Cache-centric Video Recommendation: An Approach to Improve the Efficiency of YouTube Caches

Dilip Kumar Krishnappa and Michael Zink
University of Massachusetts Amherst
151 Holdsworth Way
Amherst, MA, USA
krishnappa|zink@ecs.umass.edu

Carsten Griwodz and Pål Halvorsen
Simula Research Laboratory
Lysaker, Norway
and University of Oslo
Oslo, Norway
griff|paalh@ifi.uio.no

ABSTRACT

In this paper, we take advantage of the user behavior of requesting videos from the related list provided by YouTube and the user behavior of requesting videos from the top of this related list to improve the performance of YouTube's caches. We recommend a related list reordering approach which modifies the order of the videos shown on the related list based on the content in the cache. The main goal of our reordering approach is to push the contents already in the cache to the top of the related list and push non-cached contents towards the bottom, which increases the likelihood that the already cached content will be chosen by the viewer. We analyze the benefits of our approach by an investigation that is based on two traces collected from an university campus. Our analysis shows that the proposed reordering approach for related list would lead to a 2 to 5 times increase in cache hit rate compared to an approach without reordering the related list. The increase in hit rate would lead to a 5.12% to 18.19% reduction in server load or back-end bandwidth usage. This increase in hit rate and reduction in back-end bandwidth reduces the latency in streaming the video requested by the viewer and has the potential to improve the overall performance of YouTube's content distribution system. An analysis of YouTube's recommendation system reveals that related lists are created from a small pool of videos, which increases the potential for caching content from related lists and reordering based on the content in the cache.

Categories and Subject Descriptors

H.5.1 [**Multimedia Information Systems**]: Video

General Terms

Measurement, Performance

MMSys'13, February 26 – March 1, 2013, Oslo, Norway

Keywords

Caching, YouTube, Recommendation

1. INTRODUCTION

YouTube is the world's most popular Internet service that hosts user-generated videos. Each minute 72 hours of new videos are uploaded to YouTube. Viewers can choose from hundreds of millions of videos and over 4 billion hours of videos are watched each month on YouTube. According to [10], in 2011, YouTube had more than 1 trillion views or around 140 views for every person on Earth. Consequently, these numbers lead to a huge amount of network traffic, and Google (the owner of YouTube) maintains substantial infrastructure to provide reliable and well-performing video streaming service. For example, according to [3], on August 27 2012 at 8PM, the United States YouTube traffic made up more than 60% of the global Internet traffic. Google's approach to tackle these challenges is a network of caches that are globally distributed. With the aid of the caches, latency on the viewer's side and overall network utilization can be reduced.

Unfortunately, effective caching is much harder in the case of YouTube in comparison to other video streaming services like Netflix or Hulu. This is caused by several facts. First of all, services that offer professionally produced content like movies and TV shows provide an online library on the order of several 10,000s of titles, a number that is much lower compared to the hundreds of millions of videos offered on YouTube. Second, the providers of purely professionally produced content determine when new content will be made available and, thus, can much better schedule the distribution of content into caches.

YouTube's dilemma, in terms of caching, is the fact that the popularity distribution of videos is a long-tail distribution. For the large number of videos in YouTube, this means that every cache that is limited to storing a small portion of the total number of videos will essentially perceive accesses to the tail as random choices, impossible to distinguish from a uniform distribution. This characteristic has already been shown in earlier studies. For example, in our trace-based characterization of YouTube traffic in a campus network [21], 70% or more of the overall videos are only requested once within 24 hours. In [13], similar results are obtained by a global analysis of metadata made available by YouTube. This characteristic leads to the fact that many of the requested videos will not be available in a cache. This problem has been officially recognized by YouTube [9], and

Figure 1: This screenshot shows a YouTube video page and the location of the related video list on that page.

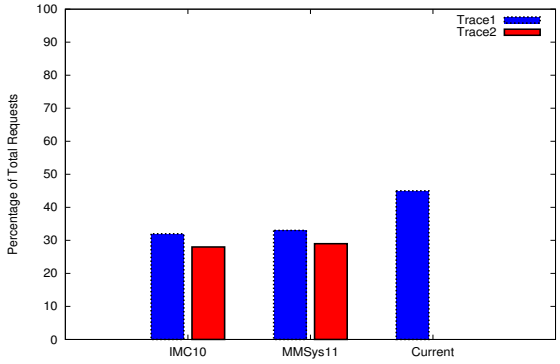


Figure 2: Percentage of videos selected from related video list as reported in earlier studies (IMC10 [19], MMSys11 [17]) and obtained from an analysis of the trace used in this paper.

the current approach is to increase the number of caches that serve content from the tail and fill these caches through prefetching.

In this paper, we propose a different caching approach that makes use of YouTube's recommendation system to increase the efficiency of its caches, since it is impossible to predict what a viewer is most likely to watch next. The general idea is to influence the video selection behavior of a viewer based on the content of the cache that serves the viewer. Each YouTube video web page offers a list of recommended videos, next to the video a viewer has selected (see Figure 1). In YouTube's own terms, this is described as the *related video list*. Earlier studies [19, 17] have already shown that users make significant use of the related video list. This means that, after watching the initially selected video, the viewer chooses to watch a video that is offered on the related video list next. Figure 2 shows a comparison of these result that were derived from five different data sets. It is important to mention that the results from [19] are obtained from two data sets of global meta information (taken at different points in time) that is offered by YouTube for each individual video. The results for [17] are obtained from two network traces taken at a campus gateway. From these results, we conjecture that both, on a global and regional level, $\sim$ 30% or more of video requests are made by the viewer by selecting a video from the related video list.

