Impact of Frequency-Based Cache Management Policies on the Performance of Segment Based Video Caching Proxies

Anna Satsiou, Michael Paterakis

Laboratory of Information and Computer Networks
Department of Electronic and Computer Engineering
Technical University of Crete,
73100 Chania, Crete, Greece.
{anna, pateraki}@telecom.tuc.gr
Tel: +30-28210-37343, 37225, Fax: +30-28210-37542

Abstract. This paper examines a novel cache management policy, which we call LRLFU and is applied to groups of blocks of the media files. This cache management policy is capable of capturing the changing popularities of the various videos by attaching a caching value to every video according to how recently and how frequently the video was requested, and decides to cache the most 'valuable' videos. Our event-driven simulations have shown that the frequency considerations we have introduced in the caching values attached to the videos (i) improve the byte-hit ratio and (ii) significantly reduce the fraction of user requests with delayed starts.

Keywords: Multimedia Streaming, Video Proxy Caching, Segment-Based Video Caching, Frequency Based Cache Management Policies.

1 Introduction

The widespread use of the Internet, and a mature digital video technology have led to an increase in various streaming media applications such as corporate communications, distance education, Internet TV broadcasting and entertainment video distribution. However, the high bandwidth requirements and the long duration of digital video are two major limitations in supporting high quality streaming applications at a large scale over the Internet. Particularly, during periods of network congestion and media server overload, start-up latency and interrupts of video streams are very common. In order to alleviate the above situation, research efforts have been undertaken in two directions: a) use of multicast or broadcast connections to stream a popular video to groups of clients, and b) video caching at proxies located closer to the end-user.

Recently, there has been research in a combination of the above areas (see [1]-[4]), where partial caching and i) periodic broadcast [1],[2], or ii) techniques like batching, patching and stream merging [3],[4], are studied together. However multicast is considered impractical as its deployment in the Internet is still very limited. As far as sole proxy caching is concerned, research has been focusing in two directions: i) the effective maximization of delivered quality to heterogeneous clients (see [5],[6]) and ii) the reduction of server loads, network traffic and user access latencies (see [7]-[11]).

Most of the above video caching schemes do not dynamically cache the files in response to individual client requests, rather they employ replication of parts of the videos according to a pre-estimated access pattern of each video. In reality, the request rate for a particular video may vary with time, and the relative popularities of the videos may vary from proxy to proxy. Proxies must be flexible to the particular preferences of different client populations and should be able to dynamically change their content accordingly. In [9] dynamic caching mechanisms, which are robust for evolving client workloads, are examined. Partial caching is used by segmenting video files in segments of variable size forming a pyramid and a portion of the cache capacity is dedicated to cache only the prefixes of each object, while the remaining part of the cache is used for the later segments. Caching a prefix of the video relieves clients from delays and jitter on the server-proxy path, while a proper cache admission and replacement policy is used to achieve a high byte-hit-ratio (BHR) even when popularity changes occur frequently. Video segments are ejected from the cache according to a caching value, which is based on how recently they have been requested, and their distance from the beginning of the video file.

Our work is based on some ideas from [9], it adopts the idea of using a portion of the cache referred to as part A to cache the prefixes of the videos and the rest referred to as part B for the latter segments, while using a different than in [9] cache management policy and examining a new segmentation scheme. Our work aims at i) improving the byte-hit ratio (BHR) at the proxy, thus reducing the average backbone load, irrespectively of the transmission scheme that is used and ii) reducing the fraction of user requests with delayed starts.

The remainder of the paper is organized as follows. Section 2 presents the system model and describes the segmentation schemes and cache management policies that we use. Section 3 presents our event-driven simulation model, parameters and results. The same section contains the discussion of the results. Finally, section 4 concludes the paper and describes ideas of ongoing work.

