MiddleMan: A Video Caching Proxy Server

Soam Acharya
sacharya@inktomi.com
Inktomi Corporation
Foster City, CA 94404

Brian Smith
bsmith@cs.cornell.edu
Cornell University
Ithaca, NY 14853

Abstract

This paper describes MiddleMan, a collection of cooperating proxy servers connected by a local area network (LAN). MiddleMan differs from majority of existing proxy research in that it concentrates exclusively on video. Other approaches are optimized for HTML documents and images.

MiddleMan offers several advantages. By caching videos near clients, MiddleMan reduces start-up delays and the possibility of adverse Internet conditions disrupting video playback. Additionally, MiddleMan reduces server load by intercepting a large fraction of server accesses and can easily scale to a large number of users and web video content. It can also be extended to provide other services such as transcoding.

1. Introduction

The use caching to improve the performance of clients viewing web documents is well studied. Systems that cache HTML and images include browsers, such as Internet Explorer [25] and Netscape Communicator [26], and collections of cooperative proxies, such as Harvest [9] or Squid [24]. These systems do not normally cache videos, because video files are usually much larger than other web documents and hence would fill up the cache quickly.

This paper describes MiddleMan, a cooperative caching video server. MiddleMan is a collection of proxy servers that, as an aggregate, cache video files within a well-connected network (e.g., a campus network or LAN). By cooperatively caching video files, MiddleMan can have a large aggregate cache while placing minimal load on each participating client. For instance, if each of 500 clients on a college campus participated in a MiddleMan cluster, and each allocated 100 MB of their local disk for the storage of cached videos, the total cache size would be 50 GB, enough for caching a significant number of video files. But since each client rarely views more than one video file at a time, the typical load on any one client would be less than that required to service one video file (about 1 Mbit/sec).

The characteristics of videos stored on the Internet today suggest that caching would be effective. Such features include high bandwidth requirements and the fact that videos rarely change. Our survey of video data on the web [2] shows that sustained bandwidths of approximately 1 Mbps are required in order to stream most MPEG, AVI, and QuickTime video files stored on the web today. These bandwidth requirements make video files susceptible to Internet brownouts. However, our subsequent investigation of video access patterns on the web [3] provided the key insight for our approach: requests to a video server tend to exhibit locality of reference. Some videos are much more popular than others. Hence, it is possible to exploit caching techniques that reduce redundant video accesses to the server while simultaneously mitigating the unreliability of the Internet and improving access latency.

In brief, the architecture we propose, called MiddleMan, consists of a collection of caching video proxy servers that are organized by *coordinators*. A coordinator is a process that keeps track of the files hosted by each proxy and redirect requests accordingly. The coordinator also uses the proxies to manage the copied video files stored at each machine. If there is no free space left in the system, it is the coordinator that decides which files to eliminate in order to make room.

MiddleMan offers a potentially rich set of benefits in dealing with VOW (Video on the Web) problems. Our evaluations show that it achieves high cache hit rates with a relatively small cache size. By caching videos relatively close to the clients and ensuring a large number of video requests are satisfied locally, it reduces overall start-up delays and the possibility of adverse Internet conditions disrupting video playback. From the point of view of the server, MiddleMan dramatically reduces load by intercepting a large number of server accesses. Hence the net effect of MiddleMan is to increase the effective bandwidth of the video delivery system allowing more clients to be serviced at any given time.

Several issues must be addressed before MiddleMan can be built and deployed. This paper describes these problems and our solutions. Our contributions include:

- *Summary of intrinsic VOW file characteristics and overview of their browsing patterns.* Not much is known about either and, hence, we carried out two studies [2,3] which yielded several valuable insights into video files on the web and user access behavior. In this paper, we summarize relevant findings that contributed heavily to the design of MiddleMan.

- *MiddleMan Architecture*. We present the MiddleMan architecture and illustrate how the insights from our studies affected the design.
- *Cache Analysis*. Via trace driven simulations, we inspect the effects of various caching algorithms on MiddleMan performance.
- *Load Balancing Analysis.* Since MiddleMan is a distributed architecture, we examine the load balancing properties of various caching algorithms. The design achieves both high hit rates and excellent proxy load distribution.

