TECHNICAL UNIVERSITY OF CRETE
DEPARTMENT OF ELECTRONIC AND COMPUTER ENGINEERING
Telecommunications and Information & Computer Networks Laboratories

'Frequency-Based Cache Management Policies for Collaborative and Non-Collaborative Topologies of Segment Based Video Caching Proxies'

Anna Satsiou

Supervisor:
Professor Michael Paterakis

Guidance Committe:
Professor Nicholas Sidiropoulos
Professor Vassilis Digalakis

M.Sc. Thesis

September 2004

Αφιερωμένο στους γονείς μου,
Ιωάννη και Μαρία Σάτσιου

Ευχαριστίες

Με την ολοκλήρωση της παρούσας μεταπτυχιακής διατριβής αισθάνομαι την ανάγκη να ευχαριστήσω πρώτα από όλα τον Θεό, που με αξίωσε να φέρω εις πέρας άλλον έναν από τους στόχους μου.

Ολόψυχα ευχαριστώ και τους γονείς μου Ιωάννη και Μαρία Σάτσιου, καθώς και τους αδερφούς μου Κώστα και Βαγγέλη Σάτσιο που με στηρίζουν πάντα με την αγάπη τους και μου δίνουν δύναμη. Είναι οι άνθρωποι στους οποίους οφείλω κάθε μου επιτυχία και τους υπεραγαπάω.

Επιπλέον, θα ήθελα να ευχαριστήσω το Ίδρυμα Ωνάση που με στήριξε οικονομικά καθ'όλη την διάρκεια των μεταπτυχιακών σπουδών μου, παρέχοντάς μου υποτροφία για μεταπτυχιακές σπουδές εσωτερικού.

Ευχαριστώ μέσα από την καρδιά μου όλους μου τους φίλους και φίλες από το Πολυτεχνείο Κρήτης, αλλά και αυτούς που βρίσκονται μακριά για την ψυχολογική στήριξη και υπέροχη φιλία τους.

Ευχαριστώ πολύ και τα μέλη της εξεταστικής μου επιτροπής, Καθηγητές κ. Νικόλαο Σιδηρόπουλο και κ. Βασίλη Διγαλάκη που υπήρξαν και καθηγητές μου σε μεταπτυχιακά μαθήματα για τις πολύτιμες γνώσεις και βοήθεια που μου παρείχαν.

Τέλος, οι πιο θερμές ευχαριστίες ανήκουν στον επιβλέπων της μεταπτυχιακής μου εργασίας Καθηγητή κ. Μιχάλη Πατεράκη, για το ενδιαφέρον του, τις πολύτιμες γνώσεις και συμβουλές του, καθώς και την βοήθεια και συμπαράσταση του. Είμαι πραγματικά πολύ χαρούμενη και το θεωρώ τιμή μου που συνεργάστηκα μαζί του!

Abstract

In response to the increasing demand for video and audio streaming over the Internet, partial caching of media objects in proxies located close to the clients is a useful technique which reduces the traffic in the backbone links of the network and shields clients from delays. However, most of the existing 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.

In the first part of this Thesis we introduce and evaluate a novel cache management policy which is capable of capturing the changing popularities of the various videos by attaching a caching value to each 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, (ii) significantly reduce the fraction of user requests with delayed starts and (iii) require less CPU overhead, when compared with previous work in this area.

The second part of the Thesis studies a collaborative environment of more than one proxy servers that serve homogeneous or even heterogeneous client requests for streaming of video files. While most of the research in the area of collaborative proxies uses a centralized coordinator which organizes and keeps track of the cache contents, the proxy servers in our hierarchical tree topology system of proxies act autonomously according to their type and location. Our simulation results show that the hierarchical tree topology of proxies achieves a much higher byte-hit ratio and better performance characteristics when using the same overall cache capacity with the simple topology of non-collaborative proxies that we examine in the first part of the Thesis.

Table of Contents

Chapter 1

Introduction

1.1 Introduction to Video Caching

Today's Internet has been increasingly used for carrying multimedia traffic, and on demand streaming for clients of asynchronous playback requests is amongst the most popular networked media services. Given the broad spectrum of streaming media applications, like Internet TV, corporate communications, distance education and entertainment video distribution, video streaming has attracted much attention with many practical deployments in recent years. 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.

To reduce client-perceived access latencies as well as server/network loads, an effective means is to cache frequently used data at proxies close to clients. This approach also enhances the availability of objects and mitigates packet losses, as a local transmission is generally more reliable than a remote transmission. Proxy caching thus has become one of the vital components in virtually all web systems. Streaming media, particularly those pre-stored, could also benefit significant performance improvement from proxy caching, given their static nature in content and highly localized access interests. However, existing proxies are generally optimized for delivering conventional web objects (*e.g.*, HTML pages or GIF images) and may not meet the requirements of steaming applications. The remainder Chapter 1 is organized as follows. In section 1.1.1, we list some important and unique features of streaming media and discuss their implications to proxy cache design. Section 1.1.2 specifies standard terminology of streaming media caching deployed in the Internet. Section 1.1.3 discusses a generic proxy caching architecture and some protocol considerations. In section 1.2 we describe and discuss related work on video caching. Finally, we underline the goal and contribution of this Thesis in section 1.3 and give its outline in section 1.4.

1.1.1 Features of streaming media

Huge size: A conventional web object is typically on the order of 1K to 100K bytes. Hence, a binary decision works well for proxy caching: either caching an object in its entirety or not caching. In contrast, a media object has a high data rate and a long playback duration, which combined yield a huge data volume. For illustration, a one-hour standard MPEG-1 video has a volume of about 675 MB; caching it entirely at a proxy is clearly impractical, as several such large streams would exhaust the capacity of the cache. One solution is to cache only portions of an object. In this case, a client's playback needs a joint delivery involving both the proxy and the origin server. So, the objects and the portions of the objects that must be cached, have to be carefully selected, such that the benefit of caching outweighs the overhead of the joint delivery.

Intensive bandwidth use: Streaming nature of delivery requires a significant amount of disk and network I/O bandwidth, sustained over a long period. Hence, minimizing bandwidth consumption becomes a primary consideration for proxy cache management, even taking precedence over reducing access latencies in many cases. Moreover, the bandwidth bottleneck limits the number of clients that a proxy can simultaneously support; employing multicast delivery and cooperation among proxies thus become particularly attractive for media streaming applications.

High interactivity: The long playback duration of a streaming object also enables various client-server interactions. As an example, recent studies found that nearly 90% media playbacks are terminated prematurely by clients [1]. In addition, during a playback, a client often expects VCR-like operations, such as fast-forward and rewind. This implies the access rates might be different for different portions of a stream, which potentially complicates the cache management.

Given these unique features of media objects, novel caching algorithms have been developed in the literature the most important of which are presented and briefly discussed in section 1.2.

1.1.2 Basic Terms on Video Caching

In this section we specify the standard terminology that is used on Video Caching.

Client: A program that establishes connections for the purpose of sending requests.

Server : An application program that accepts connections in order to service requests by sending back responses.

Cache: A program's local store of response messages and the subsystem that controls its message storage, retrieval, and deletion. A cache stores cacheable responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests. Any client or server may include a cache.

Origin server: The server on which a given resource resides or is to be created.

Proxy: An intermediary program that acts both as a server and as a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, with possible translation, to other servers. A proxy must implement both the client and server requirements of this specification.

1.1.3 Architecture and Protocols for Streaming Caching

Streaming applications generally support diverse client-server interactions and have stringent demands on packet delay and jitter to ensure discontinuity-free playback. To meet these requirements, the Internet Engineering Task Force (IETF) has developed the RTP/RTCP/RTSP protocol suite [23]. A generic system diagram of proxy-assisted media streaming using this suite is depicted in Figure 1.1.

In this system, the basic functionalities for data transferring are provided by the Real-Time Transport Protocol (RTP), including payload identification, sequence numbering for loss detection, and time stamping for playback control. Running on top of UDP, RTP itself does not guarantee Quality-of-Service (QoS), but relies on its companion, the Real-Time Control Protocol (RTCP), to monitor the network status and provide feedback for application-layer adaptation. The Real-Time Streaming Protocol (RTSP) coordinates the delivery of media objects and enables a rich set of controls for interactive playback. For the proxy-assisted streaming, the proxy has to relay these control messages between the client and the server. The problem is particularly involved if only part of a media object is cached at a proxy. In this case, the proxy must reply to the client PLAY request and initiate transmission of RTP and RTCP messages to the client for the cached portion, while request the uncached portion(s) from the server. Such fetching can be achieved through an RTSP range request specifying the playback points. The range request also enables clients to retrieve different segments of a media object from multiple servers or proxies, if needed.

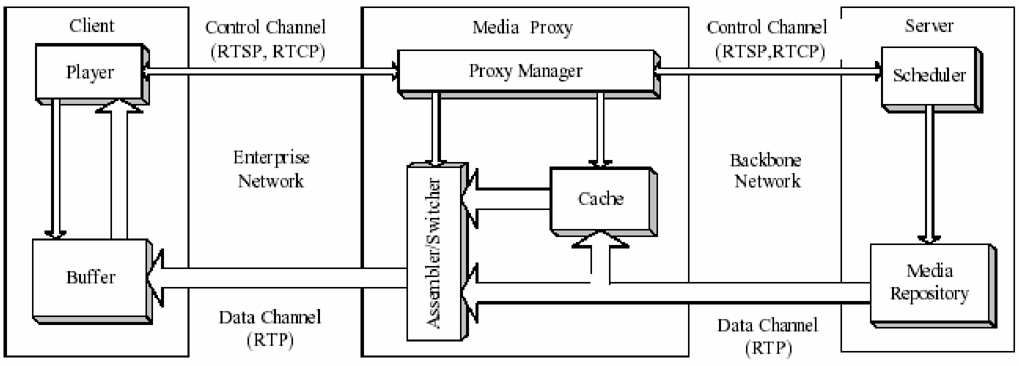


Figure 1.1: A generic system diagram of proxy-assisted media streaming using RTP/RTCP/RTSP.

1.2 Related Work on Video Caching

In order to support high quality streaming applications, 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]-[7]), where partial caching and i) periodic broadcast [2], [4],[5], or ii) techniques like batching, patching and stream merging [6],[7], are studied together.

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 [8]-[11]) and ii) the reduction of server loads, network traffic and user access latencies. Work in the second direction (see [12]-[19]) was mainly interested in partial caching, as storing the whole content of few media objects would exhaust the capacity of a conventional proxy cache.

Finally, research efforts have been made in collaborative caching (see [20] - [22]). A group of proxies can cooperate with each other to increase the aggregate cache space, balance loads, and improve system scalability. The aforementioned areas of research are described and discussed in the following sections.

1.2.1 Combining Proxy Caching with Multicasting [1-7]

Like caching, multicasting also explores the temporal locality of client requests. Specifically, it allows a media server to accommodate concurrent client requests with shared channels through batching, patching, or periodic broadcast. However, multicast delivery suffers from two important deficiencies. First, to save more bandwidth, it is better to accommodate more requests in one multicast channel by using a large batching/patching window, which however leads to long startup latencies. Second, while IP multicast is enabled in virtually all local-area networks, its deployment over the global Internet remains limited in scope and reach. Hence, it is unlikely that a multicast streaming protocol can be used for geographically dispersed servers and clients.

Interestingly, both deficiencies can be alleviated through the use of proxies. Specifically, a request can be instantaneously served by a cached prefix while waiting for the data from a multicast channel [1,7], and proxies can bridge unicast networks with multicast networks, *i.e.*, employing unicast for server to proxy delivery while batching and/or patching local accesses. Wang *et al*. [6] have derived the optimal length of the prefix to be cached for most typical multicast protocols, and showed that a careful coupling of caching and multicasting can produce significant cost savings over using the unicast service, even if IP multicast is supported only at local networks.

1.2.2 Stream Caching for Heterogeneous Clients [8-11]

Owing to diverse network models and device configurations, clients behind the same proxy often have quite different requirements on the same media object, in terms of streaming rates or encoding formats. To accommodate such heterogeneity, a straightforward solution is to produce replicated streams of different rates or formats, each targeting on a subset of clients. Though being widely used in commercial streaming systems, the storage and bandwidth demands of this approach can be prohibitively high [8]. An alternative is to transcode a media from one form to another of a lower rate or a different encoding format in an on-demand fashion [9]. The intensive computation overhead of transcoding however prevents a proxy from supporting a large, diverse client population.

Yet a more efficient approach to this problem is the use of layered encoding and transmission. A layered coder compresses a raw media object into several layers: the most significant layer, called the *base layer*, contains the data representing the most important features of the object, while additional layers, called *enhancement layers*, contain the data that can progressively refine the quality. A client thus can subscribe to a subset of cumulative layers to reconstruct a stream commensurate with its capability. For layered caching, Kangasharju *et al*. [10] assumed that the cached portions are semi-static and only completed layers are cached. To maximize the total revenue, they developed effective heuristics based on an analytical stochastic knapsack model to determine the cache content. In their model, the client population and the distribution of their capacities are known *a priori*. For layered streaming under dynamic conditions, Rejaie *et al*. [11] studied segment-based cache replacement and prefetching policies to achieve efficient utilization of cache space and available bandwidth (see Figure 1.2(a)). The main objective is to deal with the congestion problem for individual clients. To this end, the proxy keeps track of popularities of each object on a per layer basis. When the quality of the cached layers is lower than the maximum deliverable quality to an interested client, the proxy sends requests to the server for missing segments within a sliding prefetching window. On cache replacement, a victim layer is identified based on popularities, and its cached segments are flushed from the tail until sufficient space is obtained.

A critical drawback of the existing layered streaming systems is that the number of layers is pretty small, typically 2 or 3 only; hence, their adaptation granularity remains coarse. Fortunately, recent developments in the coding area have demonstrated the possibility of fine-grained postencoding rate control. An example is the MPEG-4 Fine-Grained Scalable (FGS) coder with bitplane coding, which generates embedded streams containing several bitplanes and each can be partitioned at any specific rate. As such, for narrowband clients, the proxy can reduce the streaming rate using a bitplane filter; for wideband clients, the proxy can fetch some uncached portion (*i.e.*, higher-order bitplanes) from the server and assemble it with the cached portion to generate a high-rate stream. As illustrated in Figure 1.2(b), the available bandwidth of a client can be almost fully utilized, and, more importantly, both the filtering and the assembling operations in FGS can be done with fast response. Several caching algorithms have been proposed to minimize the bandwidth consumption and/or improve the client utility [8].