To exploit the fact that viewers frequently choose videos from the related list, we present an approach in which the cache modifies the related video list that is sent from the server through the cache to the client. This related list modification is based on a stable sort [18] the cache performs on the list of related videos that uses cache hit as its primary sorting criteria. The set of cached videos are moved to the top of the related list, while the set of uncached videos forms the rest. The order inside each set remains unchanged. The cache changes the order of the related video list because, as we show in Section 2.1, it is more likely that a viewer will select a video from the top of the related video list.

To investigate the feasibility of our approach, we perform an analysis on the chains created by user behavior in selecting YouTube videos from the related list. Chains are created when a user consecutively selects videos from the related list until she changes to choose a video by other means (e.g., a search). We also perform a PlanetLab based study to analyze the global behavior of loop creation. Loops are created when a user selects videos from the related list consecutively until she selects the initial requested video again. This selection is either based on the position of the video on the related list, or randomly chosen from the related list. Our results show that YouTube creates its related list from a small pool of videos which leads to lot of potential to related list caching and reordering based on the content in the cache.

In addition, we perform a measurement study from our campus network to determine the origin of videos that are requested from the related list. The results of this study indicate that YouTube uses a three-tier cache hierarchy to serve videos and that the origin is chosen in a way that achieves load balancing between the different levels in the caching hierarchy. Preference to the top videos of the related list for a requested video is not given. By using our approach of the related list reordering at the cache, we recommend changing the behavior of the caching technique to take advantage of user behavior in selecting a video from the top order of the related list by serving the videos from the nearest cache to reduce latency in serving the videos to the user.

To evaluate the proposed related list reordering approach, we perform a study based on YouTube traces obtained from a university campus network. Based on these traces, we show that first, users indeed choose top ranked videos from the related video list and second, with our proposed approach, the cache hit rate increases from about 2 to 5 times the original hit rate. This increase in cache hit rate due to reordering of related list reduces the bandwidth consumption on the uplink from the level 1 cache to higher level caches or the server by 5.12% to 18.19% and reduces the latency in serving the videos requested to the user.

The remainder of the paper is structured as follows. In Section 2, we present the data sets we use to analyze our new caching approach, and analyze the potential influence of the related list on the viewers video selection. In addition, we perform a measurement study to determine the origin of initially requested videos and videos that were chosen from the related list. Section 3 presents our cache-centric video

Trace file	T1	T2
Duration	3 days	3 days
Start date	Feb 6th 2012	Jan 8th 2010
# Requests	105339	7562
# Videos with "related_video" tag	47986	2495

Table 1: Trace characteristics

recommendation approach and evaluates that approach with the aid of trace-based simulations. We discuss the results of our analysis in Section 4, and present related work in Section 5. Section 6 concludes the paper.

2. THE IMPACT OF VIDEO RECOMMENDATION

YouTube's related video list, shown in Figure 1, is a way to influence the decision of viewers on which video to watch next. When a viewer selects a video, a list of recommended videos are offered to the viewer on the same web page (in YouTube jargon defined as *"watch page"*), and the viewer may choose a video from that list.

It has been shown in related work that the related video list actually influences the viewer's video selection [19, 17]. Such behavior, causing the selection of a video through the related video list, is also defined as click-through rate. It is specified as the probability that a user who views item i will click on item j from its related video list. It is our goal to investigate if the related video list can be used to improve video distribution and delivery within YouTube. But before we try to answer this question, we investigate how related lists influence what a viewer is watching.

2.1 Data Set Analysis

For the analysis of our proposed cache reordering approach we analyze two network traces obtained from monitoring YouTube traffic entering and leaving a campus network. The monitoring device is a PC with a Data Acquisition and Generation (DAG) card [1] which can capture Ethernet frames. The device is located at a campus network gateway, which allows it to see all traffic to and from the campus network. It was configured to capture a fixed length header of all HTTP packets going to and coming from the YouTube domain. The monitoring period for these traces was 72 hours each, and the general statistics are shown in Table 1.

In trace T1, 47,986 video requests had the **related_video** tag out of total 105,339 requests ($\sim$ 45%), which indicates that this video was selected from a viewer by choosing it from the related video list. In the case of trace T2, the numbers are significantly lower since the trace was taken during winter break when the student population on campus is much lower, i.e., $\sim$ 33% of the 7562 requested videos are selected from the related video list.

To investigate whether users choose videos from the top of the related video list with higher probability, we further analyze data from these traces to determine the position of the related video selected by the users. We determine the position of a video that was selected by a viewer from the related video list as follows. We take the related video list for video A requested by a viewer and check if video B, requested by the same user, is in video A's related video list. If so, we determine the position of video B in video A's related video list. Figure 3 show the results of this analysis for our trace (Trace T1), as well as the trace data used by Khemmarat et al. in [17] (Trace T2). Both traces show that about 50% of the viewers select an entry from the top five related videos. For the top ten videos, this number increases to over 70%. This observation confirms our assumption about YouTube user behavior, that a user (potentially out of laziness) usually selects videos ranked higher on the related video list independent of the video content (but we assume also that the related list is sorted by relevance).

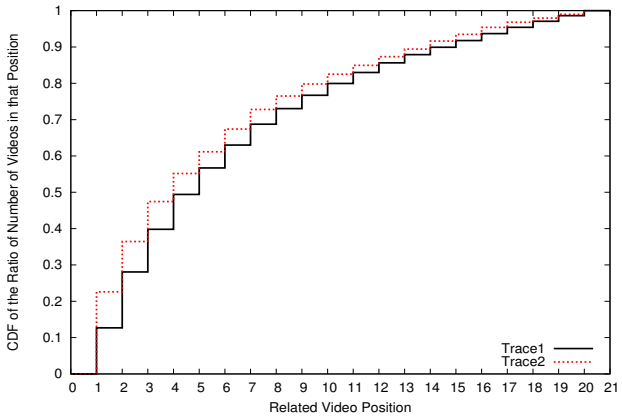


Figure 3: Position of video selected by users from related video list for traces T1 and T2.