The remainder of this paper is structured as follows. Section 2 summarizes the relevant insights from our previous research of video on the web. Section 3 describes the architecture of MiddleMan and section 4 describes the evaluation of the design. We outline related work in section 5 and, finally, present our conclusions and direction for future work in section 5.

2. Trace Observations

A thorough understanding of VOW access patterns and file characteristics is an essential first step prior to architecting MiddleMan. In the absence of any such publicly available work, we conducted two studies. The first, an investigation into the properties of videos on the web [2], involved the downloading and analysis of over 57000 AVI, QuickTime, and MPEG files stored on the Web -- approximately 100 GB of data. Observations from this study relevant to MiddleMan are:

1. *Web video size:* Videos are around 1 Mbytes in size, orders of magnitude larger than HTML documents which are usually sized around 1-2 Kbytes. Playback time is about a minute or less.
2. *Videos are WORMs:* Web files tend to follow the write-once-read-many principle. Once a video has been placed online, chances are that it will stay there. Hence, cache consistency is not a major issue in video caching systems. Examination of periodic snapshots of the VOW server file system used in the second (user browsing behavior) study further corroborated this trend.

For the second study, an analysis of how users access video data, we inspected log file records from the mStar [17] experiment at Lulea University in Sweden. MStar is a hardware/software infrastructure developed by the Center for Distance-spanning Technology at Lulea University for facilitating distance learning and creating a virtual student community. Our initial analysis [3] and subsequent follow-up yielded the following insights that are relevant to the design of MiddleMan:

3. *File size trends:* videos are becoming larger as more network bandwidth becomes available and low bitrate streaming protocols get deployed in video distribution. With a high bandwidth network and H.261 based multicast architecture in place, the median size of files at the Lulea University video server was 96 MBytes. Median duration was 70 minutes.
4. *Video browsing patterns:* users often viewed the initial part of videos in order to determine if they are interested or not. If they like what they see, they continue watching. Otherwise, they stop. We found that about 61% of all playbacks went to completion in our analysis. Most of the remaining 39% stopped very early on in the movie playback.
5. *Temporal Locality:* LRU (Least Recently Used) stack depth analysis [4] of the traces showed that accesses to videos exhibit strong temporal locality. If a video has been accessed recently, chances are that it will be accessed again soon.

Observations 4 and 5 hint that a caching approach could yield rich dividends. The remaining lessons suggest some basic requirements for the initial blueprint of MiddleMan:

- *Flexibility*: it should be able to handle both files with sizes in the megabyte range (observation 1) as well as the hundreds of megabyte range (observation 3).
- *Partial files:* since users are likely to view only part of the video (observation 5), video files need not be stored in the local caching system in their entirety.
- *Cache consistency* is not a significant issue in the system design (observation 2).

In the next section, we illustrate how we utilize these observations and requirements for the design of MiddleMan.

3. The Design of MiddleMan

This section presents the architecture of MiddleMan as well as the concepts and assumptions behind its design. Initially, we outline the overall system and its constituent components. Next, we describe how video files are stored and transferred from within MiddleMan. Finally, we sketch how MiddleMan responds to user requests.

3.1 System Component Configuration

MiddleMan consists of two types of components: proxy servers and coordinators. A typical configuration consists of a single coordinator and a number of proxies running within a local area network. A coordinator keeps track of proxy contents and makes cache replacement decisions for the entire system. A proxy can either interface with users or manage video files.

