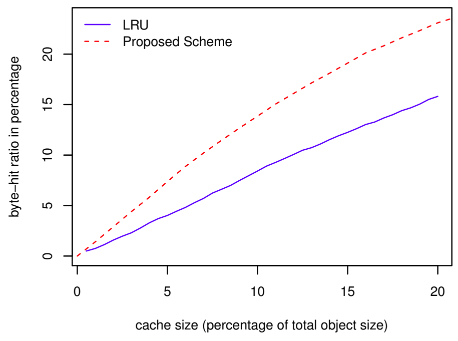

Fig. 5: Performance for static pool with MZipf(0.5,20)

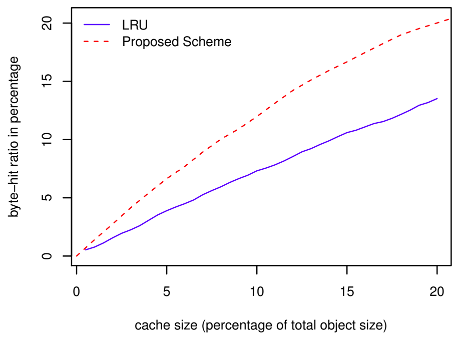

Fig. 8: Performance with ageing for MZipf(0.5,20)

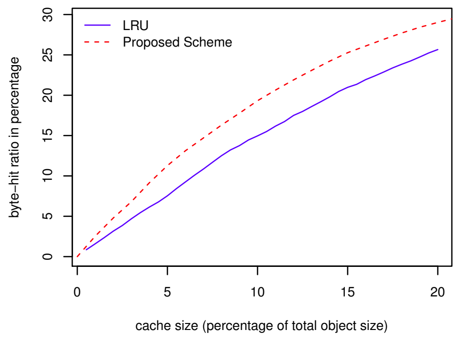

Fig. 6: Performance for static pool with MZipf(0.9,20)

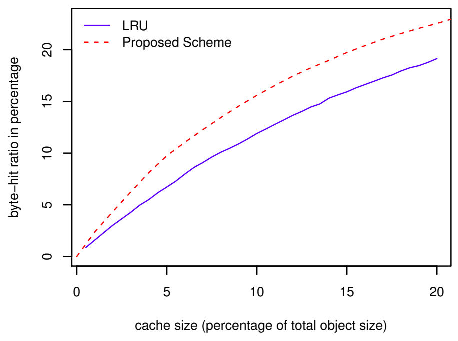

Fig. 9: Performance with ageing for MZipf(0.9,20)

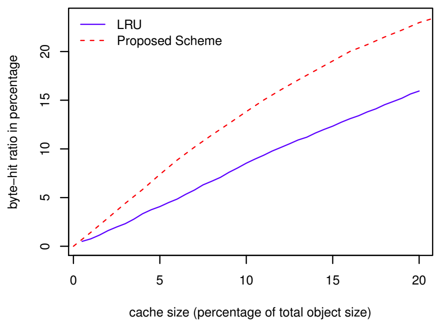

Fig. 7: Performance of proposed caching scheme with dynamic object pool

from 1 to X were inserted into the object pool. Equivalently, each object remained for $\frac{N}{X}$ days in the object pool over its active lifetime. The total number of objects in the object pool remained constant. In our simulation, we kept the rate of ageing to 200 objects per day, which is equal to 1% of the total number of objects in the pool.

Fig. 8 and 9 show the performance of our algorithm and LRU for different values of α with ageing and the dynamic object pool. We fixed the ratio of size of $Cache_2$ to total cache size at 0.1. We see from the results that in all the cases, our algorithm performs better than LRU. The increase in performance relative to LRU was upto 10%. Also, we noticed that LRU approached the performance of our algorithm as the value of α increased. This is because our scheme dynamically updated the utility of segments in the cache and by using two partitions, our scheme allowed the utility computation to get stabilized before replacement.

3) Effect of ageing rate: We also studied the impact of the rate of ageing on the performance of the caching schemes. For this study, we set $\alpha = 0.9$ and $q = 10$. We varied the rate of ageing from 100 objects per day to 200 objects per day, which represents 0.5% to 1.0% of the total number of objects in the pool. Fig. 10 shows the performance of the proposed caching scheme and LRU for different ageing rates. We see that the gain in byte-hit ratio decreased with an increase in the rate of ageing. This may be explained from the fact that the constant churn requires the cache content to be updated continually. On the other hand, in the static case, i.e., when there is no ageing, the cache content gradually saturates to a fixed set of objects as requests arrive for the same objects. We conclude from the result that our algorithm is constrained by the rate of ageing and the benefits of caching decreases with the ageing rate.

We also studied the impact of the relative sizes of the cache partitions ($Cache_1$ and $Cache_2$) on the performance of the proposed scheme. We kept the total size of the cache fixed and varied the ratio of $Cache_2$ to the total cache size from 0.1 to 0.3. Fig. 11 shows the performance for various ratios. We see that for a small cache size, the performance increased by a small amount with an increase in the size of $Cache_2$. However, for larger sizes of cache, the size of $Cache_2$ did not effect the performance of the caching algorithm. This is due to the reason that when the cache size is large, the segments above certain popularity are already promoted and the segments, which are less popular (but are promoted because of large cache size) do not contribute to the performance. That is why when the cache size is small, the impact of the change in the size of $Cache_2$ is seen in the performance of a cache but not when the cache size is large. Thus, a moderate ratio can be used for the partitioning of caches. This ratio can be obtained from data already collected or can be calculated experimentally. Also, due to fetch-atmost-once behavior of users and the large number of unpopular objects, the fraction of popular objects almost remains constant over time. Any small change in the fraction of the popular objects does not affect the ratio noticeably. Thus, we can use a moderate ratio to partition the cache to obtain maximum benefits.