2 System Description

2.1 Network Topology

Figure 1 illustrates the system architecture we are considering. Our system consists of a far-distant origin server which stores the video files, proxy servers located close to the client populations which cache the files or parts of the files, and client devices. User requests for videos are directed to the nearest proxy. If the proxy has the whole or part of the media file, it transmits it to the client, while it contacts the content server for the rest of the file. If the video is popular enough, the proxy server decides to cache a part of it so that it can satisfy subsequent requests for the same video. In order for the traffic volume in the server-proxy path to be reduced, the most popular videos must be cached in the proxy. It is assumed that the bandwidth between the proxy and the clients is sufficient to support video streaming with negligible latency, while the latency between the content server and the proxy is quite significant. Therefore, the idea is to cache enough initial segments of each video to mask the latency between the content and the proxy server.

2.2 Segmentation of video files

Here we introduce and briefly discuss two different segmentation schemes. In the first scheme, a media object consists of multiple equal-sized blocks, which is the smallest unit of transfer. Blocks of a media file received by the proxy server are then grouped into variable-sized, distance-sensitive segments. The segment size increases exponentially from the beginning segment, forming a pyramid. Assume that b is a constant and $b > 1$, S_i^p denotes segment i of a media stream under pyramid segmentation and N_i^p denotes the number of media blocks contained in S_i^p then

$$N_i^p = 1 \qquad \text{if } i = 0$$
$$N_i^p = (b - 1)b^{i-1} \qquad \text{if } i > 0$$

Segment S_i^p contains media blocks b^{i-1}, $b^{i-1} + 1$, $\cdot\cdot\cdot$ $b^i - 1$, if $i > 0$. Segment S_0^p contains block 0.

In the second segmentation scheme, we consider that the first segment consists of a small number of blocks Bp, forming the prefix of the video and later segments consist of a larger fixed number of blocks. Bp is chosen in a way to guarantee that enough blocks will be in the cache to mask the latency in the server-proxy path in order to guarantee continuous streaming of the video once it is started. We refer to this segmentation scheme as the ‘fixed segmentation with prefix’.

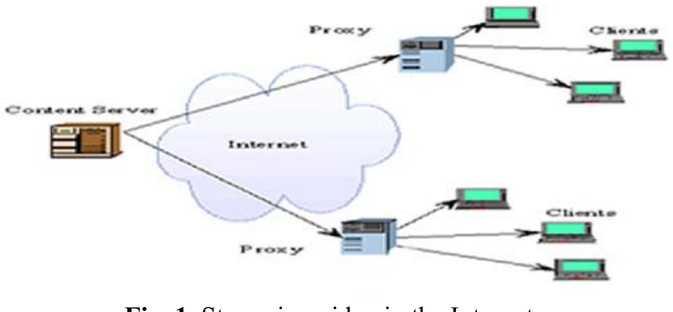

Fig. 1. Streaming video in the Internet.

2.3 Cache management policies

We have adopted from [9] the idea of dividing the cache in two parts. The size of the first part is a small portion of the total cache capacity and is dedicated to store only the initial blocks Bp (prefix) of each video while the rest is used to cache later segments. We refer to the first part as Cache A and to the second as Cache B. Different replacement policies are used in Cache A and Cache B, that is, different replacement policies are applied to prefixes than to later segments. When a media object that is not in the cache is requested, its Bp blocks are always eligible for caching in Cache A and form the prefix of the video. The later segments of the video will be considered for caching in Cache B in subsequent requests for the same video. Once an objects' prefix is cached in Cache A, information is maintained for this object, such as the object ID, the timestamp of the last request to this object T′ (timestamp records the last time the object has been requested), and the number i of the last segment of the object inside the cache. We further keep the number of the requests for this object, RF. This information is updated at every request of the particular video.

In summary, the replacement policy in Cache A decides which videos should be cached in the proxy and the replacement and admission policies in Cache B determine the portion of these videos that should be cached.