2.2 Chain and Loop Count

To better understand YouTube's recommendation system, we first investigated how repeated selections of videos from the related lists behave. This has let us to define the concept of *chains* and *loops*.

Our definition of the *Chain Count* and *Loop Count* is as follows; *Chain Count* is the number of consecutive videos requested by the user from the recommended list until she requests a new video by other means (e.g., a search). *Loop Count* is defined as the number of consecutive videos a viewer requests from the recommended list before she requests the same video as the initial one and thereby forms a loop. Figure 4 provides an example of the definition of both Chain Count and Loop Count. In the example shown in Figure 4, both Chain Count and Loop Count is two.

The goal of this section is to gain better insight into the characteristics of related lists and how they influence the performance of caching.

2.2.1 Observed Chain Counts

In this section, we provide the results of the chain count analysis performed for the campus network traces described in Table 1.

For the chain count analysis, we go through the requests from each user in our trace to determine the number of consecutive requests made by the user from the related list of the previous video requested. The results from our chain count analysis on the two traces are summarized in Table 2. For trace T1, the viewers choose videos from the related list at least once in 84.76% of the cases, and in 15.24% of the time, the chain count is larger than one, leading to an average chain count of 1.195. The maximum chain count seen

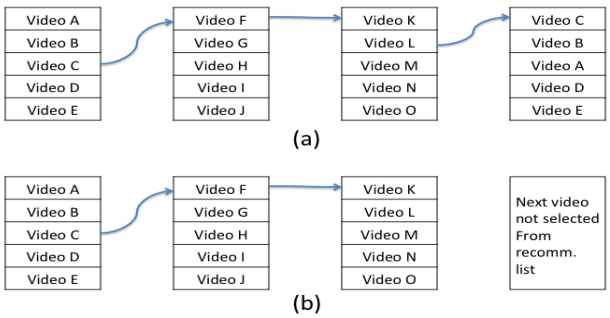


Figure 4: Chain and Loop count example. (a) Example for a loop of length 2, (b) Example of a chain with length 2.

Approach	Trace T1	Trace T2	Global
Avg. chain count	1.195	2.304	-
Max. chain count	8	21	-
Avg. loop count	-	-	1.243
Max. loop count	-	-	19

Table 2: Chain and Loop count analysis results

in trace T1 is 8, i.e., a user selects videos consecutively 8 times from the related list before selecting a video by other means. For trace T2, the values are a bit higher. In 48.20% of the cases, the chain count is at least one, and in 51.80% of the cases, it is more than one, leading to an average chain count of 2.304. The maximum chain count seen in trace T2 is 21.

The results in this section strengthens our assumption that the user selects video from the related list. In order to understand if this assumption stands good on a global scale, we also perform the loop count analysis on PlanetLab which is presented in the following section.

2.2.2 *Loop Length at Fixed Positions*

To investigate the loop count on a global scale we performed a PlanetLab-based measurement. In a first experiment, 100 videos were accessed from each PlanetLab node. Then, we followed the chain of related movies by repeatedly selecting the same position. We did that for each of the positions 1 through 20. Making these deterministic selections, we continued to follow the chain until the loop closed and we arrived at the original video again. For example, if always the top video is selected from the related list and the initial request was for video A and this video is selected again after four rounds, then this case results in a loop count of three. The results of the experiment are shown in Figure 5.

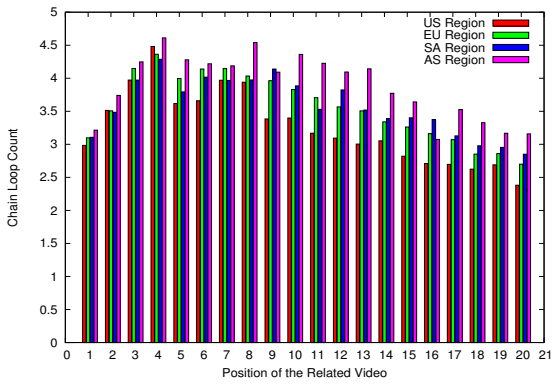


Figure 5: PlanetLab-based loop count measurement for fixed position selection from related list

The experiment was performed with possible regional deviations in mind, and the results shown in Figure 5 are subdivided by region. As can be seen from the figure, the general trend persists throughout the regions, and yields some unexpected rather surprising results.

It is understood that the related list is constructed from a search for related videos, and the result is then sorted by popularity before it is presented to the user. This understanding implies that videos that appear on lower positions in the related list are less closely related to the current video than earlier ones. Considering the large library of videos held by YouTube, we would expect that the set of less closely related videos is growing with distance; consequently, following the chain in the less closely (lower) related position should find a larger number of videos, leading to a longer loop. According to Figure 5, that is not the case.

Since there could be many reasons for this behavior, we are trying to get a better understanding why the chain does not increase for lower related list positions. For that reason, we investigate the loop length for the case in which related list positions are randomly chosen.

2.2.3 *Loop Length at Random Positions*

Section 2.2.2 indicates that the pool of videos from which related lists are selected is rather small for each position. To better understand the impact of always choosing from a fixed position, we perform another experiment which is based on random positions. We vary the number of maximum positions that can be selected.

For this measurement, each PlanetLab node initially requests the same video and obtains the recommended list associated with this video. From this list, a new video is randomly selected and this procedure is repeated 50 times. This experiment is repeated four times where the selection process for videos from the recommended list changes for each experiment. In the first experiment, only the top 5 videos are randomly selected, in the second experiment the top 10, and in the third and fourth the top 15 and top 20 are selected, respectively. This procedure results in 50 video requests per node per experiment, and we use this result to determine the loop count.

The results from this measurement are shown in Figure 6. We decided to repeat the experiment with Top 5, 10, 15, and 20 video selection to investigate if different user behavior has an impact on the chain and loop length. The X-axis in Figure 6 shows the loop count and Y-axis shows the percentage of videos with the corresponding loop count in x-axis for Top 5 to Top 20 video selection. As can be seen in Figure 6, the trend for the loop lengths is relatively similar, independent of the range of top videos from which the selection is made.