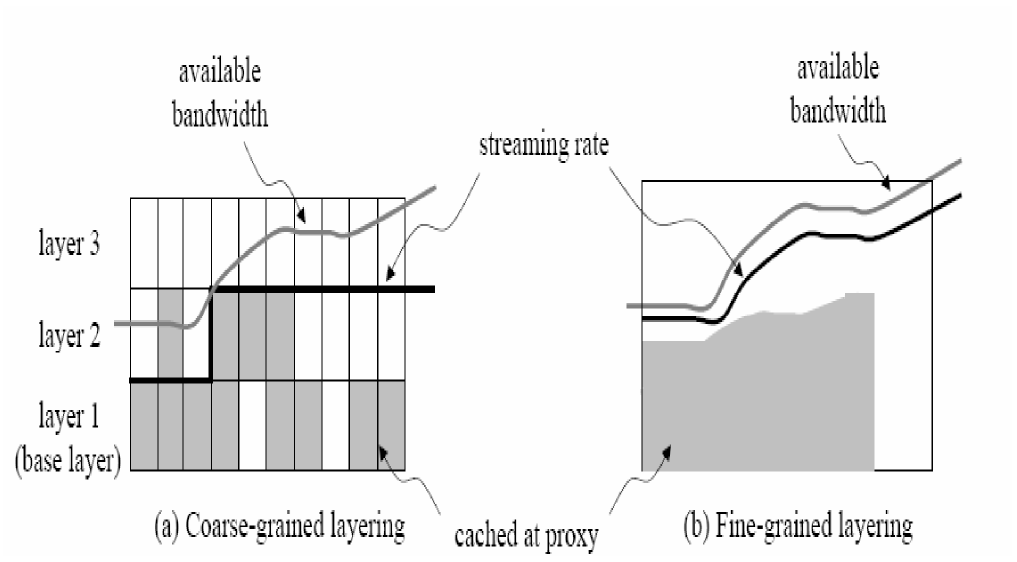
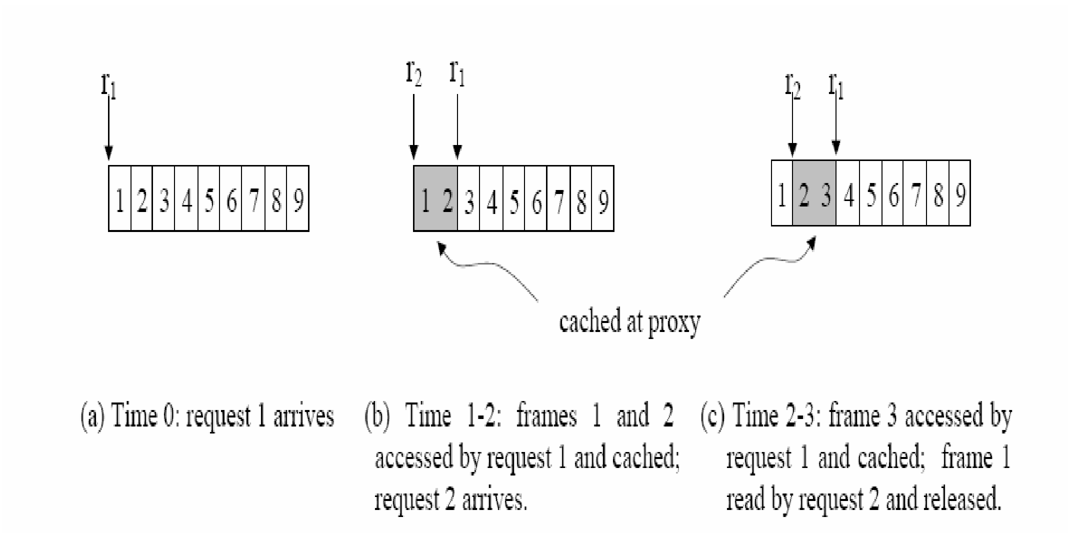


Figure 1.2: Caching for layered streaming (a) Coarse-grained layering, (b) Fine-grained layering.

1.2.3 Sliding Interval Caching [15]

This algorithm caches a sliding interval of a media object to exploit sequential access of streaming media. For illustration, given two consecutive requests for the same object, the first request may access the object from the server and incrementally store it into the proxy cache; the second request can then access the cached portion and release it after the access. If the two requests arrive close in time, only a small portion of the media object needs to be cached at any time instance, and yet the second request can be completely satisfied from the proxy (see Figure 1.3). In general, if multiple requests for an object arrive in a short period, a set of adjacent intervals can be grouped to form a *run*, of which the cached portion will be released only after the last request has been satisfied.

Sliding-interval caching can significantly reduce network bandwidth consumption and start-up delay for subsequent accesses. However, as the cached portion is dynamically updated with playback, the sliding-interval caching involves high disk bandwidth demands; in the worse case, it would double the disk I/O due to the concurrent read/write operations. In addition, its effectiveness diminishes with the increase of the access intervals. If the access interval of the same object is longer than the duration of the playback, the algorithm is degenerated to the unaffordable fullobject caching. To address these issues, it is preferable to retain the cached content over a relatively long time period. Most of the caching algorithms to be discussed in the rest of this section fall into this category.



r_1
1 2 3 4 5 6 7 8 9
r_2 r_1
1 2 3 4 5 6 7 8 9
r_2 r_1
1 2 3 4 5 6 7 8 9
cached at proxy
(a) Time 0: request 1 arrives
(b) Time 1-2: frames 1 and 2 accessed by request 1 and cached; request 2 arrives.
(c) Time 2-3: frame 3 accessed by request 1 and cached; frame 1 read by request 2 and released.

Figure 1.3: An illustration of sliding-interval caching. The object consists of 9 frames, each requiring one unit time to deliver from the proxy to a client. Requests 1 and 2 arrive at times 0 and 2, respectively. To serve request 2, only two frames need to be cached at any time instance.

1.2.4 Prefix Caching [13]

This algorithm caches the initial portion of a media object, called *prefix*, at a proxy. Upon receiving a client request, the proxy immediately delivers the prefix to the client and, meanwhile, fetches the remaining portion, the *suffix*, from the server and relays it to the client (see Figure 1.4). As the proxy is generally located closer to the clients than the origin server, the start-up delay for a playback can be remarkably reduced.

To ensure discontinuity-free playback with a start-up delay of s, the proxy has to store a prefix of length max{*dmax*-s ,0} , where *dmax* is the maximum delay from the server to the proxy. If cache space is abundant, the proxy can also devote some space to assist in performing *workahead smoothing* for variable-bit-rate (VBR) media

[13]. With this smoothing cache, the proxy can prefetch large frames in advance of each burst to absorb delay jitter and bandwidth fluctuations of the server-to-proxy path. The delay of prefetching can be hided by the prefix caching. Similar to sliding-interval caching, the content of the smoothing cache is dynamically updated with playback. However, the purposes are different: the former is to improve cache hit for subsequent requests, while the latter is to facilitate workahead smoothing.

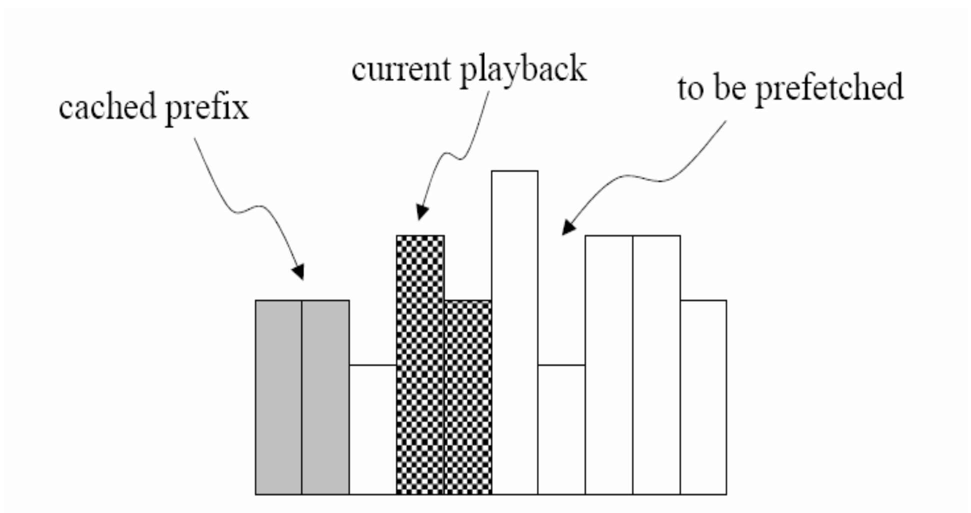


Figure 1.4: A snapshot of prefix caching with workahead smoothing.

1.2.5 Segment Caching [12,16, 17, 18, 19]

Segment caching generalizes the prefix caching paradigm by partitioning a media object into a series of segments, differentiating their respective utilities, and making caching decisions accordingly (see Figure 1.5). Various segment caching algorithms have been proposed in the literature by employing different segmentations and utility calculations. Wu *et al.* ([16], [18]) suggested to group the frames of a media object into variable-sized segments, with the length increasing exponentially with the distance from the start of the media, *i.e.*, the size of segment *i* is 2^{i-1} in frames , it consists of frames 2^{i-1}, $2^{i-1}+1$,…, 2^{i} - 1. The motivation is that a proxy can quickly adapt to the changing access patterns of cached objects by discarding big chunks as needed. Furthermore, 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. 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. In [12], the video segmentation approach is studied in general, and the trade-off between byte-hit ratio (BHR), which is defined as the fraction of total bytes that can be served directly from the proxy over the total bytes of media objects requested, and responsiveness in popularity changes is examined for fixed and variable video segmentation schemes. However, the authors in [12] apply a simple LRU replacement policy to videos that are not currently played and in contrast to [16] they do not adopt the idea of dedicating a portion of the cache for caching prefixes of the various videos.

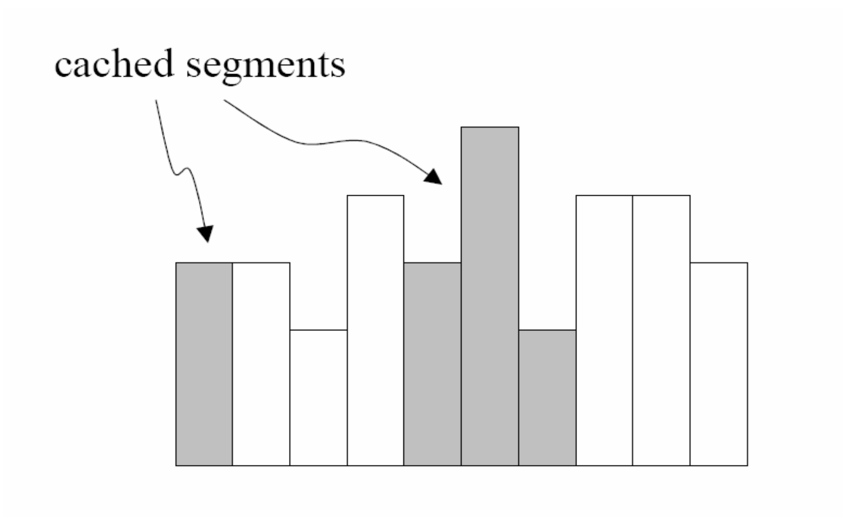


Figure 1.5: An illustration of segment caching.

A salient feature of segment-based caching is its support to VCR-like operations, such as random access, fast-forward, and rewind. As an example, Fahmi *et al.* [19] proposed to cache some key segments of a media object, called *hotspots*, which are identified by content providers. When a client requests the object, the proxy first delivers the hotspots to provide an overview of the stream; the client can then decide whether to play the entire stream or quickly jump to some specific portion introduced by a hotspot. Furthermore, in fast-forwarding and rewinding operations, only the corresponding hotspots are delivered and displayed, while other portions are skipped. As such, the load of the server and backbone network can be greatly reduced, but the client will not miss any important segments in the media object. However, while hotspot caching improves the user's perceived response time because the proxy serves more requests than the server, is not as effective as conventional caching techniques in reducing the server load. Even for popular video files, the proxy caches only a fraction of the file while the server must supply the remaining portion.

1.2.6 Rate-Split Caching [14]

While all the aforementioned caching algorithms partition a media object horizontally along the time axis, the rate-split caching partitions a media vertically along the rate axis: the upper part will be cached at the proxy, whereas the lower part will remain stored at the origin server (see Figure 1.6). This type of partitioning is particularly attractive for VBR streaming, as only the lower part of a nearly constant rate has to be delivered through the backbone network. For a QoS network with resource reservation, the bandwidth reserved should be equal to the peak rate of a stream; caching the upper part at the proxy clearly reduces the rate variability and improves the backbone bandwidth utilization. A critical issue here is how to select the cut-off rate or, equivalently, the size of the upper part for caching. Zhang *et al.* [14] studied the impact of the cut-off rate for a single stream through empirical evaluation, and found that a significant bandwidth reduction can be achieved with a reasonably small cache space. They also formulated the multiple-stream case as a knapsack problem with two constraints: disk bandwidth and cache space, and developed several heuristics, *e.g.*, caching popular objects only, or caching those with high bandwidth reduction.

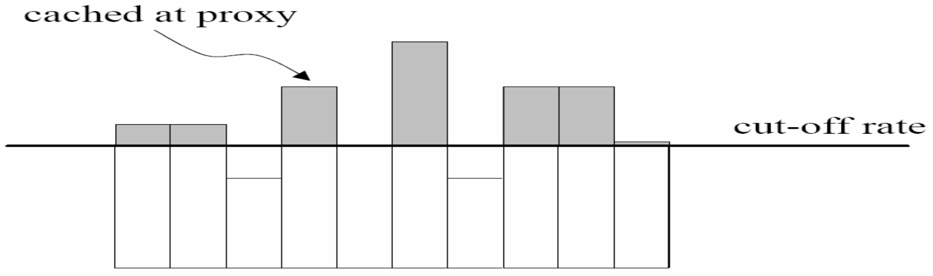


Figure 1.6: An illustration of rate-split caching.

1.2.7 Cooperative Proxy Caching

In general, proxies grouped together can achieve better performance than independent standalone proxies. Specifically, the group of proxies can cooperate with each other to increase the aggregate cache space, balance loads, and improve system scalability ([20]-[22]). A typical cooperative media caching architecture is Middleman [21], which operates a collection of proxies as a scalable cache cluster. Media objects are segmented into equal-sized segments and stored across multiple proxies, where they can be replaced at a granularity of a segment. There are also several local proxies responsible to answer client requests by locating and relaying the segments. Note that, in cooperative web caching, a critical issue is how to efficiently locate web pages with minimum communication costs among the proxies. This is, however, not a major concern for cooperative media caching, as the bandwidth consumption for streaming objects is of orders of magnitude higher than that for object indexing and discovering. Consequently, in Middleman, a centralized coordinator works well in keeping track of cache states. On the other hand, while segment-based caching across different proxies facilitates the distribution and balance of proxy loads, it incurs a significant amount overhead for switching among proxies to reconstruct a media object. To reduce such effects as well as to achieve better load balance and fault tolerance, Chae *et al.* [22] suggested a Silo data layout, which partitions a media object into segments of increasing sizes, stores more copies for popular segments, but still guarantees at least one copy stored for each segment

1.3 Thesis Goal and Contribution

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. Therefore, a cache management policy must be designed capable of capturing the changing popularities and acting accordingly. In [16]-[18]

dynamic caching mechanisms, which are robust for evolving client workloads, are examined and tested by applying a change in the popularity distribution of the videos every R requests. As already mentioned in section 1.2.5, partial caching is used in [16] 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.