2.3.1 Replacement policy in Cache A

Bp blocks are cached as an entity in Cache A and can only be replaced by other initial Bp blocks of another video. In [9], in order to find space to cache the Bp blocks of a video, a simple LRU policy is applied to all the videos in Cache A that are not currently played. We however use here another replacement policy. We consider that every prefix in Cache A has a caching value given by $\frac{RF}{T - T'}$, where T is the current time. In order to cache the Bp blocks (the prefix) of a requested video, the caching values of the prefixes of all the videos in Cache A that are not currently played are examined and the one with the smallest caching value is removed from the cache. Every time a prefix of a video is moved out from Cache A the remaining segments of the video (the suffix), if any, that are cached in Cache B are also moved out from Cache B and all the information maintained for this video is deleted. The prefix of the requested video is cached, and as previously mentioned, its later segments will be considered for caching in subsequent requests for the particular video.

Notice that by applying this caching value to each prefix, the video that is ejected from the cache is not the least recently used but the least valued according to how frequently and how recently it was requested.

2.3.2 Replacement and admission policies in Cache B

Once a video is requested and its Bp blocks are already cached in cache A, the proxy checks to see if it must cache another segment X of that video in Cache B. At every subsequent request, only one segment is considered for caching. The authors in [9] assigned a caching value to every segment of a video, given by $\frac{1}{(T - T')i}$, where i is the number of the segment. According to this scheme, initial segments have larger caching values. Furthermore in [9], a **small number** of videos in Cache B that are not currently played are examined from the bottom of an LRU stack and the caching values of their last cached segments are compared. Then, the caching value of the least valued segment Y is compared with the caching value of the segment X that is considered for caching. If it is smaller than the caching value of X, the proxy removes segment Y. The same procedure is repeated until there is enough free space in Cache B to cache segment X. If not enough segments with smaller caching values than the caching value of X can be found in order to free space in Cache B, segment X is not cached in the proxy. Segments are always removed from the end of the cached video files.

In contrast to the above, we use a caching value for every video and not for individual video segments as this would simplify the cache management operation. The caching value of every video participating in the cache admission and replacement policy in Cache B is the same with the one used in Cache A, given by $\frac{RF}{T - T'}$. We would like to add here that we also tried to use the number of segment i in the denominator, as in [9], and assign a different caching value to each segment of a video file given by $\frac{RF}{(T - T')i}$. However, our simulation results were no sensitive at all to the presence of the segment number i in the denominator.

The procedure followed by our scheme in order to cache segment X of the requested video is the same with the one described above with the only differences that we examine **all** the videos in Cache B that are not currently played in order to select our victim segments and that we use a different caching value, which is the same for all the segments of the same video. Similarly, at every subsequent request only one segment of the video is considered for caching and video segments are always removed from the end of a cached video file.

In order to compare our scheme with the one in [9], we simulated their scenario but in order to find segment victims from Cache B, we examined the last segments of **all** the videos in Cache B that are not currently played and this explains the observation that their scenario appears to work better in our simulations than in theirs (according to the results reported in [9]). We compared our scheme with the one in [9], using pyramid and fixed segmentation with prefix schemes and we have shown that our scheme achieves much better results in all the performance metrics. We have concluded that this is attributed to the use of the Reference Frequency RF, which plays important role both in the replacement policy we use in Cache A and the replacement and admission policies we use in Cache B. In this way we attain a combination of an LRU and an LFU replacement policy. According to this LRLFU policy, the more frequently and the most recently a video file is requested the larger its caching value, and therefore the more difficult for it to be removed from the cache.

When a video is requested for the first time, the proxy always cache its initial Bp blocks as no information is available for that video to compare it with other videos in the cache. Once this video is cached in Cache A, we know its RF and timestamp T′ and therefore we can compute its caching value and update it at every subsequent request. If this video is popular enough, it can gradually bring its whole content inside the cache in subsequent requests, as in every request only one segment is considered for caching, dependent on its caching value. If it is not popular enough its Bp blocks will quickly be discarded from Cache A and the wrong decision to cache a small part of this video does not significantly affect the performance of the system. Popularity of each video is well captured by RF and timestamp, two metrics that adjust their values according to the client preferences.