It is noteworthy that the average loop length in case of random selection from a set of positions is similar to that of a fixed position, with minor changes for an increased num-

ber of maximum positions to choose from. The likely explanation for this effect is that the related list is mostly constructed from a limited and closed pool of videos. The search features that keep videos semantically close to each other dominate over features that would promote drift towards other videos. This is probably for the benefit of the user: users selecting from the related list may not be interested in drift; the small pool prevent a "lost-in-hyperspace" effect. However, the limited pool size is also advantageous for the effectiveness of proxy caches. A request entering a pool through any video is apparently going to promote staying in the pool, increasing the chance that a chosen related video is in the cache.

Figures 5 and 6 indicate that YouTube does not assign the video to positions for some internal purpose. In particular, it does not seem to be built to make their caching more efficient. If that was the case, then loop lengths shown in Figure 5 should be different from loop lengths in Figure 6, which is not the case. Therefore, we can conclude that we can reorder the related list without interfering with the original goal of the related list.

2.3 Video Origin

Since we are aware of the fact that YouTube is already employing caches in its video distribution infrastructure (see, e.g., [9] or [12] for more details), we executed the following experiment to investigate if a video is served from a close by cache or not.

Our intention behind this idea is the conjecture that a larger average RTT for the TCP streaming session means a larger distance (in hops) between source and client. To support this conjecture, we performed the following measurement from our campus network. First, we randomly selected 100 videos from trace T1 and retrieved each of the 100 videos while capturing a packet trace by using tcpdump [6]. We then analyzed each of the 100 resulting packet traces and determined the average RTT of the TCP session that transmits the actual video data. We then repeated this experiment for the top 20 videos from the related list of each of the initial 100 videos, resulting in 2000 data points.

Figure 7(a) shows the CDF of the average RTTs for each of the YouTube video requests as described above. As it can be seen from the figure, there are three different ranges of average RTTs (<3ms, between 3ms and 12ms and >12 ms). Mapping these RTT ranges to IP addresses gives us three different subnet ranges. Thus, we can safely say that, we see three different cache levels in our experiment and the CDF plot for the video requests from each of the cache levels (Cache Level1 is <3ms, Cache Level2 is between 3ms and 12ms and Cache Level3 is >12ms) is shown in Figure 7(b).

As shown in Figure 7(b), the majority for both groups of videos (initially selected and selected from the related list) are served from caches that are on level 2 (55% and 52%). Only $\sim$ 30% of the video from both groups are served from a level 1 cache. Further analysis revealed that all sessions with an average RTT of 3ms or less have a sender IP address from an address range that is administered by the University of Massachusetts, while address ranges for the level 2 and 3 caches are administered by Google (the owner of YouTube).

Based on the results presented above, we are now able to identify which of the videos selected from the related list are served from a close by cache (level 1) and which are served from a caches further away (levels 2 and 3). With that

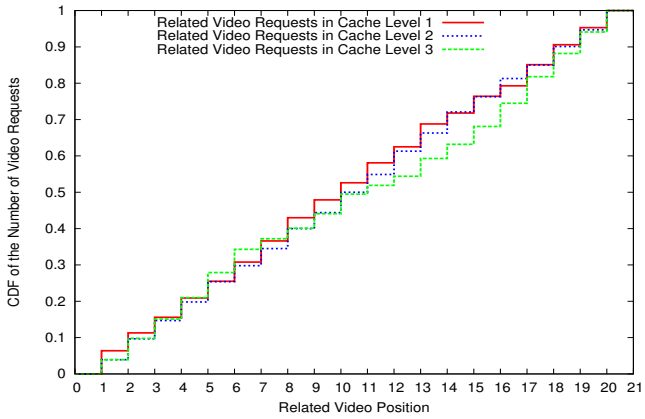


Figure 8: Distribution of position of video in related list compared to all requests originating from the related list.

information we determined the distribution of the position of the requested video in the related list for videos served from a close by cache and for those served from more distant sources. This distribution (in form of a CDF) is shown in Figure 8. The nature of the CDFs shown for both cases is almost uniform. This is a strong indicator that YouTube is not giving any preferences, in terms of caching based on the position of the video in the related list. It rather confirms the results presented in [12] and [11] that claim YouTube mainly focuses on load balancing between different cache levels and not on the goal of moving content as close to the client as possible.

Knowing that viewers are more likely to select top ranked videos on the related list, we are interested on researching if a caching approach that is tailored more towards this viewer behavior can improve caching efficiency. We present and investigate such an approach in the following section.

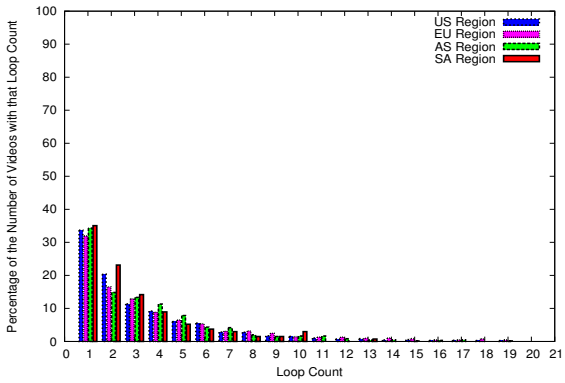
3. LOCAL RECOMMENDATION TO IMPROVE CACHING

In this section, we present and analyze our related video list reordering approach to investigate if it can improve the performance of caching. After introducing the general approach, we discuss two different approaches of video selection from reordered related list and present the results for both approaches obtained through trace-based simulation.