The first part of our work [17] is based on some ideas from [16]. Firstly, it uses the same mechanism of change in the popularity distribution of the videos to test the adaptability of the cache management policies we introduce. Secondly, 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 [16] cache management policy. In particular, we apply a different replacement policy than in [16] in part A of the cache and a different replacement and admission policy in part B of the cache, by introducing frequency considerations in the caching values. Furthermore, apart from the pyramid segmentation scheme used in [16], we also examine a different segmentation scheme according to which a video is divided into a prefix of a small number of initial blocks and a suffix of fixed-size segments. We compare our results with the work in [16]. Our work aims at i) improving the byte-hit ratio (BHR) at the proxy, thus reducing the average backbone network load, irrespectively of the transmission scheme that is used, ii) reducing the fraction of user requests with delayed starts (a request for a video has a delayed start if it does not find the first initial blocks of the video inside the cache), and iii) requiring the least possible cache management overhead, and therefore the least CPU overhead.

The second part of our work studies a collaborative environment of more than one proxy servers that serve homogeneous or even heterogeneous client requests for streaming of video files. Most of the research in the area of collaborative proxies uses a centralized coordinator ([20], [21]) which organizes and keeps track of the cache contents. However, the centralized approach does not scale well. Firstly, the several interactions with the centralized coordinator lead to high latencies and secondly, the coordinator can quickly become the bottleneck and a single point of failure. In contrast, the proxy servers in our hierarchical tree topology system of proxies act autonomously according to their type and location without the need of a centralized coordinator. Specifically, the prefixes of the videos are stored in small size proxy caches each located very close to the corresponding client community, while larger caches located further away from the client communities are used to cache the latter segments of the videos requested by more than one client communities. The proxies efficiently cache the video contents without the need to know the state of the other proxies. Frequency-Based Cache Management Policies are used in order to efficiently and dynamically cache the content of the most popular videos among the various proxies. Our event-driven simulations have shown that the hierarchical tree topology of proxies achieves a much higher byte-hit ratio when using the same overall cache capacity with the simple topology of non-collaborative proxies that we examine in the first part of our work. We also examine the cache management policies that are used in [16] when applied to our collaborative environment and compare them with ours.

1.4 Thesis Outline

The remainder of the Thesis is structured as follows. In Chapter 2 we present the first part of our work. Section 2.2 presents the system model and describes the segmentation schemes and cache management policies that we use. Section 2.3 presents our event-driven simulation model, system parameters and performance results. The same section contains the discussion of the results.

In Chapter 3 we present the second part of our work. Similarly, section 3.2 presents the collaborative topology of proxies and the cache management policies that are used. Section 3.3 presents our event-driven simulation model, system parameters and examined scenarios. Finally, section 3.4 contains the performance results and their discussion.

Finally, Chapter 4 contains our concluding remarks, the research contribution of the Thesis and some ideas for future work.

Chapter 2

Design and Performance Evaluation of Frequency-Based Cache Management Policies for Segment Based Video Caching Proxies

2.1 Introduction

This chapter introduces and evaluates the performance of a cache management policy, applied to groups of blocks of the media files which form the segments of the media files according to two different segmentation schemes. The 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. The cache is divided in two parts where the replacement policy in the first part determines which videos should be cached in the proxy in the form of a fixed length prefix, and the replacement and admission policies in the second part determine the additional portion of these videos beyond the prefix that should be inside the cache. Our event-driven simulations show that the frequency considerations we have introduced in the caching values attached to the videos (i) improve the byte-hit ratio, (ii) significantly reduce the fraction of user requests with delayed starts and (iii) require less CPU overhead, when compared with previous work in this area.

2.2 System Description

2.2.1 Network Topology

Figure 2.1 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.

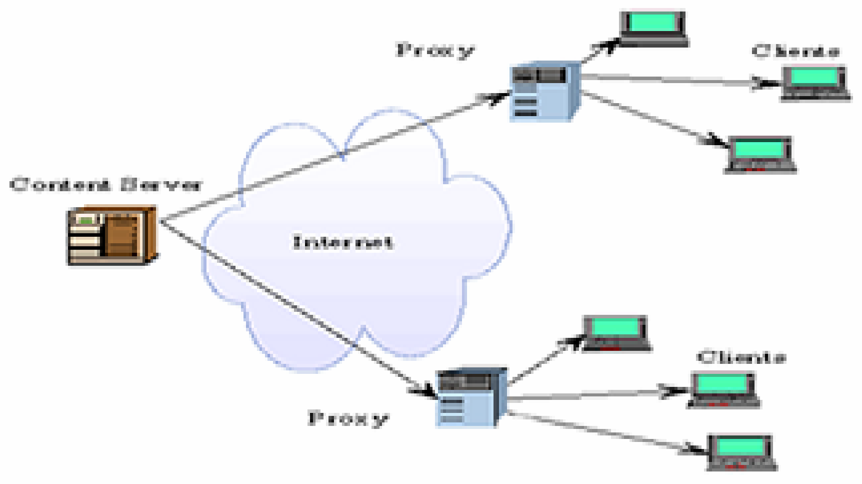


Figure 2.1: Streaming video in the Internet.

2.2.2 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}, $b^{i-1} + 1, \cdots b^i - 1$, if $i > 0$. Segment S_0^p contains block 0. In our simulations, we examined the pyramid segmentation scheme for b=2.

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'.

2.2.3 Cache management policies

As we already mentioned in the introduction, we have adopted from [16] 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. Bp should be determined by the fact that enough of the non-cached segments can be fetched in time from the content server in order to guarantee continuous streaming to the user once it is started. 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.

2.2.3.1 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 [16], 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.

2.2.3.2 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 [16] 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 [16], 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 [16], 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 [16], 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 [16]). We compared our scheme with the one in [16], 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.

We refer to our cache management policies as the LRLFU scheme because of the combination of an LRU and an LFU replacement policy that we attain, while we denote by LRU-i the scheme of [16] because of the contribution of the number of segment i in an LRU policy.

2.3 Performance Evaluation

2.3.1 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 study, not as important as the others, 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, because the latter determine the segment victims more accurately. We consider the replacement of the Bp blocks as one action, therefore as one segment replacement both in pyramid and fixed segmentation with prefix schemes, while all the other segment replacements that occur in Cache B are added to the metric.

2.3.2 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 [16], 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. The only difference is that in pyramid segmentation the initial 32 blocks form 6 segments, while in the fixed segmentation with prefix scheme form one segment. However, when they are ejected from Cache A they are considered as one segment replacement.

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 [16], 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 2.1.

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 video blocks (10%)
Runs	Number of requests in a simulation run (100,000 requests)

Table 2.1: System parameters and default values

2.3.3 Simulation results

In our simulations, we compared our cache management policy with the cache management policy used in [16] 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. Despite the fact that the 32 initial blocks of a video form 6 segments in the pyramid segmentation scheme, in contrast to 1 segment in the fixed segmentation with prefix scheme, the eviction of these Bp blocks from the cache is done in one action and is considered to correspond to one segment replacement.

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. As it can be seen from Figure 2.2, the scheme with segment size equal to 512 blocks and 960 blocks behaves equally well in terms of byte-hit ratio, but we preferred to use a segment size of 960 blocks in order to attain fewer segment replacements and also because we found that the case with 960 blocks behaves better than the one with 512 blocks for larger than the default mean video length. Therefore, 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 should further note that a larger segment size is also preferable when the cache admission policy of [16] is used, as it can be seen from Figure 2.2. 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 [16] 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.

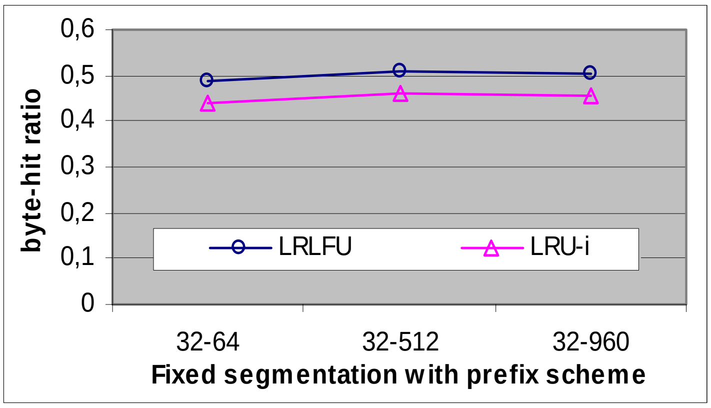


Figure 2.2: The impact of fixed segment size on byte-hit ratio

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

Initially, we studied the performance of our scheme and the scheme in [16] 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 [16] in terms of byte-hit ratio. However, fixed segmentation with prefix scheme achieves much fewer segment replacements in both scenarios.

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 with 33% fewer segment replacements 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 [16], simulated by us with an exhaustive search of segment victims in Cache B. Otherwise, direct comparison with the results in [16] shows that our scheme is 11% better in terms of byte-hit ratio.

From the results in Figures 2.3, 2.4 and 2.5 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.

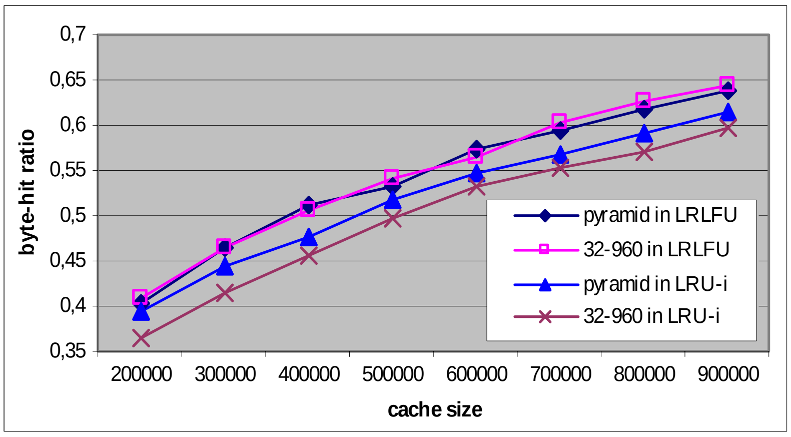


Figure 2.3: Byte-hit ratio versus cache size

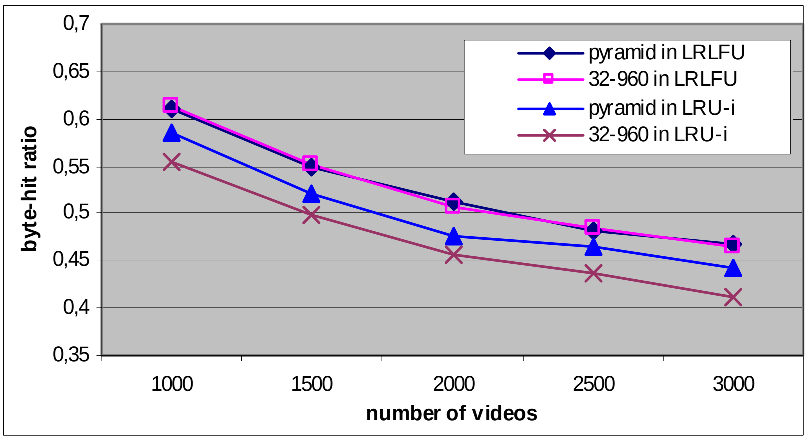


Figure 2.4: Byte-hit ratio versus number of total videos

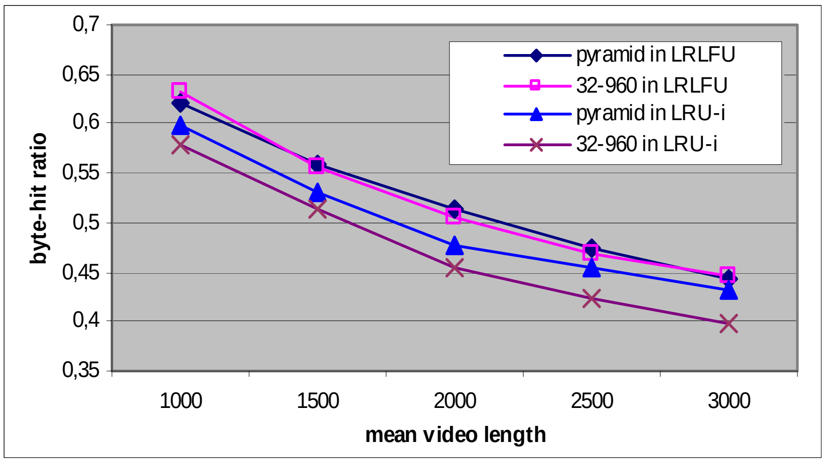


Figure 2.5: Byte-hit ratio versus mean video length.

When we vary the size of Cache A, that is, the percentage of cache capacity used for storing initial blocks, we notice from Figure 2.6 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.

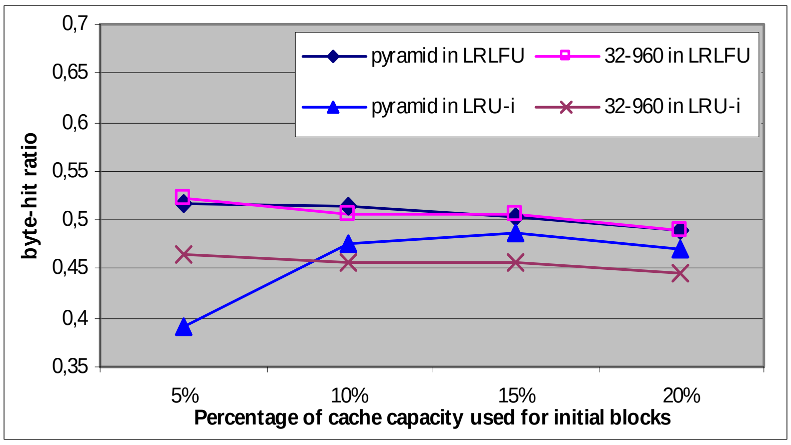


Figure 2.6: Byte-hit ratio versus percentage of cache capacity used for Cache A videos

The fraction of requests with delayed start depends only on the replacement policy in Cache A and the results presented in Figures 2.7, 2.8, 2.9 and 2.10 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.