3 Performance Evaluation

3.1 Performance Metrics

The main goal of the proxy server is to efficiently manage the cache capacity in order to reduce the required backbone rate. Byte hit ratio (BHR) is the primary metric that provides a direct measure of the savings in remote network and server bandwidth. It is defined as the fraction of total bytes that can be served directly from the proxy over the total bytes of media objects requested.

Playback delay is a very annoying effect to the clients and for that reason another important performance metric is the percentage of requests with delayed start. If a request for a video does not find the first initial Bp blocks in the cache A, it has a delayed start.

Another performance metric that we examined is the number of segment replacements during the simulation. The smaller the number of segment replacements is, the less the cache management overhead and the more effective the cache management policies are. However, due to loss of space we do not present results for this metric in the paper.

3.2 Simulation Model

We conducted event-driven simulations to evaluate the performance of a proxy with the cache management policy that was previously described. In order to compare our scheme with the one in [9], we have used the same system parameters, but we have also examined some additional scenarios .The Bp initial blocks cached for a video (prefix) was set to 32 blocks both in the pyramid and the fixed segmentation with prefix scheme.

We assume that client requests for videos arrive according to a Poisson process, therefore the inter-arrival times are exponentially distributed with mean λ. The default value of λ is 60.0 seconds. Videos are selected from a total of V distinct videos that are stored in the content server. The size of the videos is assumed uniformly distributed between 0.5M and 1.5M blocks, where M is the mean video size. The default value for M is 2,000. The playing time for a block is assumed to be 1.8 seconds, which means that the default playing time for a video is between 30 and 90 minutes, and that the playing time for a video prefix is almost one minute.

The popularity of each of the V videos is assumed to follow a Zipf-like distribution Zipf (s,V), where s corresponds to the degree of skew and V to the total number of videos in the content server. Every video x, x ∈ {1,….,V} has a probability given by $p_x = c / x^{1-s}$, where $c = 1 / \sum_{x=1}^{V} 1 / x^{1-s}$ is a normalization constant. For s=0 the distribution is highly skewed, while for s=1 the distribution is uniform with no skew. The default value for s is 0.2. We also assume that the popularity of each video changes over time, so that we examine the behavior of the cache when popularity changes occur. In particular, as in [9], it is assumed that the popularity distribution changes every R requests in our simulations. When it does, another well-correlated Zipf-like distribution with the same parameters s, and V, is used. The correlation between the two Zipf-like distributions is modeled by using a single parameter k that can take any integer value between 1 and V. First, the most popular video in Zipf-like distribution 1 is made to correspond to the r1-th most popular video in Zipf-like distribution 2, where r1 is chosen randomly between 1 and k. Then, the second most popular video in distribution 1 is made to correspond to the r2-th most popular video in distribution 2, where r2 is chosen randomly between 1 and min (V, k +1), except that r1 is not allowed, and so on.

Thus, k represents the maximum position in popularity a video title may shift from one distribution to the next. Hence, k = 1 corresponds to perfect correlation, and k = V to the random case or no correlation. In most of our simulation scenarios, we consider that k=10 and R=200. This means that we adopted a scenario where the popularities of videos change progressively, approximately every 3 hours. The change is slight, but happens very frequently. We have also examined scenarios with larger or steeper popularity changes.

The cache size is expressed in terms of the number of media blocks. The default size is assumed equal to 400,000 blocks. This means that in the default scenario, we consider a cache capacity equal to 10% of the video repository. The portion of the cache capacity that is used for the initial Bp blocks of videos (Cache A) is denoted by pr and its default value is 10%. All the parameter default values are shown in Table 1.