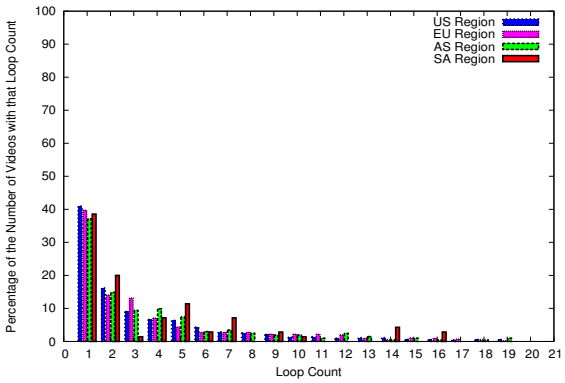
From the results obtained in Section 2.1, we conjecture that the distribution of user requests on related video position is innate in the way in which users interact with the GUI. It should therefore be possible to use the order of the related video list for our purpose, which is to increase the cache hit rate.

The basic idea is to perform cache-based reordering, where the order of videos in the related video list is modified based on the cache's content. Figure 9 shows an example of the cache reordering approach. In this case, the related video list is sent from the server through the cache to the client. Upon receipt, the cache modifies the related video list according to the videos it has currently cached. After modifying the list, the cache forwards the reordered list to the client.

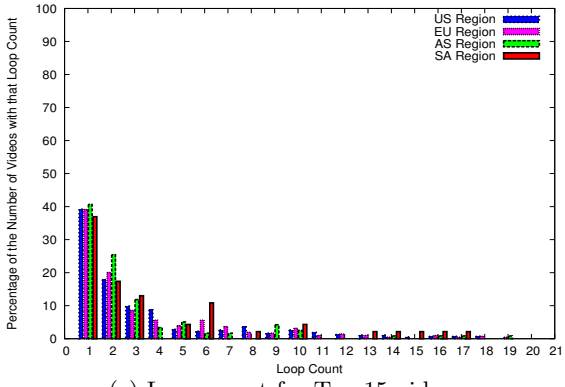
The reordering maintains the order that the cached videos have on the original list as well as the order among the un-



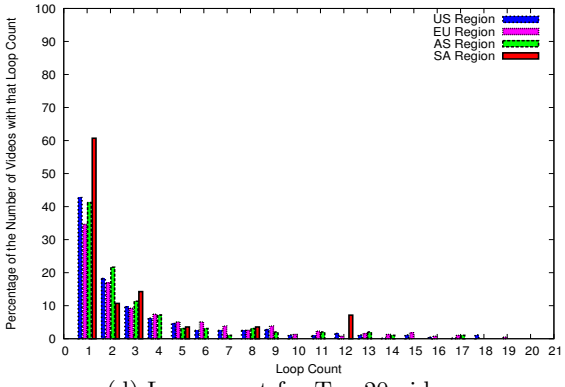
(a) Loop count for Top 5 videos



(b) Loop count for Top 10 videos

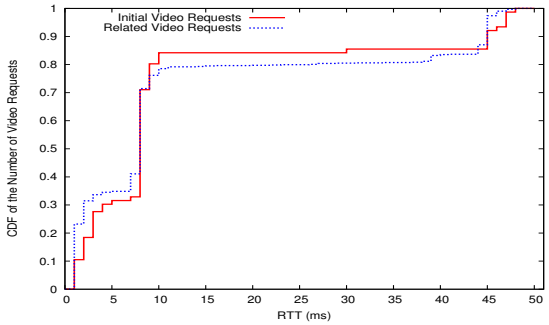


(c) Loop count for Top 15 videos

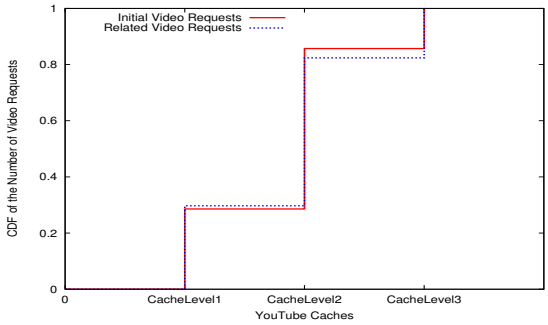


(d) Loop count for Top 20 videos

Figure 6: PlanetLab-based loop count measurement for random selection from related list



(a) CDF of average RTTs for TCP streaming session



(b) CDF of video requests based on cache levels

Figure 7: Distribution of video delivery based on response times and cache levels.

cached videos. Considering the example shown in Figure 9, this means, while videos B, C, and D are moved to the top of the related video list that is transmitted to the client, they (amongst themselves) keep the order they had in the related video list the cache originally received from the server.

Implementing the proposed cache-based reordering of the related video list is straight-forward and will add only a small amount of overhead on the cache. As shown in Figure 9, the reordering approach requires the cache to provide the following two functionalities: *a)* maintain a list of the videos that are stored in the cache, and *b)* a process that is able to modify the related list. Functionality *a)* will be provided by any ordinary cache since it needs to determine if a client request can be served locally or has to be forwarded to a server. The additional functionality *b)* that is required for cache-based reordering is a simple manipulation of an HTML page that represents the related video list as, e.g., shown in Figure 1. Such simple modification of HTML code should increase the computational load on the cache insignificantly.