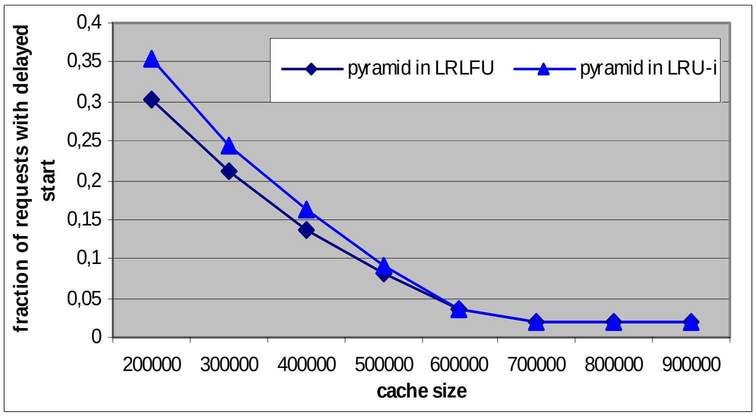


Figure 2.7: Fraction of requests with delayed starts versus cache size

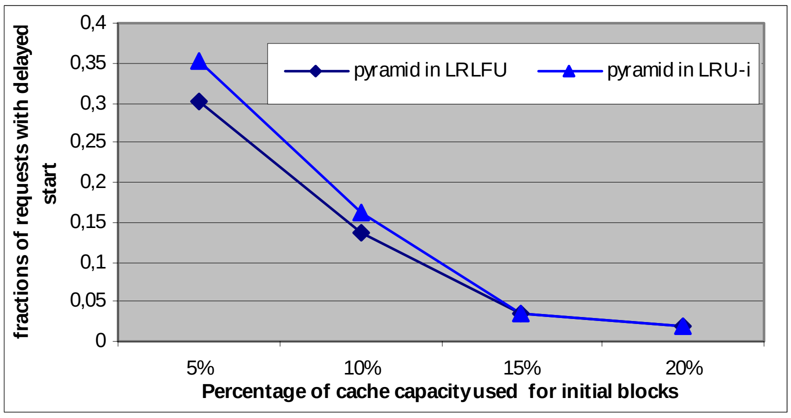


Figure 2.8: Fraction of requests with delayed starts versus percentage of cache capacity used for Cache A

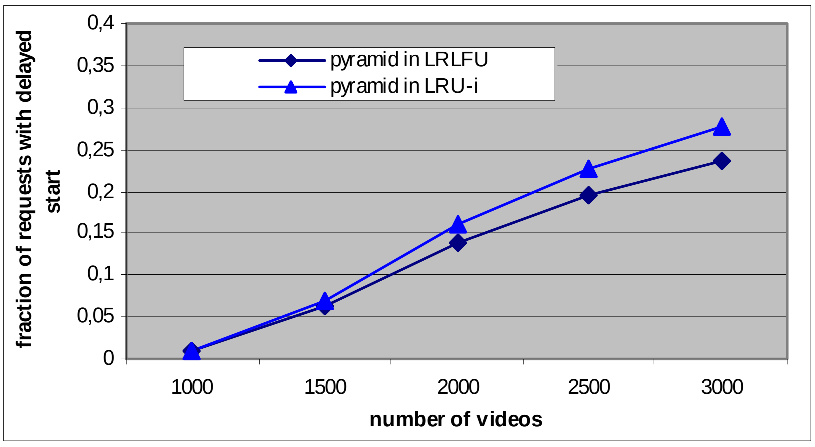


Figure 2.9: Fraction of requests with delayed starts versus number of total videos

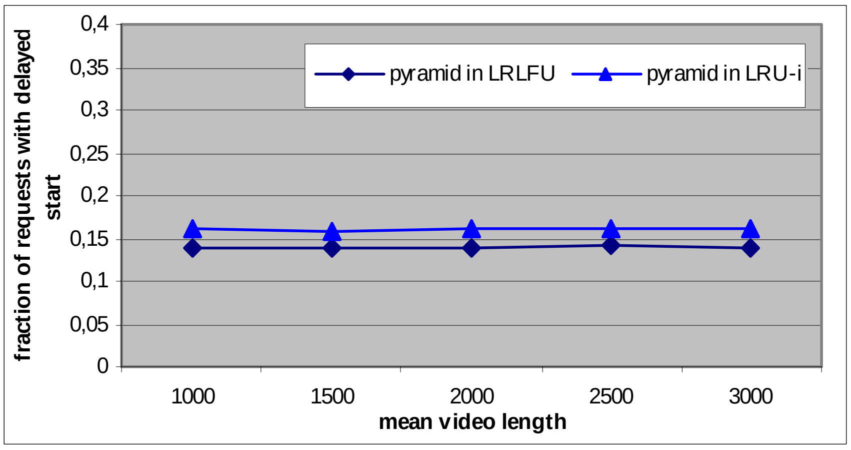


Figure 2.10: Fraction of requests with delayed starts versus mean video length.

For pr=20%, the prefixes of all the videos that we have considered are cached in Cache A. Therefore, the segment replacements that are presented in Figure 2.12 for pr=20% are mainly due to segment replacements in Cache B. These results demonstrate the superiority of our cache replacement and admission policies in Cache B, as much fewer segment replacements are needed in order to achieve a higher byte-hit ratio than the one achieved by LRU-i. Our cache management policies are more accurate, because of the frequency considerations based on which they cache segments of a video, and a segment is replaced only when this is necessary. Fewer segment replacements contribute to less CPU overhead. The fixed segmentation with prefix scheme achieves much fewer segment replacements as larger portions of videos are replaced in one action because of the large segment size of 960 blocks that is used in Cache B.

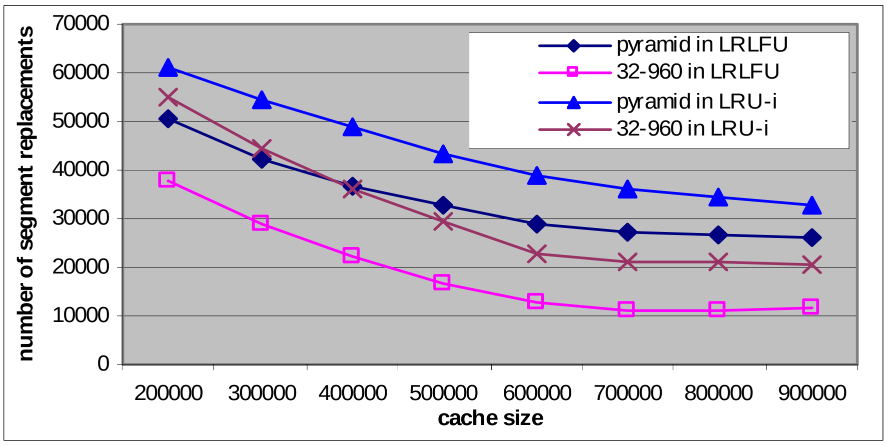


Figure 2.11: Segment replacements versus cache size

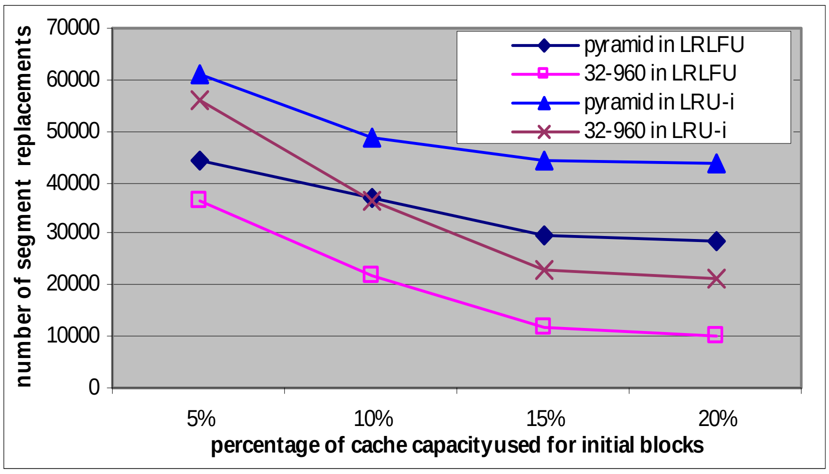


Figure 2.12: Segment replacements versus percentage of cache capacity used for Cache A

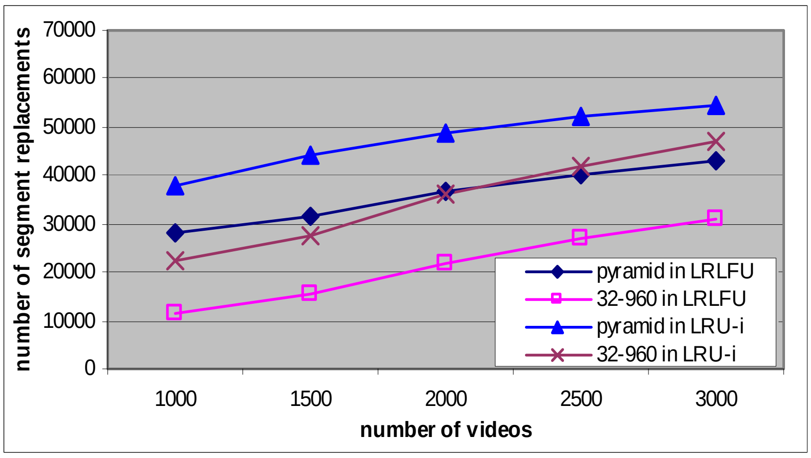


Figure 2.13: Segment replacements versus number of total videos

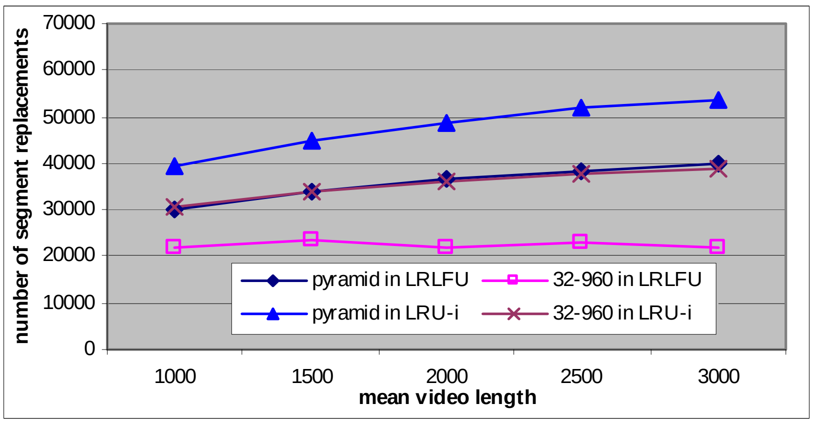


Figure 2.14: Segment replacements versus mean video length

2.3.3.2 The impact of the parameters of the popularity distribution.

The results in Figures 2.15, 2.16 and 2.17 show the impact of the degree of skew in video popularity on the byte-hit ratio, percentage of delayed starts and segment replacements correspondingly. 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 and segment replacements increase. However, LRLFU performs better than LRU-i even for large values of the parameter s.

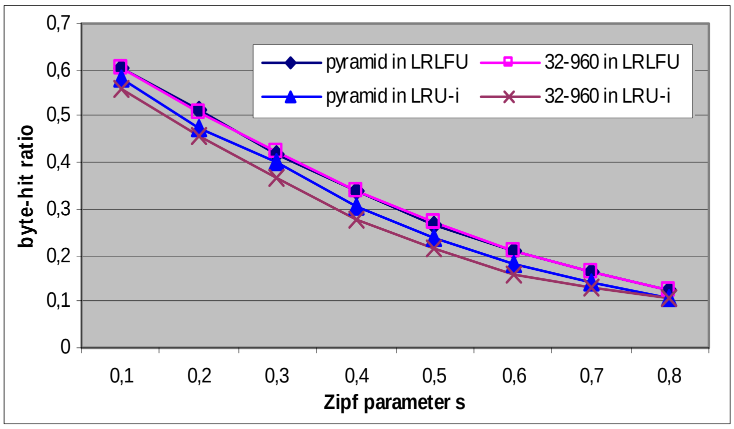


Figure 2.15: The impact of skew in video popularity on byte-hit ratio

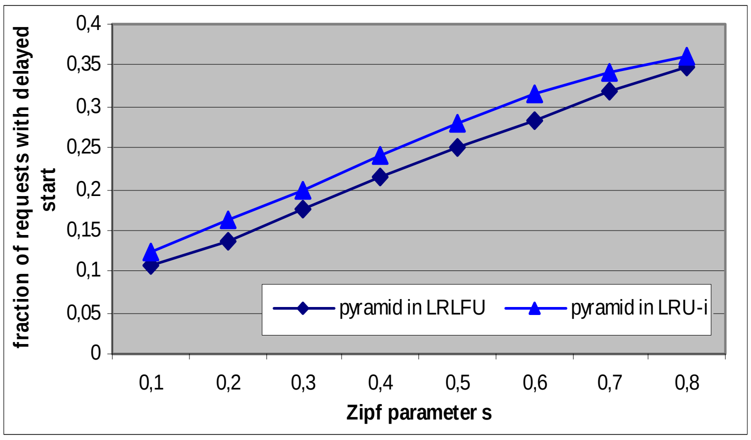


Figure 2.16: The impact of skew in video popularity on delayed starts

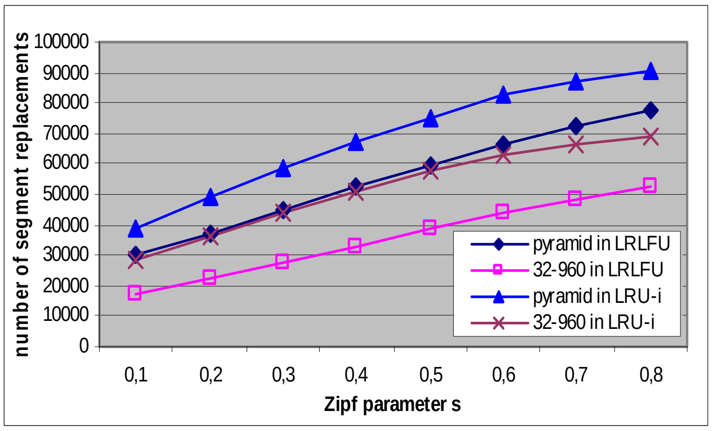
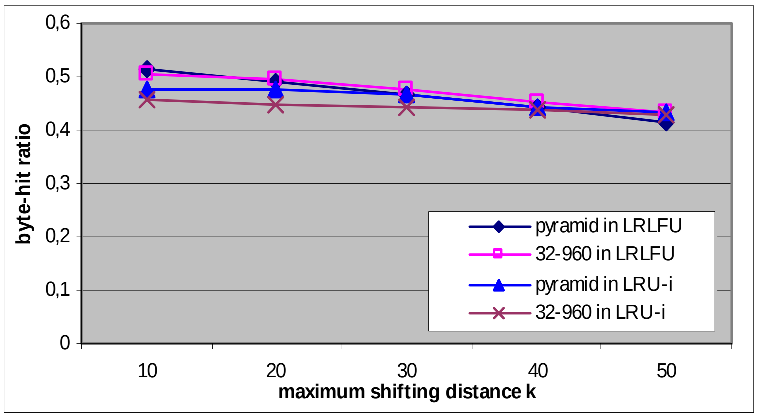


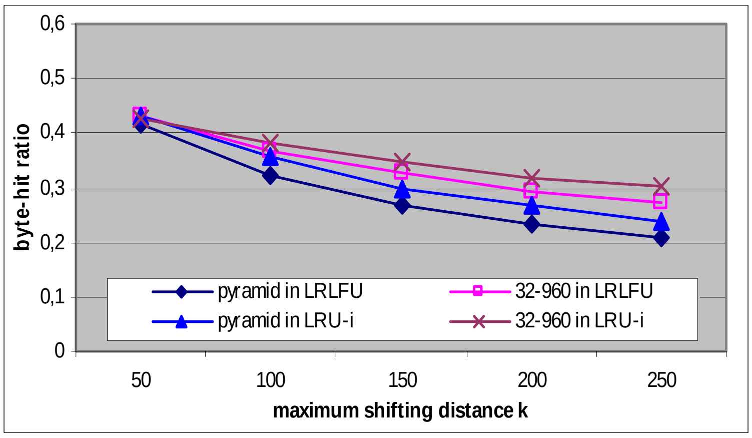
Figure 2.17: The impact of skew in video popularity on segment replacements.