MiddleMan defines two different types of proxies: *local*

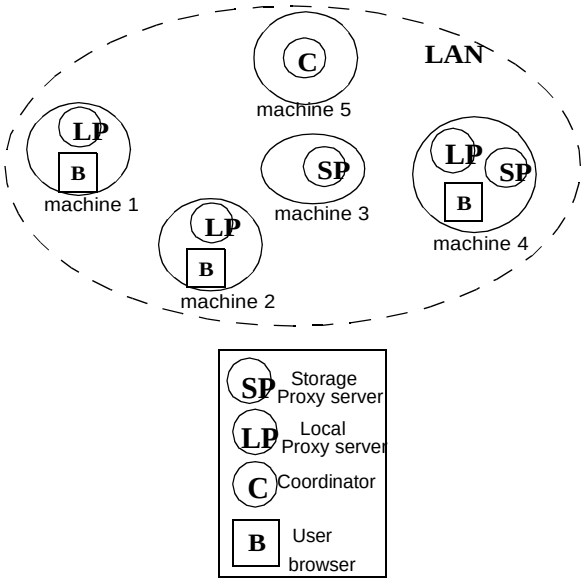

Figure 1: A proxy cluster

and *storage* proxies. Local proxies can run on the same machine as the client and are responsible for answering client requests. They do not store any data and are functionally similar to browser plug-ins. Ideally, a local proxy would run on every user machine in a domain, but this might be hard to deploy. Hence, we assume proxies run on selected machines in the network. Each proxy services a small collection of browser clients that have been configured to forward their requests for video data to that particular proxy. Storage proxies, on the other hand, do not directly service client requests, they just store data. Storage proxies can be located anywhere on the local area network.

A collection of both types of proxies running in a LAN and organized by a single coordinator together form a *proxy cluster*. Figure 1 shows an example configuration. Arranging the system components in the form a of a proxy cluster provides a number of advantages:

- *latency reduction:* communication and data transfers amongst the cluster components can exploit the high bandwidths of the local area network.
- *high aggregate storage space:* by running on user machines, proxies can take advantage of cheap disk space.
- *load reduction:* distributing video files on multiple machines allows the load induced by video requests to be distributed over multiple machines, a better approach than a single central proxy that services all local video requests and becomes a system bottleneck.
- *scalability*: the capacity of the system can be expanded by adding more proxies. Globally, multiple clusters can be linked together by allowing individual coordinators to communicate.

There are two possible disadvantages to this approach. First, since the local proxies consult the coordinator for every request, its central nature might make it a bottleneck. However, the relatively large inter-request arrival times for video, and the fact that the coordinator does not transfer video data, implies this is not a cause for concern. A second problem could be that the coordinator is the central point of failure. In case of a coordinator crash, the system loses state. One possible solution is the *coordinator-cohort* approach where the coordinator maintains a backup coordinator. The coordinator keeps the cohort updated with its current state so that, in the event of a crash, the cohort takes over. Since building fault tolerant central servers is a well studied problem, we did not build such a coordinator. Nothing in the design of MiddleMan prevents making the coordinator fault tolerant, however.

3.2 Video Storage Policies

As we discovered in section 2, users are far more likely to view the opening of a movie than play it back until its end. Hence, unlike HTML documents, an entire video document does not have to be present in the caching system for a request to be satisfied. Based on this observation, MiddleMan incorporates the concept of *partial video caching*. When the user requests a video in the cache, it is served by sending to them the portion of the video locally present while obtaining the remainder from the main WWW server and transparently passing it on to the client.

In order for the partial video caching to work, video servers and streaming protocols must allow random access. Fortunately, all major streaming protocols and HTTP 1.1 [23] allow this.

MiddleMan fragments cached videos into equal sized file blocks in the storage system, which allows the cached title to be spread across multiple storage proxies. Hence, blocks are the minimum unit of replacement for the MiddleMan cache. Representing video as an ordered sequence of file blocks simplifies the architecture considerably. It provides a convenient mechanism for spreading portions of a single title across multiple proxies, thus allowing for better load balancing, simplification of cache replacement and partial video implementation. If a new title T_1 needs to be brought into the cache, yet the entire system is full, blocks allow MiddleMan to simply eliminate the end portions of an unpopular title T_2 on a block by block basis. Hence, instead of getting rid of T_2 entirely, we can have a portion of it present in case it is requested in the future. Similarly, T_1 grows on a block by block basis, its ultimate size depending on how much of it is played back by the user.