In the following we first analyze how well the videos that are placed at the top of the related list overlap with the movies that are present in the cache. To do this, we replay the trace that we described in Section 2.1, but with reordered related lists and reordered video selections. We

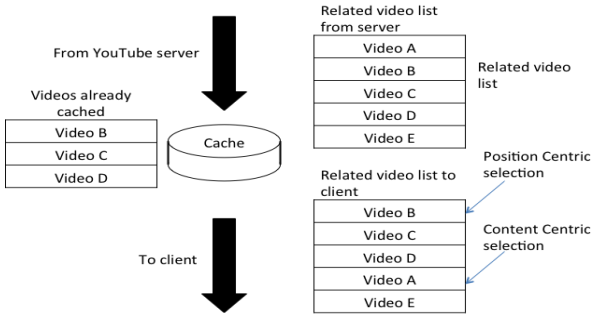


Figure 9: Reordering of the related video list by the cache

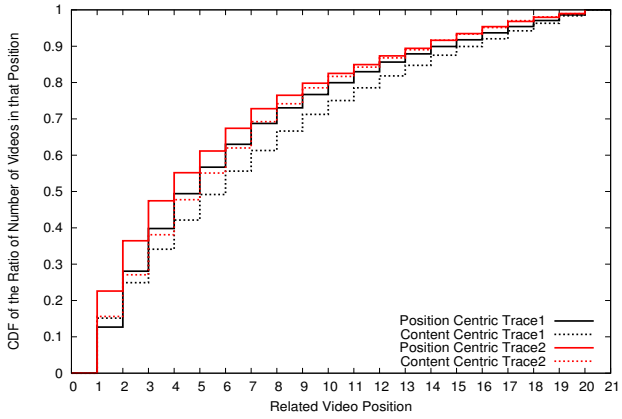


Figure 10: Position of video selected from related video list after reordering.

call this the content-centric reordering, and describe it in the following subsection.

3.1 Content Centric

In content centric approach, the video the user requested in the original trace before reordering the related list is fixed, i.e., the sequence of videos that are requested is identical to the original trace. The position of the video that is requested from the modified related list, however, is different after reordering.

For the example shown in Figure 9, the viewer would select the video at position four of the related video list received from the cache, if the interest is absolutely in video A's content and not simply in one of the top listed videos. (In Section 3.2, we will present a position centric approach where the selected position stays fixed and, consequently, the content changes.) The content centric approach leads to a different probability distribution of the selected related video list position as shown in Figure 10. The dashed line in this figure shows the distribution of the positions that were chosen by a viewer in the case of content centric caching and a comparison with an unmodified related video list (see Section 2.1) as indicated by the solid line shows the difference in the two probability distributions. The comparison shows that the content centric approach leads to the selection of lower ranked videos. This is caused by the fact that non-cached videos will be pushed down by the cache reordering scheme in the related video list forwarded to the client. The probability distribution reflects a content-driven behavior, where the viewer makes the video selection based on the content and not the position of the video.

Figure 10 serves as an indication that one of our basic assumptions is correct: users are more likely to select from the top of the related video list than from an arbitrary position. It is only an indication, though, verification would require us to construct related lists ourselves. Theoretically, it remains possible that videos included in the related lists have very different relevance and that real viewers would select by content rather than order. We can only disprove this by anecdotal evidence.

This observation confirms our assumption that the order of videos in the related list does actually matter, and that an increase in cache hits can actually be achieved. To find out just how much improvement we can achieve, we investigate an approach that we call the position-centric reordering. It is investigated in the following subsection.

3.2 Position Centric

With the position centric approach, the related video list **position** selected by the viewer stays fixed. This clearly might result in a different content to what the viewer would have watched would the cache not have modified the related video list (see Figure 9). In Figure 10, the solid line shows the distribution of position centric video selection from the reordered list for both the traces used in our analysis. The distribution is similar to the one in Figure 3, as we keep the position of the related video selected by the user same as in the original request.

Compared to the content centric approach, the position centric approach introduces two inaccuracies that we cannot account for without real-world testing. First, with more hits, the set of cached videos will be more stable, and more videos will be cached in competition with other content. This leads to an underestimation of reordering on cache hits in our study. Second, reordering keeps popular videos at the top of the related video lists. The high probability of choosing videos from the top of the list leads the emulated user into cycles, which most real users would avoid. This leads to an overestimation of reordering on cache hits in our study.

We used the data from the campus network traces and information retrieved through the YouTube API [8] to perform a simulation of the cache-based reordering. We simulate the caching of videos which were requested in the traces. E.g., if we detect a request for Video A in the trace, we log this video as stored in the cache. Since each YouTube video is assigned a unique identifier which is included in the video requests in the network trace, we can accurately simulate the behavior of the cache. Via the YouTube API, we retrieve the related video list for each video that is requested in the network traces. Each time we simulate a video request we simulate the modification of the related list (retrieved via the API) based on what is stored on the cache. E.g., if a video in the related list obtained from the YouTube API is present in the cache, the video is placed at the beginning of the re-ordered related list and the videos which are not presented in the cache are pushed downwards. It is important to note that we do not change the content of the related list

Trace	No Reordering	Cont. Centric	Pos. Centric
T1	6.71%	6.71%	11.83%
T2	4.71%	4.71%	22.90%

Table 3: Comparison of hit ratio

obtained from the YouTube API at all. Only the order of the videos in the related list is manipulated. We simulate user behavior where the viewer selects the video from the same position in the related list as presented to her in the original trace. In this scenario, the content the user selects at that position in the re-ordered related list might be different from the original content that was requested in the trace. For this investigation we use a custom-built simulator implemented in perl. The results from our analysis are summarized in Table 3.

The simulation of a simple cache that does not perform reordering of the related video list results in 7055 hits or a 6.71% hit ratio for trace T1 and a 4.71% hit ratio for trace T2. Simulating a cache that employs the reordering of the related video list with the position centric scheme results in 12431 hits or a 11.83% hit ratio for trace T1 and 1735 hits or a 22.90% hit ratio for T2. Although the hit ratio is not very high, these results shows the potential of this approach since it significantly increases the hit ratio compared to the simple caching case that does not modify the related video list.

Finally, simulating a cache that employs the modification of the related video list with the content centric scheme results in 7055 hits or a 6.71% hit ratio for trace T1 and 397 hits or a 4.71% hit ratio for trace T1, which is identical to the results for the simple caching scheme. This is caused by the fact that the video is selected that is not at the very top of the related video list and, thus, not stored on the cache.