The results in Figures 2.18, 2.19 and 2.20 show the impact of maximum shifting distance k on byte-hit ratio, on delayed start and on segment replacements, accordingly. Maximum shifting distance k determines the extent of the popularity change once such a change occurs. We can see from Figure 2.18 (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 2.19 (a)) and segment replacements (Figure 2.20 (a)). The popularity distribution changes in a steeper way when k takes larger values. For steeper changes in popularity, the corresponding results are presented in Figures 2.18(b), 2.19(b) and 2.20 (b). LRU-i appears to behave better than LRLFU both in terms of byte-hit ratio and delayed starts, but not in terms of segment replacements. It is remarkable that the 32-960 segmentation scheme in LRU-i presents the best byte-hit ratio results among all the scenarios and this is the only case where the 32-960 scheme in LRU-i performs better than the pyramid segmentation scheme in LRU-i. This behavior is due to the fact that when there are steep popularity changes occurring in small time durations (R=200) the reference frequencies, RF, of the videos fail to reach their steady state values, which capture the popularities of the various videos stored in the cache, before the next change in popularity distribution occurs.

In such case, the difference between the current time and the timestamp indicates how recently a video has been requested and it better reflects the popularity of the video. 32-960 scheme in LRU-i is even more preferable because it quickly fills the cache with large portions of the most recently requested videos, before a new change in the popularities of the videos occurs.

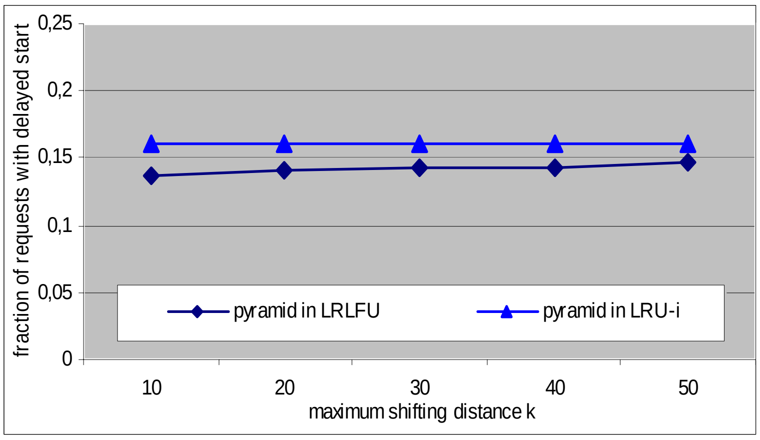


(a)

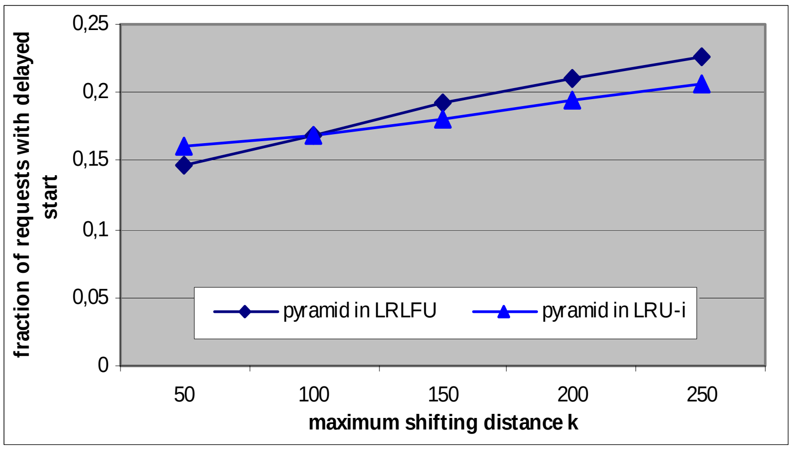


(b)

Figure 2.18: The impact of maximum shifting distance on byte-hit ratio.

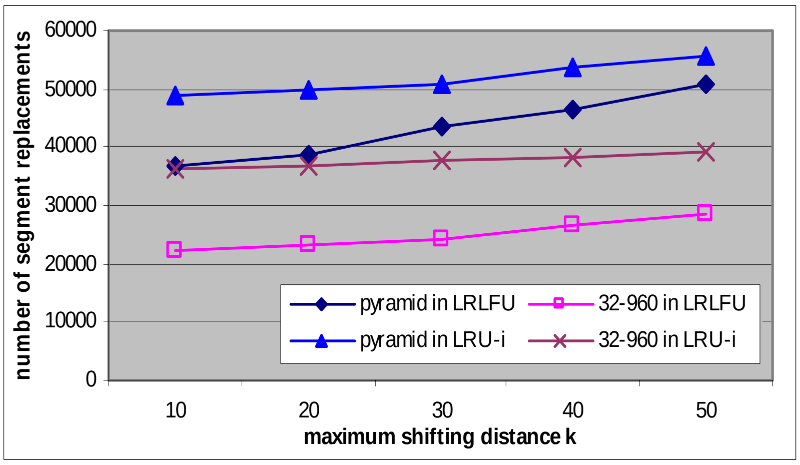


(a)

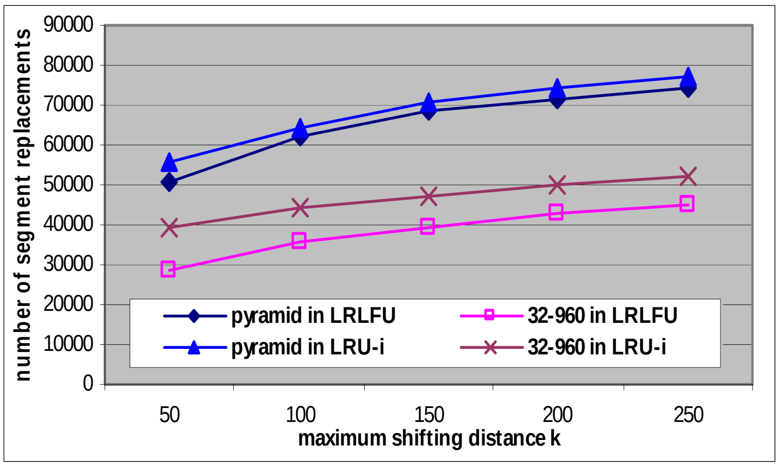


(b)

Figure 2.19: The impact of maximum shifting distance on delayed starts.



(a)



(b)

Figure 2.20: The impact of maximum shifting distance on segment replacements.

From the results in figure 2.18 (b) we observe that for k=150 and R=200, the 32-960 scheme in LRU-i presents the best results followed by the 32-960 scheme in LRLFU. However, if we keep k=150 and increase the number of requests between popularity changes, R, we can observe that LRLFU progressively performs identically with LRU-i in terms of byte-hit ratio while it appears once again superior to LRU-i in terms of delayed start and segment replacements. These results are presented in Figures 2.21, 2.22 and 2.23. For even larger values of R, LRLFU surpasses LRU-i in terms of all the metrics.

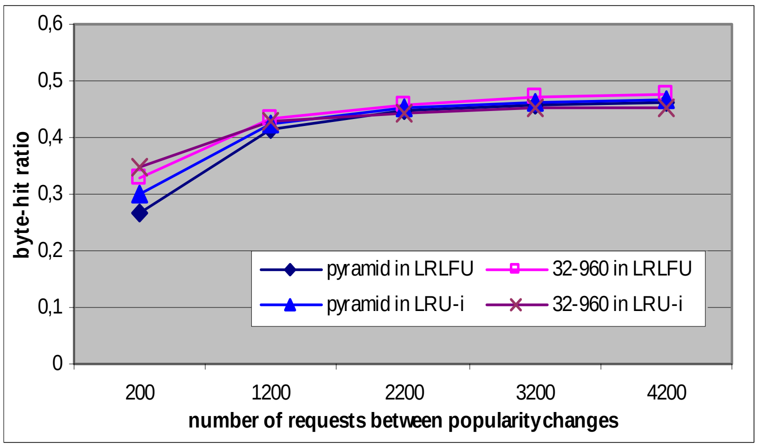


Figure 2.21: The impact of the number of requests between successive popularity changes on byte-hit ratio, for k=150.

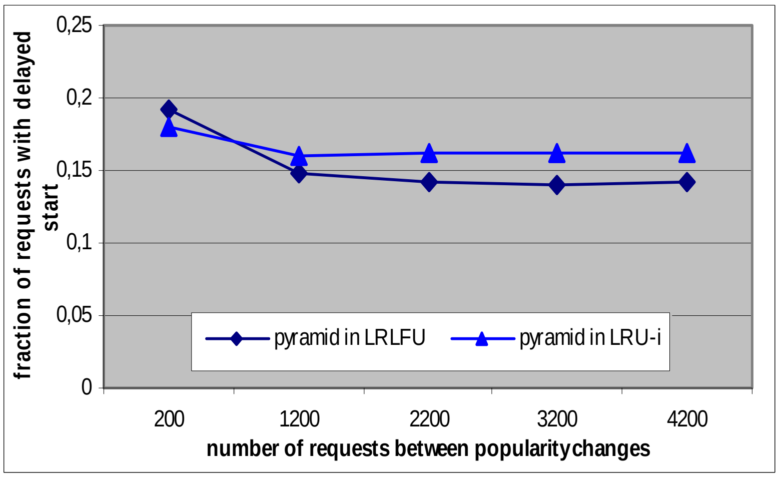


Figure 2.22: The impact of the number of requests between successive popularity changes on delayed starts, for k=150.

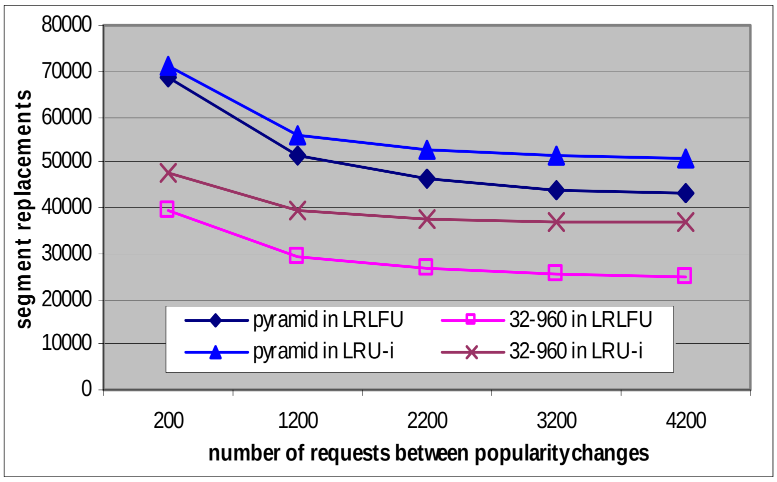


Figure 2.23: The impact of the number of requests between successive popularity changes on segment replacements, for k=150.

From the results presented in this section, we conclude 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.

Chapter 3

Efficient Caching of Video Content to an Architecture of Proxies according to Frequency-Based Cache Management Policies

3.1 Introduction

This chapter focuses on a collaborative environment of more than one proxy servers that serve homogeneous or even heterogeneous client requests for streaming of video files. Under a hierarchical tree topology system of proxies, the prefixes of the videos are stored in small size proxy caches each located very close to the corresponding client community, while larger caches located further away from the client communities are used to cache the latter segments of the videos requested by more than one client communities. Frequency-Based Cache Management Policies are used in order to efficiently and dynamically cache the content of the most popular videos among the various proxies. Our event-driven simulations have shown that the hierarchical tree topology of proxies achieves a much higher byte-hit ratio when using the same overall cache capacity with the simple topology of non-collaborative proxies described in Chapter 2.

3.2 System Description

3.2.1 Network Topology

We consider a hierarchical tree topology system consisting of proxies which are equipped with a small size cache, are located very close to the client communities and are used to serve one client community each (we refer to these proxies as type A proxies) and proxies of type B which are equipped with a much larger size cache than that of type A proxies, are located further away from the client communities and are used to serve more than one client communities each. (We refer to these proxies as type B proxies). In particular, we investigate a binary-ternary tree that consists of two proxies of type B and six proxies of type A to serve six client communities. Figure 3.1 illustrates the system architecture we are considering. Our system consists of (i) a far-distant origin server which stores all the video files, (ii) six proxy servers of type A located each very close to the client communities I, II, III, IV, V, VI, which cache the prefixes of the most popular video files, (iii) two proxy servers of type B, each of which caches the later segments of the most popular videos requested by the three client communities and (iv) six client communities. We assume that there exist high speed dedicated or shared lines connecting type A proxies to the corresponding type B proxy. User requests for videos are directed to the corresponding type A proxy. If the proxy has the prefix of the video, it transmits it to the client, while it contacts the corresponding type B proxy and the content server for the rest of the file, otherwise, under a collaborative scenario it seeks the prefix from the other two type A proxies connected to the same type B proxy and contacts the corresponding type B proxy and the content server for the rest of the file. If the video is popular enough, the type B 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-proxies path to be reduced, the most popular videos must be cached in the proxies. It is assumed that the bandwidth between the proxies and the clients is sufficient to support video streaming with negligible latency, while the network latency between the content server and the proxies is quite significant. Therefore, the idea is to cache enough initial segments of the most popular videos to mask the latency between the content and the proxy servers.

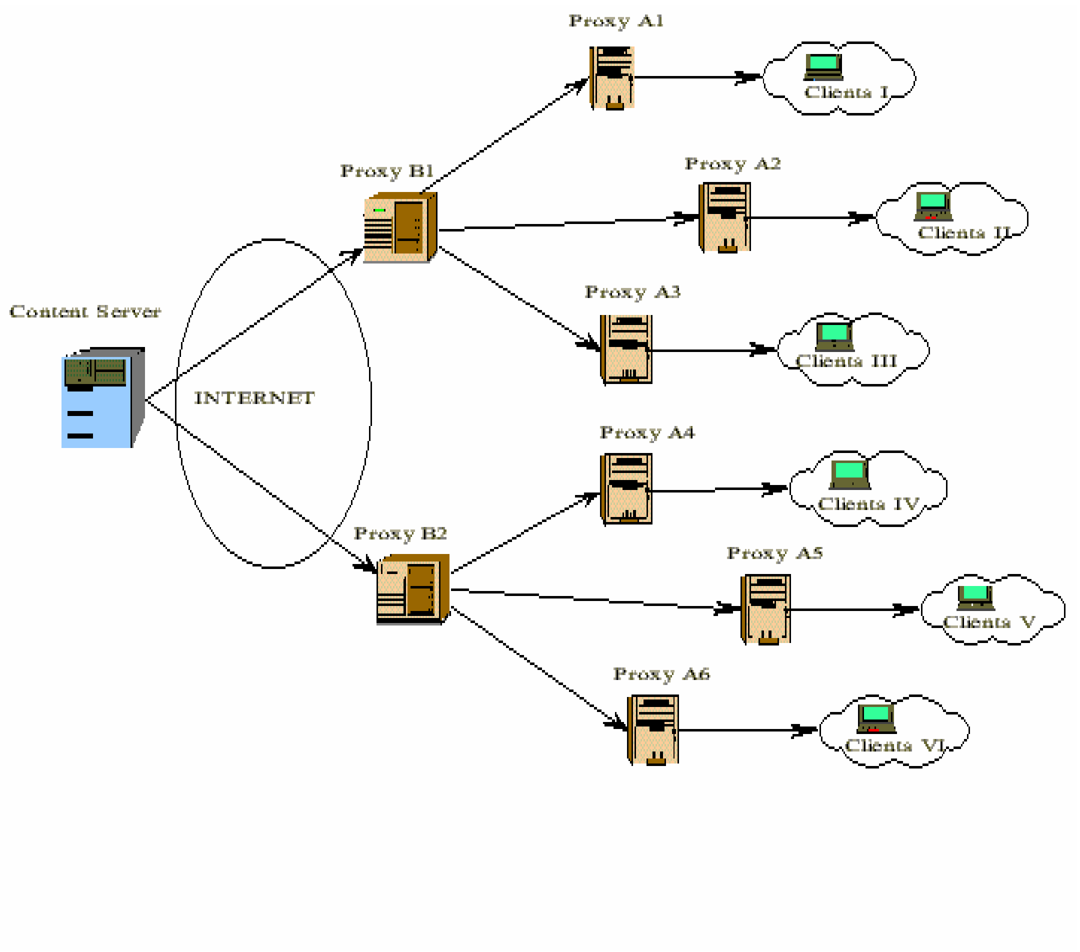


Figure 3.1: Hierarchical tree topology system of proxies for video streaming

3.2.2 Segmentation of video files

We use the same segmentation schemes as in Chapter 2. For details refer to section 2.2.2.