Table 1. System parameters and default values

Notation	***Definition (Default values)***
Bp	Initial blocks cached for a video (prefix=32 blocks)
λ	Mean request inter-arrival time (60 sec)
V	Number of distinct video titles (2,000)
M	Mean number of blocks per video (2,000 blocks)
s	Parameter s in Zipf distribution (0.2)
k	Maximum shifting distance for a hot video (10)
R	Number of requests between shifting of video Popularity distribution (200)
C	Total cache capacity (400,000 blocks)
pr	Portion of cache capacity used for storing initial blocks (10%)
Runs	Number of requests in a simulation run (100,000 requests)

3.3 Simulation Results

In our simulations, we compared our cache management policy with the cache management policy used in [9] when i) pyramid segmentation and ii) fixed segmentation with prefix is used. The prefix was chosen to be 32 blocks long in both segmentation schemes to facilitate result comparisons.

We examined the byte-hit ratio for a variety of fixed segment sizes (all multiples of 64 blocks) in the fixed segmentation with prefix scheme and we observed that the scheme works better with a larger segment size. This is due to the fact that an attentive cache admission policy is used, in order to cache a segment of a video. Therefore, when most of the decisions to cache a segment are right, it is preferable to cache a large segment. Throughout our simulations of the fixed segmentation with prefix scheme, we used the 32-960 case as our default case, where 32 is the size of the prefix in blocks and 960 is the size of the fixed segment also in blocks. We also examined the 32-960 case in our simulations, as the default fixed segmentation with prefix scheme for the cache management policy used in [9] which for brevity we denote by LRU-i because of the contribution of the number of segment i in an LRU policy. In our simulations of the 32-960 case and for video length smaller than 960 blocks, we considered that such videos are divided in two parts: the prefix which is 32 blocks and is cached in cache A and the suffix which consists of the remaining video and is cached in Cache B as one segment.

3.3.1 The impact of cache size, number of videos and mean video length.

Initially, we studied the performance of our scheme and the scheme in [9] with pyramid and fixed segmentation with prefix for different cache sizes, or different number of videos in the content server or different mean video lengths. For all these differentiations in the above parameters we observe that the pyramid and fixed segmentation with prefix schemes perform equally well in terms of byte-hit ratio and percentage of delayed starts in our scenario, but the pyramid segmentation scheme outperforms the fixed segmentation with prefix scheme in the scenario of [9] in terms of byte-hit ratio.

The percentage of delayed starts for the pyramid and fixed segmentation with prefix scheme is the same for a given cache management policy, as it depends only on the presence or not of the initial 32 blocks of the requested videos in Cache A, which is determined by the cache management policy and not by the segmentation scheme. Hence, our figures present the fraction of requests with delayed start only when the pyramid segmentation scheme is used.

It is remarkable that the LRLFU method performs better than LRU-i in terms of all the performance metrics for all the different cases examined. For the pyramid segmentation scheme, there is an 8% improvement in byte-hit ratio and 17% improvement in percentage of delayed starts when using LRLFU instead of LRU-i. For the fixed segmentation with prefix scheme the improvement is even larger. We note here that this improvement in byte-hit ratio is observed when the comparison is done between our scheme and the scheme in [9], simulated by us with an exhaustive