A possible problem with this method of fetching blocks from multiple proxies and combining them into a contiguous stream might be due to *inter-block switching delays*.

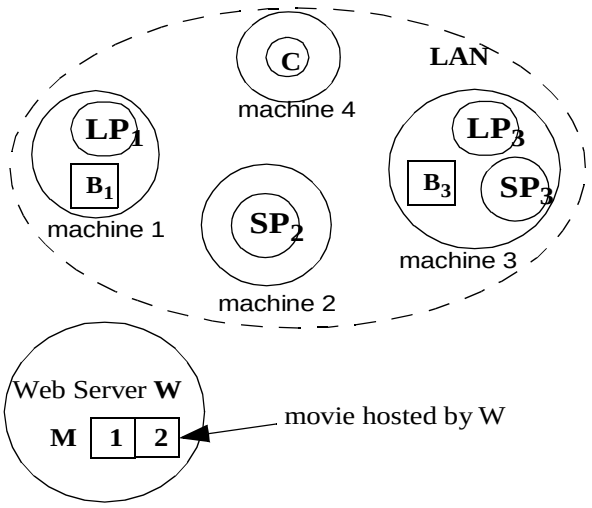

Figure 2: Proxy cluster used in example

This can cause interruptions in the data flow from the proxy to the user. Delays may be caused by the latencies faced by the video data receiving client proxy when it is changing its source from one storage proxy to another. Such switching delays can be eliminated via fetching data at a higher rate and pre-rolling/double buffering against latencies.

3.3 How MiddleMan Responds To Requests

So far we have described the MiddleMan system architecture. In this section, we outline how MiddleMan reacts to common user scenarios including cache miss i.e. an user request for a video not cached by MiddleMan, cache hit, and finally, a request cancellation. We use the example system in figure 2 to illustrate the interaction of individual MiddleMan components. The example system consists of a proxy cluster and *W*, a WWW server external to the LAN. The proxy cluster contains a coordinator *C* plus local proxies LP_1 and LP_3 serving two client browsers B_1 and B_3, respectively. There are also two storage proxies, SP_2 and SP_3, in the system. W hosts a movie **M** that can be logically divided into two file blocks, M_1 and M_2. We now present how the system reacts to the three scenarios (cache miss, cache hit, and request cancellation) in the following subsections.

3.3.1 Cache Miss

The following events occur when B_1 requests the title **M** and the request is intercepted by LP_1.

1. LP_1 simultaneously contacts both *C* and *W*.
2. *C* replies in the negative as **M** is not cached by the system. *W* replies with header information for **M**.
3. From the header information, LP_1 determines **M** characteristics. LP_1 requests the location of a block of space from *C*, sufficient to store M_1.
4. If sufficient space is available, *C* replies with the location of a block. Space might not be available if the cache is full. If so, *C* runs a *cache replacement algorithm* in order to identify an undesirable block and return its location to LP_1. For our example, we choose the location to be SP_2.
5. LP_1 commences receiving M_1 (the first part of **M**) from *W* which it then streams to both B_1 and SP_2.
6. After M_1 has been received in its entirety, LP_1 requests the location of another block from *C* (in order to locally cache M_2).
7. *C* returns SP_3 as the location for the next block. LP_1 continues receiving M_2 (the next part of **M**) from *W* which it then streams to both B_1 and SP_3.

In general, a movie not cached locally by MiddleMan, is brought into the system and stored on a block by block basis, depending on how much of the movie is actually viewed by the user. One possible drawback of this approach might be the costs of the additional proxy-coordinator communication per block transaction. However, the additional overhead is insignificant compared to the cost of transferring the actual videos.