3.2.3 Cache management policies

As already mentioned in the introduction, we apply to different proxies the cache management policies we used in Chapter 2 ([17]) where we studied the case of a single proxy cache. As in Chapter 2, we compare our cache management policies with these of [16] applied to our tree topology system of proxies. For clarity, we describe here these policies and explain how they are implemented at the different type proxies. Every time a client makes a request for a video, the corresponding proxies of types A and B implement their cache management policies independently. When a media object that is not in the cache of the type A proxy is requested, its initial Bp blocks are always eligible for caching in the type A proxy and form the cached prefix of the video. Only the later segments of the video will be considered for caching in the type B proxy. In this way, there is not an overlapping between the contents of type A and B proxies. Once a video prefix is cached in the type A proxy, information is maintained for this object in the type A proxy, such as the object ID, the timestamp of the last request to this object T′ (the timestamp records the last time the object has been requested), and the number of the requests for this object, RF. Similarly, the same information for a video is kept in the type B proxy when a request is made for that video. The type A proxy does not know the information kept in the corresponding type B proxy and neither the type B proxy knows the information kept in the corresponding type A proxies. This information is updated at every request of the particular video.

3.2.3.1 Replacement policy in type A proxy

Bp blocks are cached as an entity in the type A proxy and can only be replaced by other initial Bp blocks of another video. In [16], in order to find space to cache the Bp blocks of a video, a simple LRU policy is applied to all the videos that are not currently played. In contrast, we consider that every prefix in the type A proxy 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 the type A proxy 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 the type A proxy, all the information maintained for this video in the proxy is deleted. However, some later segments of it may be still cached in the corresponding type B proxy and so information is kept for it in the latter proxy. The prefix of the requested video is then cached in the type A proxy and its later segments are considered for caching in the corresponding type B proxy. Notice that by applying this caching value to each prefix, the video that is ejected from the type A cache is not the least recently used but the least valued according to how frequently and how recently it was requested.

3.2.3.2 Replacement and admission policies in type B proxy

Once a video is requested, the type B proxy checks to see if it must cache another segment X of that video in its cache. The segments are cached in an order from initial to later segments. If the video that is requested is not already cached in the type B proxy, then information is kept for it so that it can be considered for caching in subsequent requests. This information is updated at each subsequent request for the same video. At every subsequent request, only one segment of the video is considered for caching. The authors in [16] 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 [16], a small number of videos in the type B proxy 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 for segment X to be cached. If not enough segments with smaller caching values than the caching value of X can be found in order to free space in the cache, 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 simplifies the cache management operation. The caching value of every video participating in the cache admission and replacement policy in type B proxy is the same with the one used in type A proxy, given by $\frac{RF}{T-T'}$.

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 the cache 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.

When a video is requested for the first time from the corresponding proxies of types A and B, the type A 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 the type A proxy and requested from the corresponding type B proxy, the type A proxy and the corresponding type B proxy know its RF and timestamp T′, and therefore they can compute its caching value and update it at every subsequent request. If this video is popular enough, it can gradually bring its remaining content inside the type B proxy in subsequent requests, as in every request only one segment is considered for caching, dependent on its caching value. If the video is not popular enough, its Bp blocks will quickly be discarded from the type A proxy and the wrong decision to cache a small part of this video will not significantly affect the performance of the system. Popularity of each video is well captured by RF and timestamp, two metrics that dynamically adjust their values according to the client preferences.

Information for a video cached in type B proxy is maintained as long as there are segments of that video cached in the latter, otherwise is deleted.

3.3 Performance Evaluation

3.3.1 Performance metrics

We use the same performance metrics as in Chapter 2. For details refer to section 2.3.1.

3.3.2 Simulation model

We conducted an event-driven simulation study to evaluate the performance of our tree topology of proxies with the cache management policies that were previously described. In correspondence to [16] and Chapter 2 ([17]), we have used the same system parameters. The Bp initial blocks cached for a video (prefix) were set equal to 32 blocks both in the pyramid and in the fixed segmentation with prefix scheme. The only difference is that in the pyramid segmentation the initial 32 blocks form 6 segments, while in the fixed segmentation with prefix scheme form one segment. However, when they are ejected from a type A proxy they are considered as one segment replacement.

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 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, $x \in \{1,....V\}$ has a popularity given by $p_x = c / x^{1-s}$, where $c = 1 / \sum_{x=1}^{V} 1 / x^{1-s}$ is a normalizing 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 behaviour of the caches when popularity changes occur. In particular, it is assumed as in [16] that the popularity distribution changes every R requests. 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 modelled 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.

In Chapter 2, we have used 400,000 blocks as the default size for the single proxy cache assumed there. If one proxy of 400,000 blocks served every client community in figure 30, then we would need a total of 6*400,000= 2,400,000 blocks. However, if from these 2,400,000 blocks we use 40,000 blocks for each type A proxy as we used in Chapter 2 for the part A of the cache which is used to cache the video prefixes, then 2,160,000 blocks remain which are split to 1,080,000 blocks for each type B proxy. These large caches are each used to accommodate the requests from three client communities. The benefits of such a cache distribution will be demonstrated in the following sections. All the default system parameter values are shown in Table 2.1.

Notation	Definition (Default values)
Bp	Initial blocks cached for a video (prefix=32 blocks)
λ	Mean request inter-arrival time for each client community (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_A	Cache capacity of proxy A (40,000 blocks)
C_B	Cache capacity of proxy B (1,080,000 blocks)
Runs	Number of requests in a simulation run (300,000 requests)

Table 3.1: System parameters and default values

3.3.3 Simulation scenarios

1) Homogeneous scenario:

Under this scenario, all the client communities have the same preferences for the video files. Video popularities obey the same Zipf-like distribution with the same parameters s and V.

2) **Heterogeneous scenario with different client preferences:**

Under this scenario, each client community has different preferences for the video files following a different Zipf-like distribution. More specifically, the Zipf-like distributions of the different client communities are correlated with the same parameters s and V. The correlation between the distributions is modelled exactly as it was described in section 3.3.2. This time, however we use the parameter H, which can also take any integer value between 1 and V. H represents the maximum position in popularity a video title may shift from one distribution to the next. For H=V, the distributions are uncorrelated, that is, the client communities have totally different preferences of video files.

3) **Heterogeneous scenario with different skew in client preferences:**

In this scenario, we examine the case where the preferences of videos of various client communities follow Zipf-like distributions that differ only in the parameter s (skew of the distribution). Under the two heterogeneous scenarios, requests for a particular video are made with different probability to each type A proxy. We investigate how this behaviour affects the performance of type B proxies, which serve the different client preferences of three client communities.

4) **Non-Cooperative scenario:**

Under this scenario, if the prefix of a requested video is not found in the corresponding type A proxy, then the video is requested from the corresponding type B proxy and the server and there is a delayed start for this video. If there are any cached blocks of that video in the corresponding type B proxy, these are sent to the client and the server sends only the prefix and the remaining non-cached blocks.

5) Cooperative scenario:

In this scenario, we consider that the proxies A1, A2 and A3 are linked with high-speed connections, as shown in Figure 3.2. If, for example, a client from client community I requests a video from proxy A1 and proxy A1 does not have the prefix of this video, it asks proxies A2 and A3 for it. If it finds it there, it caches it and then sends it to the client. Because of the high-speed connections considered we assume that the client does not experience a delayed start in such case.

In all the aforementioned scenarios, the change in the popularity distribution of each client community is still applied, as explained in section 3.3.2, every R requests to each type A proxy.

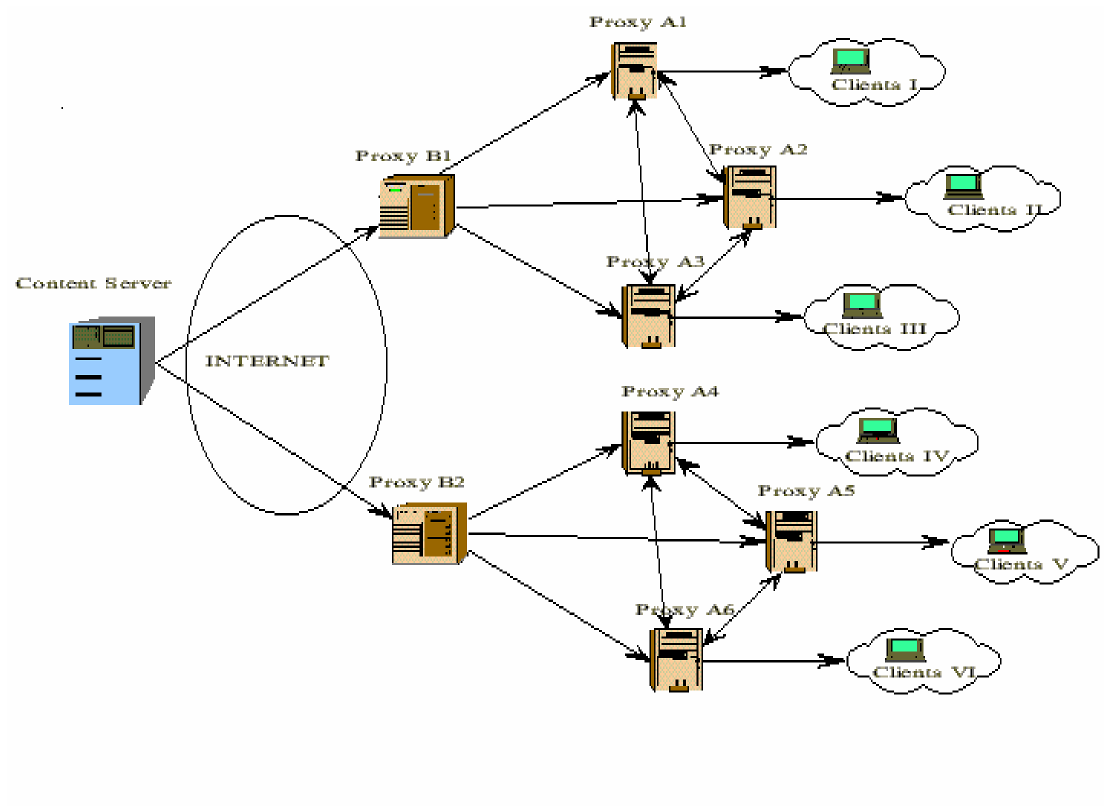


Figure 3.2: Hierarchical tree topology system of collaborative proxies for video streaming.

3.4 Simulation results

3.4.1 Performance of Homogeneous Scenarios

In order to demonstrate the superiority of the tree topology system of proxies against the simple topology of Chapter 2 ([17]) we simulated the requests from 3 client communities using our LRLFU scheme and the LRU-i scheme of Chapter 2 when pyramid segmentation is used. We assume that all three communities have the same preferences for video files, that is, we consider a homogeneous scenario. According to Chapter 2, the three client communities will each be served by a proxy C. Each of these proxies has a cache of 400,000 blocks (Figure 3.3). As explained in section 3.3.2, for the tree topology we use a type B proxy of 1,080,000 blocks and 3 type A proxies of 40,000 blocks each, in order to have the same total cache capacity in the two topologies. As can be seen from the results of Chapter 2, the byte-hit ratio that is achieved by the simple topology for every proxy is around 0.51 (see Figure 2.3 for cache size 400,000 blocks) while the byte-hit ratio for each type A proxy for the tree topology is around 0.69 (see Figure 3.4 for cache size 1,080,000 blocks), that is a 35% improvement in byte-hit ratio. The percentage of delayed starts is the same in the two schemes as it depends only on the capacity used to store the prefixes, which is the same in the two schemes.

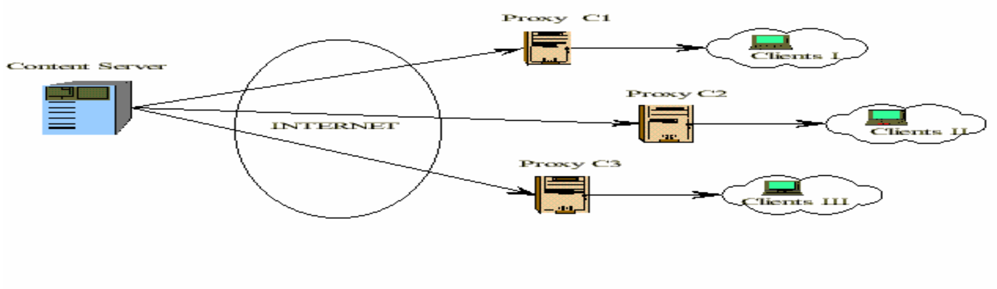


Figure 3.3: A simple topology of proxies for video streaming

Moreover, there is a 31% reduction in the total number of segment replacements when using the tree topology instead of the simple one (compare Figures 2.11 and 3.7 for the Homogeneous scenario (H=0) – note that the overall segment replacements for the simple topology correspond to three times the number of segment replacements for cache size 400,000 blocks, to take into account all three proxies.). Therefore, we conclude that the overall cache management overhead in the tree topology is less than in the simple topology.

The byte-hit ratios achieved by different cache capacities of type B proxy can be seen in Figure 3.4. From the results shown in Figure 3.4 we can see that even with a small cache size of type B proxy, e.g., equal to 380,000 blocks resulting in a total cache capacity of 500,000 blocks in the tree topology, we can achieve the same byte-hit ratio with the one of the simple topology with a total capacity of 1,200,000 blocks (see Figure 2.3 for cache size equal to 400,000 blocks). In other words, the tree topology achieves the same byte hit ratio with the simple topology with a 58% reduction in the overall cache capacity in the system.