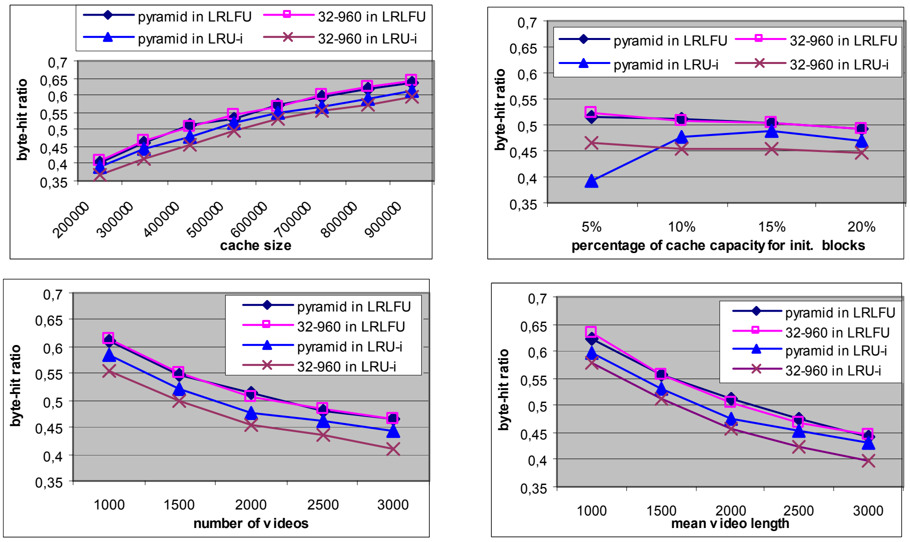

Fig. 2. Byte-hit ratio versus a) cache size b) percentage of cache capacity used for cache A c) number of total videos d) mean video length.

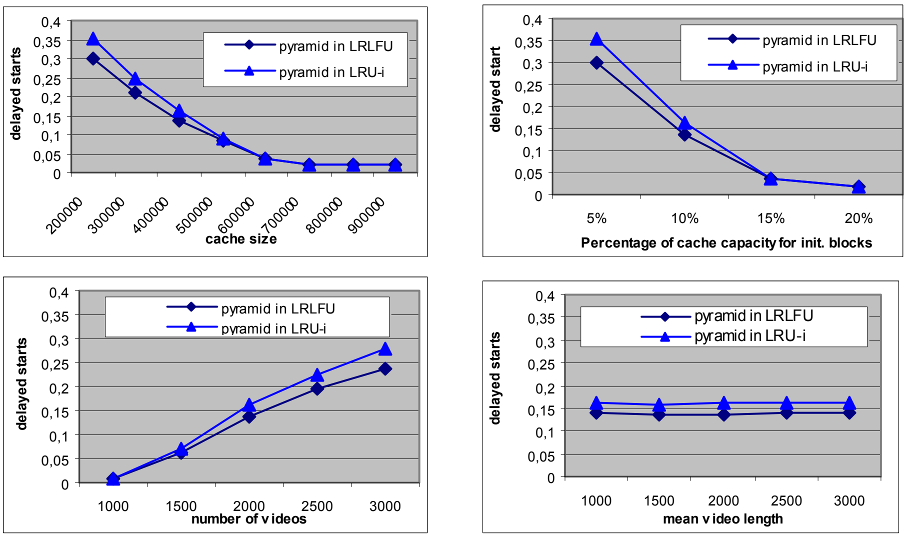

Fig. 3. Fraction of requests with delayed starts versus a) cache size b) percentage of cache capacity used for Cache A c) number of total videos d) mean video length.

search of segment victims in Cache B. Otherwise, direct comparison with the results in [9] shows that our scheme is 11% better in terms of byte-hit ratio.

From the results in Figure 2 we conclude that byte-hit ratio is better for larger cache sizes, smaller number of videos and smaller video lengths for all the scenarios examined, as it was expected. When we vary the size of Cache A, that is, the percentage of cache capacity used for storing initial blocks, we notice from Figure 2 (b) that for pr=5%, pyramid segmentation in LRU-i attains a much lower byte-hit ratio than in LRLFU. 32-960 scheme in LRU-i behaves better than pyramid in LRU-i but not as well as pyramid and 32-960 in LRLFU. These results reveal the superiority of our replacement policy in Cache A which decides smartly which videos to cache. When the size of Cache A is very small, not many prefixes can be cached and this also restricts the number of videos in general that can be cached in the proxy. As it has already been mentioned, only videos that are cached in Cache A can be cached in Cache B. 32-960 scheme in LRU-i performs better than pyramid in LRU-i for pr=5%, however, because the cache admission policy in Cache B succeeds to cache larger portion of popular videos with 32-960 scheme than with the pyramid scheme, before a change in popularity occurs and different videos are cached in Cache A.