We also studied the effect of the segment size. For different segment sizes, we studied the improvement in the byte-hit ratio of the proposed scheme compared to the LRU. We noticed that as the segment size increased, the byte-hit ratio decreased because only a smaller number of segments can be stored. However, the drop in the performance is not large. This convinced us that any moderate segment size is sufficient for the partial caching scheme.

V. Conclusion

In this work, we proposed a dynamic cache partitioning and caching scheme for caching video segments in a VOD system. The proposed scheme partitions the cache into two caches to avoid eviction of popular objects due to a sudden increase in the number of accesses for less popular objects. The proposed caching scheme considers both recency and frequency of accesses to overcome the problems with simple LRU and LFU schemes. The proposed caching scheme keeps track of the average duration for which the video is played and over time, protects the popular objects from being evicted (due to ageing), and new objects introduced into the system. Our simulation results showed that the proposed caching scheme works well when new objects with greater popularity are introduced into the system and likewise, when the popularity of older objects decreased over time. Partitioning of a cache could be also used to isolate popular content in one category/genre contending for cache memory with popular content of another category/genre.

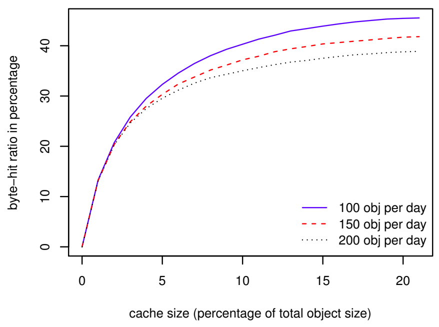

Fig. 10: Plot comparing performance for different ageing rates

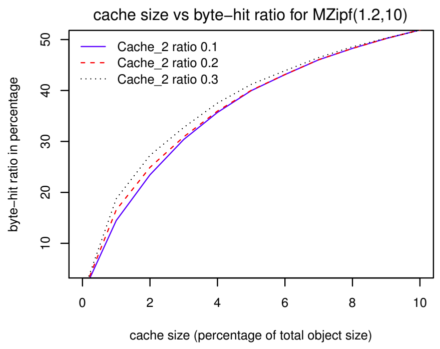

Fig. 11: Performance of caching scheme with different $Cache_2$ sizes

Acknowledgment

The authors would like to thank Mr. Magnus Hoem of www.filmtipset.se for allowing access to their data set.

References

[1] "Cisco visual network index: Forecast and methodology 2012-2017," http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf.

[2] S. Zeadally, H. Moustafa, and F. Siddiqui, "Internet protocol television (IPTV): Architecture, trends, and challenges," *IEEE Systems Journal*, vol. 5, no. 4, pp. 518–527, 2011.

[3] B. Krogfoss, L. Sofman, and A. Agrawal, "Caching architectures and optimization strategies for IPTV networks," *Bell Labs Technical Journal*, vol. 13, no. 3, pp. 13–28, 2008.

[4] J. Liu and J. Xu, "Proxy caching for media streaming over the Internet," *IEEE Communications Magazine*, vol. 42, no. 8, pp. 88–94, 2004.

[5] U. Devi, P. Ramana, M. Chetlur, and S. Kalyanaraman, "On the partial caching of streaming video," in *Proceedings of International Workshop on Quality of Service (IWQoS)*, 2012.

[6] S. Chen, H. Wang, X. Zhang, B. Shen, and S. Wee, "Segment-based proxy caching for Internet streaming media delivery," *IEEE Multimedia*, vol. 12, no. 3, pp. 59–67, 2005.

[7] D. De Vleeschauwer and K. Laevens, "Performance of caching algorithms for IPTV on-demand services," *IEEE Transactions on Broadcasting*, vol. 55, no. 2, pp. 491–501, 2009.

[8] J. Yu, C. Chou, X. Du, and T. Wang, "Internal popularity of streaming video and its implication on caching," in *Proceedings of 20th International Conference on Advanced Information Networking and Applications*, 2006, p. 6.

[9] J. M. Almeida, D. L. Eager, and M. K. Vernon, "A hybrid caching strategy for streaming media files," in *Proceedings of Multimedia Computing and Networking*, Jan. 2001.

[10] K.-L. Wu, P. S. Yu, and J. L. Wolf, "Segment-based proxy caching of multimedia streams," in *Proceedings of the 10th international conference on World Wide Web*, 2001, pp. 36–44.

[11] M. Hefeeda and O. Saleh, "Traffic modeling and proportional partial caching for peer-to-peer systems," *IEEE/ACM Transactions on Networking*, vol. 16, no. 6, pp. 1447–1460, 2008.

[12] "Video heatmaps," http://www.wistia.com/index.html.

[13] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: evidence and implications," in *Proceedings of IEEE INFOCOM*, Mar 1999, pp. 126 –134.

[14] M. Cha, H. Kwak, P. Rodriguez, Y. yeol Ahn, and S. Moon, "I tube, you tube, everybody tubes: Analyzing the worlds largest user generated content video system," in *Proceedings of the 5th ACM/USENIX Internet Measurement Conference*, 2007.