3.3.2 Cache Hit

If **M** is now cached in the system and B_3 requests **M**, the following sequence of events transpire:

1. LP_3 simultaneously contacts both *C* and *W*.
2. *C* replies in the affirmative, returning a pointer to a block housed at SP_2.
3. LP_3 closes the connections with *W*.
4. LP_3 contacts SP_2 and starts streaming M_1 to B_1.
5. After M_1 has been exhausted, LP_1 contacts *C* again to obtain the location of the next block.
6. *C* replies with a pointer to SP_3. LP_3 contacts SP_3 and starts streaming M_2 to B_3.

A possible disadvantage of using a custom protocol for transferring data between local and storage proxy servers is the added complexity as opposed to the more straightforward method of using a distributed file system such as NFS. However, this simplified approach does not scale as it does not allow local proxies to fetch data from storage proxies in proxy clusters located in other organizations and networks.

3.3.3 Request Cancellation

Broken requests are a side-effect of how users browse video - they commence playback of a title and decide to stop if they do not like what they see. Halting the playback results in the browser cancelling its local proxy connection while a cache hit or miss type transaction is taking place.

If the title is being fetched from the local cache during a

cache hit, dealing with a cancellation is relatively simple. The proxy, after detecting the closed browser channel, simply shuts down the other connections associated with the request and notifies the coordinator that it is done with the title.

If the request cancellation occurs during a cache miss, complications may arise depending on whether other users in the system are also currently accessing the same file. Assume a scenario where only a single user B_1 decides to cancel while viewing the second half of **M** (M_2). LP_1 detects this and notifies the coordinator. As no other users are currently accessing this video, the coordinator asks LP_1 to close the remaining connections. All references to M_2 are deleted by the coordinator. M_1, however, is still cached locally.

In a more complicated scenario where both B_1 and B_3 are concurrently interested in **M**, B_1 decides to cancel during the second half of the movie playback. Once again, LP_1 detects the closed connection and notifies the coordinator. However, since B_3 is still interested, the coordinator asks LP_1 to continue fetching the file and save it locally to the storage proxies. One possible disadvantage of this approach is the additional bandwidth expended by LP_1 forwarding thus affecting access on the local machine by B_1. However, since we assume a high speed local network connecting our proxies, this should not be an issue.

4. The Analysis of MiddleMan

In this section, we briefly outline the performance of MiddleMan. Factors affecting this include the cache replacement algorithm used, the number of proxies in the cluster, the size of the file blocks, total cache size, and the request pattern. However, the most critical design choice is the cache replacement policy, since this not only affects the performance of the system but also the balance of load among the proxies. Load balancing is important because we want user machines to participate as proxies in the cache. Users will not agree to this use if the load on their machines grows too high. Hence, we primarily investigate how different cache/load balancing algorithms affect MiddleMan performance.

We selected the simulation approach for evaluating MiddleMan. The simulator we developed, MiddleSim, consists of about 5000 lines of Java code. It is a trace driven discrete-event simulator. For the actual simulation runs, we used the traces from the mStar project [17]. This consisted of accesses to the mMOD video server from three subnets on the Lulea University (*lulea.se*) campus. The requests spanned a period ranging late August 1997 to mid October 1999, more than two years in total. The three subnets consisted of *cdt.luth.se* (1775 accesses in total for the period measured), *campus.luth.se* (2374 requests), and *sm.luth.se* (3190 requests). The requests were to 235 video files on the mMOD web server. Video content ranged from classroom lectures and seminars to traditional movies.

In addition to our message passing assumptions and failure model, we also fixed the file block size of MiddleSim at 1 Mbyte. This value allows MiddleMan to easily store both the relatively small web video files analyzed in [2] and the much larger video recordings analyzed in [3]. Files of the latter type have much lower bitrates than web video (median bitrate of 150 kbps as opposed to 700 kbps for web video). Hence, in the partial storage case, a file block from a video with bitrate of 150 kbps is likely to provide a proxy with a gap of about a minute before the proxy has to contact the coordinator again. Such a long interval reduces the load on the coordinator, especially if it is involved with a number of simultaneous connections.

In the subsequent sections, we first describe the simulator parameters for our experiments, our performance metrics, and finally, the results we obtained.