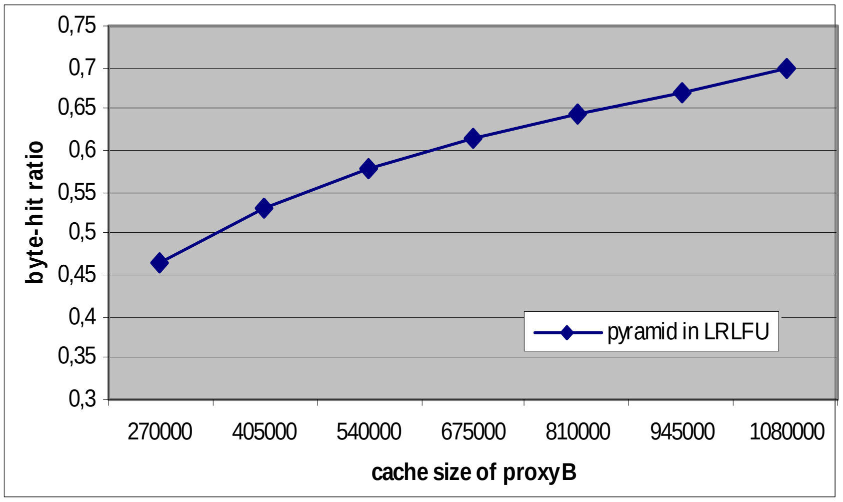


Figure 3.4: Byte-hit ratio versus cache size of proxy B

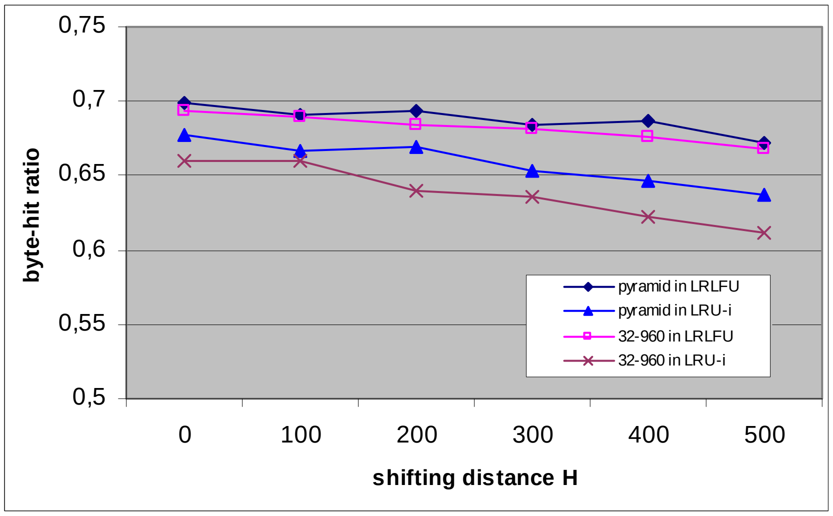
3.4.2 Performance of Heterogeneous Scenarios

In this section, we examine the two heterogeneous scenarios, described in section 3.3.3. First, we simulate the requests from three client communities, which follow different correlated distributions with parameter H. In Figures 3.5(a), and 3.5(b) we can see the impact of rather small and large values of H on byte-hit ratio. In Figures 3.6 and 3.7 we can see the impact of parameter H values on delayed starts and on the total number of segment replacements, respectively.

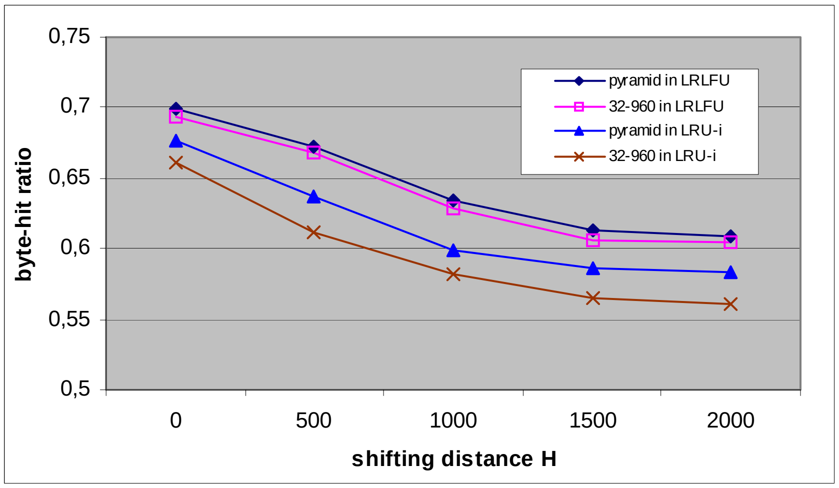
H=0 corresponds to the homogeneous scenario, while H=2000 corresponds to the scenario in which all three client communities have totally different preferences of video files. The important point here is that even for H=2000, the byte-hit ratio achieved from our tree topology is quite high (0.61) and still much larger than the one that could be attained by the simple topology (0.51). The aforementioned results are from the pyramid in LRLFU scheme. Furthermore, from the results in Figure 3.8 we can see that in the heterogeneous scenario (H=2000) even for a smaller cache capacity of 680,000 blocks for type B proxy, that is a total cache capacity of 800,000 blocks, we can achieve the same byte-hit ratio with the one achieved by the simple topology with a total cache capacity of 1,200,000 blocks (see Figure 2.3 for cache size equal to 400,000 blocks). This means that the aggregation of the requests of more than one client communities in type B proxies pays off even when the video popularity distributions are not correlated.

Apart from the pyramid in LRLFU scheme, we can see in figures 3.5, 3.6 and 3.7 the performance of all the schemes that we have examined in Chapter 2. As it can be seen from the results in the aforementioned figures, pyramid in LRLFU and fixed segmentation with prefix scheme in LRLFU perform better that the other schemes exactly as in the topology of Chapter 2. We compared all the four schemes for different values of the various parameters, and concluded that the pyramid and fixed segmentation with prefix scheme in LRLFU outperform the other schemes in any case.

This is attributed to the frequency considerations concerned in the caching values. In this way the popularity of every video is better captured and the proxies succeed to cache only the most popular videos.



(a)



(b)

Figure 3.5: The impact of the shifting distance on byte-hit ratio

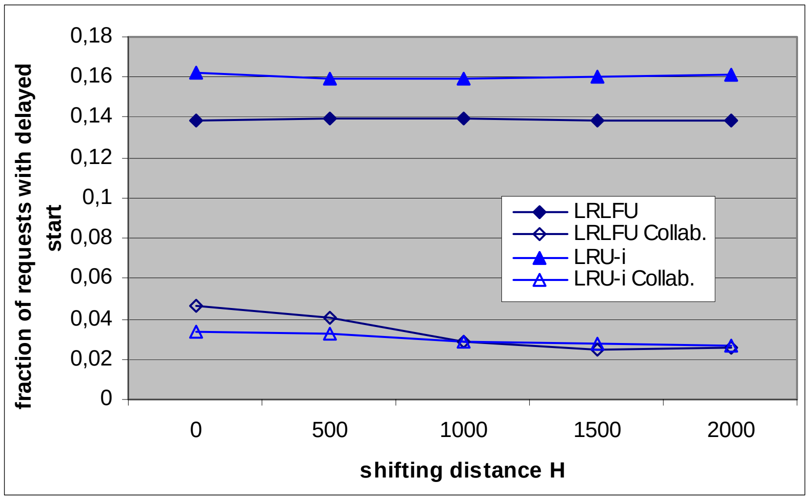


Figure 3.6: The impact of the shifting distance on fraction of requests with delayed starts

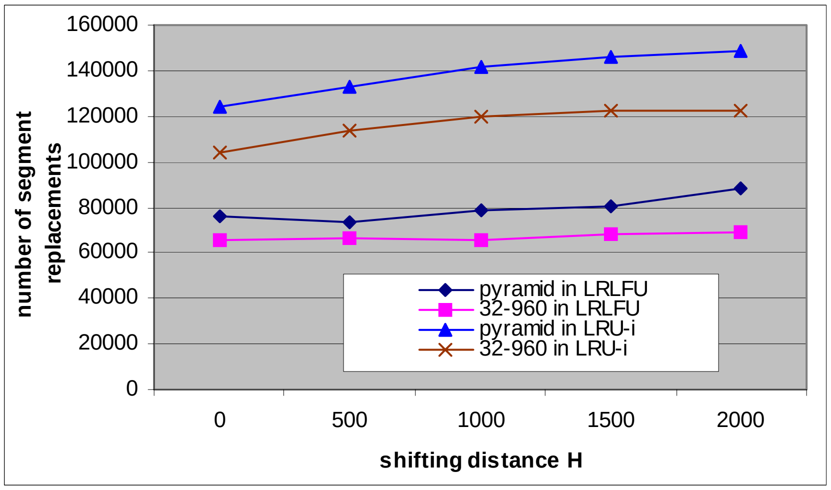


Figure 3.7: The impact of the shifting distance on segment replacements

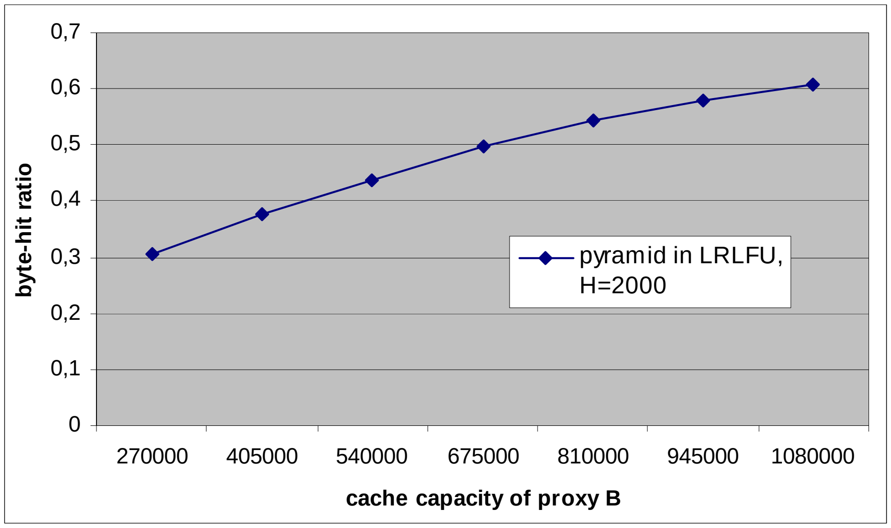


Figure 3.8: Byte-hit ratio versus cache size of proxy B (H=2000)

Furthermore, the segment replacements are fewer when the LRLFU scheme is used. These cache management policies are more accurate, because of the frequency considerations based on which they cache segments of a video, and a segment is replaced only when is necessary. Fewer segment replacements contribute to less CPU overhead. The fixed segmentation with prefix scheme achieves much fewer segment replacements as larger portions of videos are replaced in one action because of the large segment size of 960 blocks that is used in this scheme. What's more, the overall segment replacements in the tree topology are much less than the segment replacements in the simple topology even for totally different client preferences (H=2000). Actually there is a 20% reduction in the overall number of segment replacements for H=2000 when using the tree topology than the simple topology (compare Figures 2.11 and 3.7)

Figure 3.6 reveals the great improvement on the percentage of delayed starts experienced by the clients when the cooperative scenario is adopted. For H =0, there is a 66% reduction in the percentage of delayed starts while for H=2000 the reduction reaches 82%. For the LRU-i scenario this reduction is even greater ranging from 79% when H=0 to 83% when H=2000.This is attributed to the fact that under this scenario a simple LRU algorithm is applied to the videos in type A proxy which fails to capture the actual popularities of the videos, so type A proxies do not exactly cache the most popular but the most recently used videos. That is why proxies A1, A2 and A3 cache different videos even for the Homogeneous scenario and so under a cooperative scenario they can significantly contribute to a lower percentage of delayed starts experienced by the clients. On the other hand, when our algorithm is used in type A proxies, the popularity of each video is better captured because 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.

Next we examine the performance of the hierarchical tree topology for different degrees of skew in video popularity when (i) the homogeneous scenario and (ii) the first heterogeneous scenario for H=2000 is used, and compare the results with those obtained from the simple topology of Chapter 2, (see Figure 3.9).

From the results in Figure 3.9, we can see that the byte-hit ratio deteriorates as the Zipf parameter s increases, i.e., client preferences are dispersed among a plethora of videos. However, the hierarchical tree topology provides better results than the simple topology even when all three client communities have totally different preferences of video files (H=2000). It is remarkable that the less skewed the popularity distribution is (large values of s), the greater is the improvement in byte-hit ratio when using the hierarchical tree topology instead of the simple one. This is indicative of the fact that the hierarchical tree topology performs better even for the worst cases of less skewed popularity distributions and totally different client preferences among the three communities.

Similarly by examining the performance of the hierarchical tree topology for different values of the mean number of blocks per video M, or the number of distinct video titles V we will conclude that the hierarchical tree topology outperforms the simple one in any case.

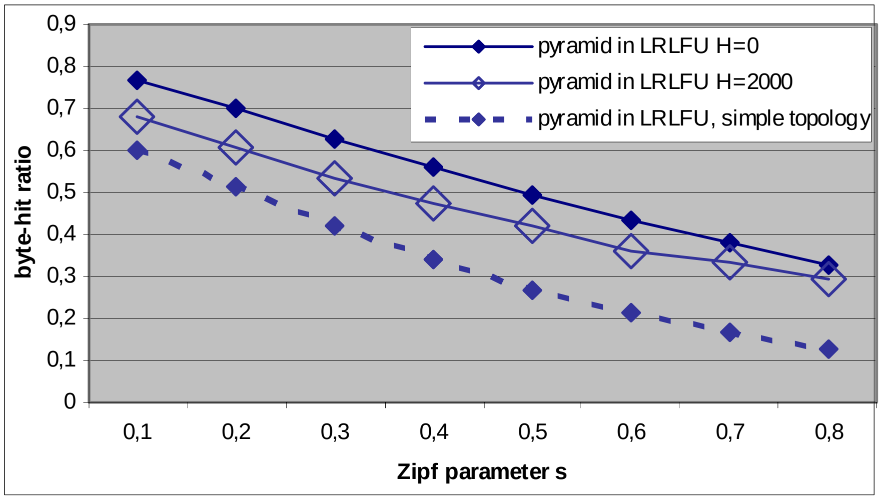


Figure 3.9: Byte-hit ratio versus Zipf parameter s

Finally, we consider the second heterogeneous scenario described in section 3.3.3. We would like to investigate the efficiency of the tree topology when the preferences of videos of the three client communities follow different distributions with a different degree of skew. In Figure 3.10, we can see (i) the byte-hit ratio achieved by the combination of proxy A1-proxy B when the distribution for the preferences of videos from client community I has a degree of skew s=0.2, (ii) the byte-hit ratio achieved by the combination of proxy A2-proxy B when the distribution for the preferences of videos from client community II has a degree of skew s=0.4 and (iii) the byte-hit ratio achieved by the combination of proxy A3-proxy B when the distribution for the preferences of videos from client community III has a degree of skew s=0.6, when the pyramid in LRLFU scheme is used. The three byte-hit ratios are shown for different values of H. It is remarkable that the byte hit ratio for the client community I is not negatively affected by the less skewed distributions of the other client communities, but it is almost the same with the one when the other distributions had the same degree of skew (see Figure 3.5(b)).