The fraction of requests with delayed start depends only on the replacement policy in Cache A and the results presented in Figure 3 exhibit once again the superiority of our replacement policy in Cache A. When this fraction is near zero, Cache A has enough size to cache the prefixes of all the videos in the content server.

3.3.2 The impact of the parameters of the popularity distribution.

The results in Figure 4 show the impact of the degree of skew in video popularity on the byte-hit ratio and percentage of delayed starts. Small values of the Zipf parameter s correspond to a more skewed popularity distribution, i.e., more clients are interested in fewer videos. As the parameter s increases, i.e., client's preferences are dispersed among a plethora of videos, byte-hit ratio decreases while the percentage of delayed starts increase. However, LRLFU performs better than LRU-i even for large values of the parameter s.

The results in Figure 5 show the impact of maximum shifting distance k on byte-hit ratio and on delayed start. Maximum shifting distance k determines the extent of the popularity change once such a change occurs. We can see from Figure 5(a) that as k increases from k=10 to k=50, LRLFU and LRU-i perform almost identically in terms of byte-hit ratio. However LRLFU still provides better results in terms of delayed starts (Figure 5(b)).We have also examined cases where either a steeper change in popularity distribution occurs or large but less frequent popularity changes occur and we concluded that LRLFU is preferable when there are small and regular popularity changes, i.e., a progressive change in the popularities of videos, or large but less frequent popularity changes. For large and frequent changes in the popularities of the videos, an uncommon case in real systems, the performance of all the scenarios deteriorates, but among them the 32-960 in LRU-i scenario provides the best results (the corresponding results are not shown here due to space considerations).

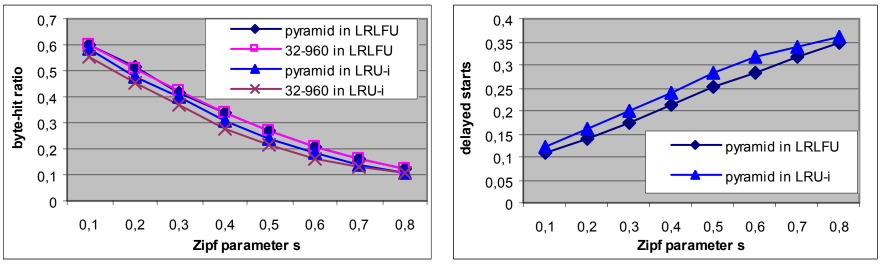

Fig. 4. The impact of skew in video popularity on (a) byte-hit ratio (b) delayed starts.

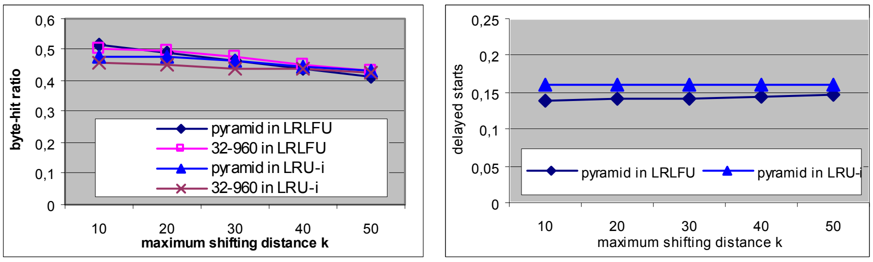

Fig. 5. The impact of maximum shifting distance on (a) byte-hit ratio (b) delayed starts.