4. DISCUSSION

In this section, we discuss the advantages and disadvantages of cache centric video recommendation. We address the issues of increased cache complexity, server load reduction, alternative sorting of the related list, and adaptive video streaming.

4.1 Cost of Recommendation List Reordering

We have shown in Section 3.2 that the cache hit rate is significantly increased when the related video list is reordered according to the cache content (see Table 3 for detailed data on hit rates), using the access patterns from our campus network traces. The cost of this process is a more complex cache server. For each request, the server must identify URLs from YouTube and upon success, the cache must be inspected to find whether the requested video is cached or not. The cost of an additional cache lookup is dependent of the cache structure and the size, but assuming a lookup structure based on a plain hash table, the average and worst case lookup times are O(1+n/k) and O(n) respectively, where n is the number of elements in the cache and k is the number of buckets in the hash-table. Adding for example a second data structure for the bucket list, like a self balanced tree, the theoretical worst-case time of common hash table operations can be brought down to O(log m) rather than O(m) where m is the number of elements in the hash-bucket.

Thus, the reordering comes at the described extra cache server cost. However, taking into account the very long tail distribution of videos, the gain in cache hit rate is substantial compared to the added processing. Additionally, there are several systems today that already rewrite webpage content. For example, Opera is currently rewriting and rearranging the content in web-pages using proxies for their opera mini browser [5]. Similarly, proxies like FilterProxy [2], Muffin [4] and WebCleaner [7] have the capability to modify content on the fly. We therefore claim that the proposed reordering of items in the related list is worth the additional complexity at the cache since this drawback is outweighed by the benefit of a significantly increased hit rate.

4.2 Reduction in Server Load

Another advantage of our related list reordering approach based on the content in the cache is the reduction in the server load, and hence, the reduction in back-end bandwidth consumption. With the reordering of the related list and pushing the contents from the bottom of the list to the top, we take advantage of the user behavior in selecting the videos from the top of the related video list. With reordering the list to contain the videos from the cache in the top of the related list, we reduce the load on the YouTube server (or higher level caches) proportional to the number of cache hit gains. From the analysis in Section 3, we show that with our reordering approach, the cache hits increase using the T1 trace is from 6.71% to 11.83% which turns into an approximately doubling of the hit rate. Similarly, the bandwidth consumption reduces from 93.29% to 88.17%, which provide a 5.12% reduction in server load and back-end bandwidth reduction from our related list reordering approach. Similarily, from the analysis of trace T2, we see a cache hit increase from 4.71% to 22.9% which is almost a five times increase on the hit rate. From the YouTube server point of view, this is a server load reduction of 18.19%.

4.3 Local Popularity Based Sorting of Related List Items

So far, the reordering of the related list does not take differences of local popularity of cached videos into account. As mentioned in Section **??**, the related list reordering is based on a stable sort. This reordering could be further evolved by slightly modifying the sorting algorithm such that it also sorts the locally cached videos by their local popularity. Similar to the original reordering approach, the locally cached video would still be pushed to the top of the related list. But the order in this group of videos would now be ordered by local popularity. For example, if video C is more popular than video B, then video C would be ranked first in the related video list that is sent to the client. This approach, however, needs further investigation to study its feasibility. First of all, the actual differences in popularity for videos that are stored on the local cache and belong to the related list of a video have to be determined. We believe that only significant differences in the popularity of the respective videos would render this approach feasible.

4.4 Adaptive Video Streaming

Today, many commercial service providers use some kind of adaptive streaming solution where the bandwidth consumption (and thus video quality) is adapted to the amount

of available resources. At first, this will obviously affect the caching. However, according to YouTube [9], this feature is currently only supported for live streaming, which is not the focus of our work. While YouTube does not support adaptive streaming for user generated content, it offers some of this content in alternative formats and resolutions, and gives the viewer the option in which quality (format) the video should be streamed. From our point of view, the same video in a completely different format represents a new object, which is treated in the same way as a object that represents different content. Furthermore, we also expect a convergence towards higher quality data in the cache if the cache hit ratio is increased as more users then experience a higher resource availability as it is unnecessary to go all the way to the YouTube servers or higher level caches. Consequently, in the work presented in the paper, we do not take into account adaptive video streaming.

Nevertheless, segmented adaptive streaming approaches will influence the caching results and how to design a caching policy. Each segment is available in multiple qualities and the system automatically adapts between qualities. However, this is out of scope for our current YouTube investigation, but such a topic is subject for future work.

5. RELATED WORK

The use of proxies and caches has been intensively studied in related work. In the following, we mention the ones closest to our work. In [13] and [21], trace-driven simulations were performed to investigate the effectiveness of caching for YouTube videos. Although the traces for both studies were different, the results showed that caching can reduce server and network load significantly. Both studies did not consider reordering the related video list to increase the efficiency of the cache.

Besides caching, YouTube's recommendation system (the related video list) has also been studied in related work. In [19], Zhou et al. analyzed two data sets (one directly crawled from YouTube and the other one a trace from a campus network) to investigate if the position of a video in the related list has significant impact on it being selected by the viewers. The results of this analysis show that a large percentage of viewers select videos they watch from the related list (see Figure 2). In follow on work [20], Zhou et al. perform further analysis of YouTube's recommendation system based on global statistics for YouTube videos. The authors show that the click through rate is proportional to the position of the video on the related list (the higher the position of the video on the list, the higher the chance that it will be selected by the viewer). As in [19], the results confirm our assumption on related list usage and support the position centric approach related list reordering approach. While the goal of the work presented in [17] was to show how the prefetching of prefixes from videos on YouTube's related list can improve caching, it also shows that the related list is often used by viewers to select a video (see Figure 2). In contrast to the work we present in this paper, no modification of the related list at the cache is proposed.