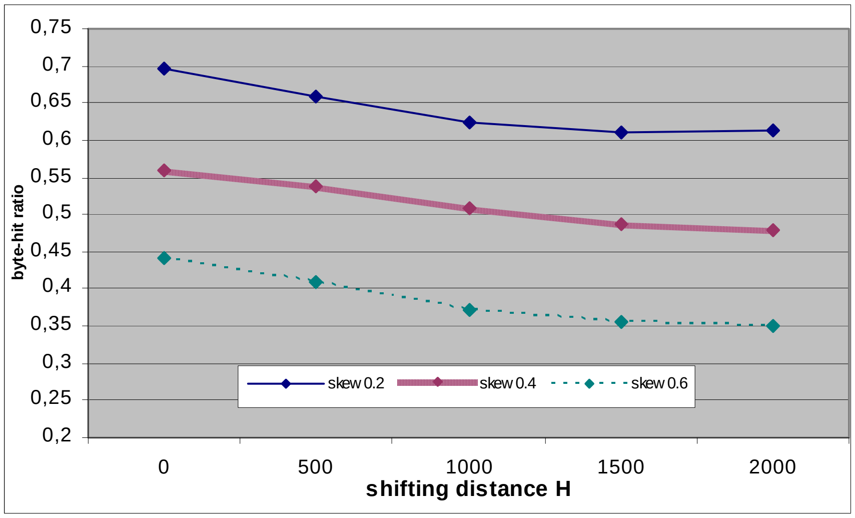


Figure 3.10: Byte-hit ratio for distributions with different degree of skew.

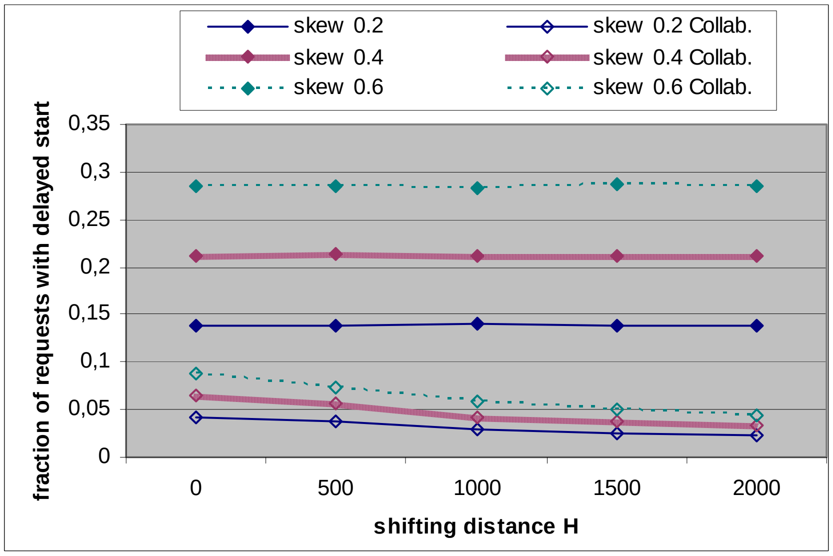


Figure 3.11: Fraction of requests with delayed start for distributions with different degree of skew.

The percentage of delayed starts for the three communities is shown in Figure 3.11 for the cooperative and non-cooperative scenarios. The cooperative scenario significantly reduces the percentage of delayed starts experienced by all three client communities.

We were interested in examining whether the byte-hit ratio achieved by the client community I is not affected by the less skewed distributions followed by the other two client communities because of the large cache capacity of proxy B.

So, as a following step we viewed the scenario where client communities II and III follow an even less skewed video preference distribution with degree of skew equal to 0.8, that is an almost uniform distribution, and investigated how this may affect the byte-hit ratio achieved by the client community I which follows a skewed distribution with degree of skew equal to 0.2 for different cache capacities of proxy B. We can see from the results in Figure 3.12 that there is only a slight deterioration in the byte-hit ratio when it is compared with the homogeneous scenario even for small cache sizes of type B proxy.

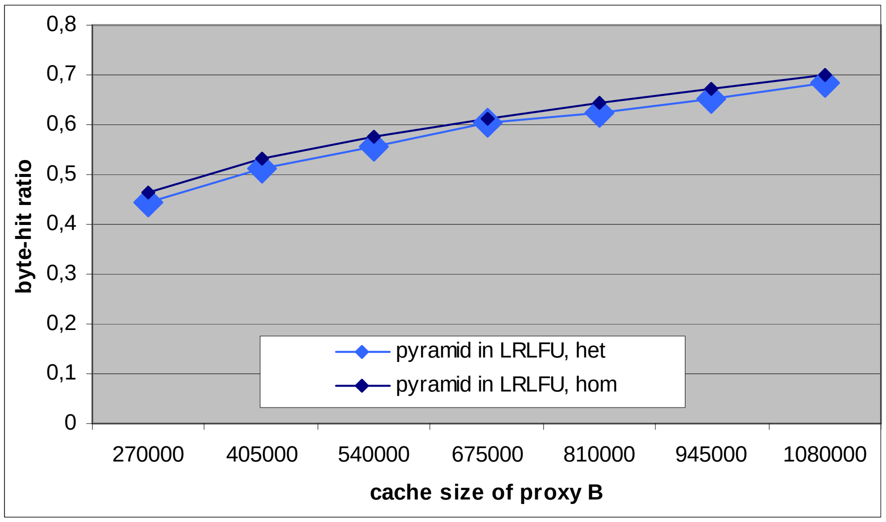


Figure 3.12: Byte-hit ratio versus cache size of type B proxy for homogeneous and heterogeneous scenarios.

We infer that the byte-hit ratio achieved by the client community I is not significantly affected by the other two because the videos that are requested by the client communities II and III cannot stay for long in the cache, as they are requested with almost the same probability (s=0.8). Therefore their popularities are not so strong, their RF values are small in the cache and so they are quickly ejected from the cache in favour of the most popular videos requested by the client community I.

Similarly, the byte-hit ratios of client communities II and III are not significantly affected by the client community I. The byte hit ratios of the client communities whose preferences follow less skewed distributions are very small even in the homogeneous scenario, because the requested videos have a small popularity and their blocks cannot stay long in the cache. Actually, under the aforementioned heterogeneous scenario the byte-hit ratio of client communities II and III is slightly improved compared to the homogeneous scenario because there are some videos, which are the most popular in the client community I, that have enough of their blocks inside the cache and can improve the byte-hit ratio of client communities II and III when they are requested, in contrast to all the other videos whose blocks enter and leave the cache very quickly.

Chapter 4

Conclusions

4.1 Overview of Work

In the first part of this Thesis (Chapter 2), 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.

In the second part of the Thesis (Chapter 3), we examined the performance of a system of collaborative proxies used for video streaming. More specifically, we considered a hierarchical tree topology of proxies where small size proxies (type A) are located very close to the corresponding client communities in order to cache the prefixes of the most popular videos of the particular communities, while larger caches (type B) located further away from the communities are used to cache the later segments of the videos requested by more than one client communities.

4.2 Main Conclusions and Research Contribution

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 [16], improve the byte-hit ratio, significantly reduce the fraction of requests with delayed starts and the number of segment replacements needed during a simulation run, thus yielding a lower CPU overhead in the proxy. The performance of our scheme deteriorates only for large and frequent changes in the popularity distribution, cases which are not expected to be common in practice. In reality, it is more common that there will be either a large change every long time intervals or a progressive (small and frequent) change in the popularity distribution.

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 and also requires less segment replacements. 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 [16] 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.

Furthermore, we simulated the requests from three client communities when using the hierarchical tree topology with three type A proxies of 40,000 blocks each and one type B proxy and the simple topology with three caches of 400,000 blocks. We concluded that for homogeneous preferences of all three client communities and for the same overall cache capacity used from the two topologies, the hierarchical tree topology achieves a 35% improvement on byte-hit ratio with 31% less segment replacements, when compared with the simple topology. Moreover, even for totally different client preferences of the three communities the hierarchical tree topology provides a 20% improvement on byte hit ratio with 20% less segment replacements.

In general, the hierarchical tree topology outperforms the simple one for every different value in the system parameters we examined. Furthermore, the byte-hit ratio for one client community is not significantly affected by the preferences of the other client communities even if they follow less skewed distributions and even for small cache capacity of type B proxies. Moreover, under the cooperative scenario between the type A proxies, the hierarchical tree topology achieves a 66% reduction on the percentage of delayed starts for homogeneous client requests and a 82% reduction for totally different client preferences.

As far as cache management policies are concerned, pyramid and fixed segmentation with prefix schemes in our cache management policies (we refer to these policies as the LRLFU scheme because of the combination of an LRU and an LFU replacement policy that we attain) outperform pyramid and fixed segmentation with prefix schemes in the scheme of [16] (we refer to this scheme as the LRU-i scheme because of the contribution of the number of segment i in an LRU policy), in the hierarchical tree topology as in the simple topology.

Finally, the hierarchical tree topology that we examined is a much more efficient Caching Architecture than the simple topology, as it drastically improves the byte-hit ratio with less cache management overhead (much fewer segment replacements) and significantly reduces the percentage of delayed starts under a cooperative scenario, when using the same overall cache capacity with the simple topology. The extra cost of one proxy for every three proxies to accommodate three client communities is low since many caches use public-domain software that runs on relatively inexpensive servers. Finally, the availability of high speed connections interconnecting proxies is commonplace nowadays and the associated communications costs are low and further falling.

4.3 Ideas for Future Work

An interesting feature of the proposed caching architecture would be its capability to support VCR-like operations. Therefore, the work in this Thesis could be extended to support VCR-like operations, e.g. pause, fast forward (FF), rewind (REW), etc. One way to do this is via an extra mechanism in the cache that would be used to cache only those frames that correspond to a video scene change and contain significant information. For instance, these can be the I frames in an MPEG encoded video. So, when a FF or REW operation is performed, these frames will be playbacked, giving the user the sense that the program speeds up.

The above mechanism could work in cooperation with the cache management policies proposed in the first part of our Thesis to cache video segments in the part B of the cache. In this way there would be two caching mechanisms in Cache B, the one that is used to cache the segments of the video files in order to support video streaming and the one that will be used to cache additional I frames of a video in order to support VCR operations.

Another idea would be to use a dedicated portion of the cache, for example the Cache A that we used in this work to cache the video prefixes, to cache the hotspots of the videos instead, and we could use the Cache B to cache all the other segments. A hotspot in a video stream represents a salient object that is semantically related to other media, such as image, audio, or text that provides details about the video object [18]. Implicit with this idea is the assumption that the users do not submit continuous playback requests, but instead preview a video file before deciding whether to view it in its entirety or quickly jump to some specific portion of it associated with a specific hotspot. In the latter case, certain hotspots of the videos and not their prefixes are more important.

Finally, the cache management policies introduced and evaluated in this work could be made network-aware by accounting not only for the popularity of the media objects but also the bit rate requirements of these objects as well as network conditions such as the available bandwidth between servers, clients and caching proxies. Therefore, the caching value of the videos should be properly modified to take into consideration all the above issues.

References

[1] S. Ramesh, I. Rhee, and K. Guo, “Multicast with cache (mcache): An adaptive zero-delay video-on-demand service”, in Proc. of the 2001 IEEE INFOCOM Conf., Anchorage, Alaska, April 2001, pages 85--94.

[2] D. Eager, M. Ferris, and M. Vermon, “Optimized regional caching for on-demand data delivery”, in Proc. of Multimedia Computing and Networking (MMCN ’99), January 1999, pages 301--316.

[3] O. Verscheure, C. Venkatramani, P. Frossard, and L.Amini, “Joint server scheduling and proxy caching for video delivery”, in Proc. of the 6th International Workshop on Web Caching and Content Distribution, June 2001.

[4] Y. Guo and 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.

[5] D. L. Eager, M. C. Ferris, and M. K. Vernon, “Optimized Caching in Systems with Heterogeneous Client Populations”, Performance Evaluation, Special Issue on Internet Performance Modeling, 42(2/3), September 2000, pages 163--185.

[6] B. Wang, S. Sen, M. Adler, and 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.

[7] C. Venkatramani, O. Verscheure, P. Frossard, and K. Lee, “Optimal Proxy Management for Multimedia Streaming in - Content Distribution Networks”, in Proc. of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV’02), Miami, Florida, USA, May 12-14, 2002.

[8] J. Liu, X. Chu, and J. Xu, “Proxy Cache Management for Fine-Grained Scalable Video Streaming”, in Proc. of the 2004 IEEE INFOCOM Conf., Hong Kong, Mar. 2004.

[9] R. Rejaie and 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.

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

[11] R. Rejaie, M. Handley, and D. Estrin, “Quality adaptation for congestion controlled playback video over the internet”, in Proc. of the 1999 ACM SIGCOMM, Cambridge, MA, pages 189—200.

[12] Elias Balafoutis, Antonis Panagakis, Nikolaos Laoutaris, and Ioannis Stavrakakis, “The impact of replacement granularity on video caching”, in Proc. of the 2002 IFIP Networking Conf., Pisa, Italy, pages 214--225.

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

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

[15] R. Tewari, H. M. Vin, A. Dan, and D. Sitaram, “Resource-based caching for Web servers,” in Proc. SPIE/ACM Conf. on Multimedia Computing and Networking (MMCN'98), San Jose, CA, Jan. 1998.

[16] K.-L. Wu, P. S. Yu, and J. L. Wolf, “Segment-based proxy caching of multimedia streams”, in Proc. of the 2001 International WWW Conference, Hong Kong, pages 36--44.

[17] A.Satsiou, M.Paterakis, “Impact of Frequency-Based Cache Management Policies on the Performance of Segment Based Video Caching Proxies”, in Proc. of the 2004 IFIP Networking Conference, Athens, Greece, pages 1120--1131.

[18] K.-L. Wu, P. S. Yu, and J. L. Wolf, “Segmentation of Multimedia Streams for proxy caching” , IEEE Trans. on Multimedia, 6(5), Oct. 2004, pages 770--780

[19] H. Fahmi, M. Latif, S. Sedigh-Ali, A. Ghafoor, P. Liu, and L. Hsu, “Proxy servers for scalable interactive video support”, IEEE Computer, 43(9), Sep. 2001, pages 54--60.

[20] S. Paknicar, M. Kankanhalli, K. R. Ramakrishnan, S.H Srinivasan, and L. H. Ngoh, , “A caching and streaming framework for multimedia”, in Proc. of ACM Multimedia 2000, October 2000, CA, pages 13-20.

[21] S. Acharya and B. C. Smith, “Middleman: A video caching proxy server”, in Proc. of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV’00), June 2000.

[22] Y. Chae, K. Guo, M. M. Buddhikot, S. Suri, and E. W. Zegura, “Silo, rainbow, and caching token: Schemes for scalable, fault tolerant stream caching”, IEEE J. Select. Areas in Comm., 20(7), Sep. 2002, pages 1328--1344.

[23] J. Liu, J. Xu, “Proxy caching for Media Streaming over the Internet”, IEEE Communications, 42(8), August 2004, pages 88--94.