4 Conclusions and Ongoing Work

In this paper, we have studied caching strategies for media objects in order to reduce the bandwidth requirements in the backbone links of the network and shield clients from delays and jitter on the server-proxy path. We considered the division of the cache into two parts, part A and part B. Replacement policy in part A decides which videos should be evicted from the cache in order for newly-requested videos to be cached, while replacement and admission policies in part B control the portion of the cached videos that should be inside the cache. All the above decisions are based on the caching value of each video which reflects its popularity. Popularity of each video is better captured when apart from the timestamp T′, which records the last time that a video has been requested, the number of requests for that video once it is cached, RF, is also considered in the caching value. Our simulation results have shown that the frequency considerations we have introduced in the caching value of the videos and the caching strategies that we used compared to the one in [9], improve the byte-hit ratio and significantly reduce the fraction of requests with delayed starts. he performance of our scheme deteriorates only for large and frequent changes in the popularity distribution, cases which are not expected to be common. Our simulation results have also shown that the fixed segmentation with prefix scheme with a large fixed segment size performs equally well with the pyramid segmentation scheme, when our cache management policies are used, in terms of byte-hit ratio and delayed starts. However pyramid segmentation outperforms the fixed segmentation with prefix scheme in terms of byte-hit ratio and delayed starts when the cache management policies of [9] are used. Therefore, when frequency considerations are used, pyramid segmentation can be avoided and the simpler fixed segmentation with prefix scheme can be used instead.

Our current work is focusing on the application of the above cache management policies and segmentation schemes in a collaborative environment of more than one proxy servers which serve homogeneous or even heterogeneous client communities. More specifically, a hierarchical tree topology system of proxies is considered where the prefixes of the videos are stored into small size proxy caches each located very close to the corresponding client community and larger caches located further away from the client communities. In such a collaborative system caches A and B are implemented in different proxy servers in a way that reduces the overall cache capacity needed to achieve certain performance goals.

References

[1] D.Eager, M.Ferris, M.Vermon,“Optimized regional caching for on-demand data delivery” in Proc. of Multimedia Computing and Networking Jan. 1999.

[2] Y. Guo D. Towsley, “Prefix caching assisted periodic broadcast: Framework and techniques to support streaming for popular videos”, Technical report, UM-CS-2001-022, Dept. of Computer Science, University of Massachusetts, May 2001.

[3] B. Wang, S. Sen, M. Adler, D. Towsley, “Optimal proxy cache allocation for efficient streaming media distribution” , in Proc. of the 2002 IEEE INFOCOM Conf., New York, NY, pages 1726--1735.

[4] C. Venkatramani, O. Verscheure, P. Frossard, K. Lee, “Optimal Proxy Management for Multimedia Streaming in - Content Distribution Networks”, NOSSDAV’02, May 12-14, 2002, Miami, Florida, USA.

[5] J. Kangasharju, F. Hartanto, M. Reisslein, K. W. Ross. “Distributing layered encoded video through caches”, in Proc. of the 2001 IEEE INFOCOM Conf., Anchorage, Alaska, April 2001, pages 1791--1800.

[6] R. Rejaie J. Kangasharju, “Mocha: A quality adaptive multimedia proxy cache for internet streaming”, in Proc. of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'01), June 2001, pages 3-10.

[7] S. Sen, J. Rexford, D. Towsley, “Proxy prefix caching for multimedia streams”, in Proc. of the 1999 IEEE INFOCOM Conf., pages 1310--1319.

[8] M. Almeida, D.L.Eager, M.K. Vernon,“A Hybrid Caching Strategy for Streaming Media Files” in Proc. of the MMCN '01, San Jose, CA, Jan. 2001.

[9] K.-L. Wu, P. S. Yu, J. L. Wolf, “Segment-based proxy caching of multimedia streams”, in Proc. of the 10th International WWW Conference, Hong Kong, 2001. An extended version titled “Segmentation of Multimedia Streams for proxy caching” will appear in IEEE Trans. on Multimedia.

[10] E. Balafoutis, A. Panagakis, N. Laoutaris, I. Stavrakakis, “The impact of replacement granularity on video caching”, in Proc. of the 2002 IFIP Networking Conf., Pisa, Italy, pages 214—225.

[11] Z.-L. Zhang, Y. Wang, D. H. C. Du, D. Su, “Video staging: A proxy-- server--based approach to end--to--end video delivery over wide--area networks”, IEEE/ACM Transactions on Networking, vol. 8, no. 4, pages 429-- 442, Aug. 2000.