Cheng et al. [16] measured the YouTube video graph created by related video links and found that the graph has a large clustering co-efficient and exhibits the small world property. A simulation-based evaluation of a P2P video sharing systems showed that if users use the related video list to browse videos, the percentage of source peers that have the requested video in their cache is high. Cheng and Liu [15] also proposed a P2P video sharing system to reduce YouTube server load and suggested using prefetching based on YouTube's related video list at the clients of a P2P system to provide smooth transition between videos. Their evaluation was based on emulated user browsing pattern. The evaluation of their approach showed that it performs significantly better (55% hit ratio) comparing with a random prefetching approach (nearly 0% hitrate).

In [12] and [11], Adhikari et al. uncover the overall architecture of the YouTube video distribution system. While our results presented in Section 2.3 agree with their findings, their work does not investigate which of the videos of the related list are transmitted from which cache level. In addition, their work is not concerned with improving the efficiency of YouTube caches.

Chakareski [14] takes a more general view on recommender systems (called catalogues in his case) to develop an optimization framework that has the goal to reward providers if an item selected by a customer can be provided immediately and penalize them if the provision is delayed. Similar to our case, the author shows that optimizing for immediate access to items at the beginning (or top) of a catalogue is beneficial and optimizes the reward for the provider. The paper does not reveal how this immediate access can be provided, and we see our work as complementary part since it addresses content provision. In addition, our study is based on actual user behavior obtained from trace data.

To the best of our knowledge the work we present in this paper is the first that investigates how reordering or information provided by a recommendation system based on a cache's content can improve its performance.

6. CONCLUSION

Recent works have shown that, YouTube viewers select their next video request from the related list provided by YouTube on the watch page of the current video with high probability. In this paper, we take advantage of this user behavior to modify the related list provided by YouTube based on the content already requested by the users in a gateway network. In our approach, the related list is modified only in the order of appearance on the user's webpage, not the content itself. During the modification, the related videos present in our cache, which are the videos already requested by the users, are moved to the top of the list and subsequently the related videos not cached are moved down the order. By analyzing a campus network trace filtered for YouTube traffic, we find that, users generally request videos from the top half of the related list and hence moving the related videos already present in our cache makes the approach more advantageous.

We analyze our approach in a simulation based study on the network trace captured on a campus gateway. We define two different approaches of video selection from the reordered related list. Content Centric selection, where the user selects the same video as he would have originally selected and Position Centric selection, where the user selects a video from the same position on the related list before reordering, which might change the original content. From our simulation study, we find that Position Centric selection of the reordered related list yields a better hit rate than Content Centric selection, which does not take advantage of the content in the cache.

We also analyze the YouTube caching strategy for the related videos of the requested videos. We find that YouTube does a three-tier caching for all its requests and uses load balancing techniques to determine the cache from which the video is served. This shows that YouTube does not give special preferences to the top order of related videos requested by the users. We recommend that, serving the top order (presumably Top 5) of the related videos of each requested video from the nearest cache reduces the latency of the YouTube videos requested by the users.

From the analysis of the caching techniques used by YouTube and our related list reordering approach, we conclude that reordering the related list presented to the user based on the content already requested by the users, yield a better hit rate of the caches, thereby reducing the bandwidth consumption by multimedia requests and hence the latency in content delivery. Also, serving the top order of the related list videos from the nearest cache instead of a load balancing approach between three-tiers of caching will reduce the latency in providing the content requested by the user.

7. REFERENCES

[1] Endace DAG Network Monitoring Interface. http://www.endace.com/.

[2] Filter Proxy. http://filterproxy.sourceforge.net/.

[3] Google Transparency Report. http://www.google.com/transparencyreport/traffic/?r=US&l=YOUTUBE&csd=1345235172273&ced=1346728973645/.

[4] Muffin. http://muffin.doit.org/.

[5] Opera Mobile. http://www.opera.com/mobile/.

[6] tcpdump. http://www.tcpdump.org/.

[7] Web Cleaner. http://webcleaner.sourceforge.net/).

[8] YouTube API. https://developers.google.com/youtube/.

[9] YouTube Keynote at MMSys 2012. https://docs.google.com/presentation/pub?id=1bMLitOe_fxARBbgcu1v1xaJj89hb_JGXYse17Xvgwro&start=false&loop=false&delayms=3000#slide=id.g47538e9_2_210.

[10] YouTube Press Statistics. http://www.youtube.com/t/press_statistics/.

[11] V. Adhikari, S. Jain, Y. Chen, and Z.-L. Zhang. Vivisecting YouTube: An Active Measurement Study. In *INFOCOM*, March 2012.

[12] V. Adhikari, S. Jain, and Z.-L. Zhang. Where Do You "Tube"? Uncovering YouTube Server Selection Strategy. In *ICCCN*, August 2011.

[13] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *IMC*, October 2007.

[14] J. Chakareski. Browsing Catalogue Graphs: Content Caching Supercharged!! In *ICIP*, September 2011.

[15] X. Cheng and J. Liu. NetTube: Exploring Social Networks for Peer-to-Peer Short Video Sharing. In *INFOCOM*, April 2009.

[16] X. Cheng, J. Liu, and H. Wang. Accelerating YouTube with Video Correlation. In *WSM*, November 2009.

[17] S. Khemmarat, R. Zhou, L. Gao, and M. Zink. Watching User Generated Videos with Prefetching. In *MMSys*, February 2011.

[18] D. E. Knuth. *The Art of Computer Programming, volume 3: (2nd ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[19] R. Zhou, S. Khemmarat, and L. Gao. The Impact of YouTube Recommendation System on Video Views. In *IMC*, November 2010.

[20] R. Zhou, S. Khemmarat, L. Gao, and H. Wang. Boosting Video Popularity through Recommendation Systems. In *DBSocial*, June 2011.

[21] M. Zink, K. Suh, Yu, and J. Kurose. Characteristics of YouTube Network Traffic at a Campus Network - Measurements, Models, and Implications. *Elsevier Computer Networks*, 2009.