4.1 Simulator Parameters

A number of MiddleSim parameters can be varied prior to starting a simulation run. However, we concentrated on two: the total proxy cache size and the cache algorithms.

4.1.1 Proxy Cache Size

The aggregate size of all the files stored by the mMOD server is about 25 GBytes. The total size of the proxy caches can be a small fraction of this total. For ease of comparison, we fixed the total number of proxies in the system to be the same as the smallest group of active machines from the three traces, namely 44 from cdt. Ultimately, we selected three proxy cache size configurations: the first allocated 12 Mbytes of cache space to each proxy for a total of 44*12 or 528 Mbytes of global cache storage (about 2.1% of the total size of all the video files). The second configuration allowed 25 Mbytes*44 or a global cache size of 1.07 GBytes (about 4.3% of total file size) and the third configuration provided 50 Mbytes per proxy for a total of 2.14 GBytes or 8.6% of total file size.

4.1.2 Cache Replacement Policies

We investigated a number of cache replacement policies. These included *Least Recently Used* (LRU [20]), *Least Frequently Used* (LFU [20]), *First In First Out* (FIFO [20]), *LRU-k* and *HistLRUpick*. The LRU-k algorithm [15] maintains a history of the previous k accesses to each title in the cache. The *k-distance* of a title at a given time is defined as the difference between the current time and the time at which the k-th access was made to that title. LRU-k chooses to replace the end-most block in the title with the largest k-distance. It resolves ties by picking the title which

has been referenced least recently i.e. running the LRU algorithm on the tied candidates. LRU itself is a special case of LRU-k where k is 1.

The *HistLRUpick* algorithm is based on LRU-k. We developed this approach in order to explicitly integrate load balancing with cache replacement. HistLRUpick runs LRU-2, LRU-3, and LRU-4. Ties are resolved by picking the block that is *managed* by the least loaded proxy. The criteria for choosing the least loaded proxy is based on the *HistLoad* metric for each proxy. Essentially, it is a measure of the load experienced by a proxy in the past hour. The metric empirically combines the number of bytes that have passed through the proxy, peak number of connections, and the peak bandwidth used by the proxy, all evaluated over the past hour. Since the metric calculates the load on a proxy relative to its counterparts, the proxy-specific terms are normalized by the aggregate values computed over the entire system i.e. the total number of bytes that have passed through the system, the peak number of connections and peak bandwidth used by the entire system. When computing the load, preference is given to the recent number of bytes that have passed through the proxy, since ultimately, our goal was to minimize the variations in byte traffic of the individual proxies within a cluster. The instantaneous rates and connection terms are given less precedence but are included, since it is also desirable to detect and prevent sudden fluctuations.

To compare the effectiveness of these algorithms, we implemented an additional approach *Perfect*. This is an ideal cache replacement mechanism [20] that uses knowledge of the future to replace the cache block that will not be used for the longest time. It can be shown that, given a finite cache size, this algorithm is optimal.

4.2 Measuring MiddleMan Performance

We employed MiddleSim to evaluate both the static and dynamic performance of the various MiddleMan configurations. A static measurement indicates overall system behavior after it has completed a simulation run. On the other hand, a dynamic measurement reports on performance during a run, indicating how well the system adjusts to sudden changes in input.

In the remainder of this section, we report the results of testing both the dynamic and static performance of various parameters and configurations of MiddleMan. We first describe the overall caching performance, then select two of the caching algorithms for further analysis of their load balancing capabilities. Our final results concern MiddleMan load balancing scenarios when the number of proxies are reduced in the system, yet total global cache size remains the same.

4.2.1 Overall Cache Performance

We used the *byte hit rate* (BHR) metric to evaluate overall caching performance of various MiddleMan configurations. The BHR of a run is defined as:

BHR = *(total bytes served from the cache)/(total bytes read by all clients)* (EQ 1)

A BHR close to 1 implies good performance since most of the bytes requested by the users are served from the local cache.

Table 1 reports the BHR values for all the algorithms discussed in section 4.1.2 with the exception of LRU-k. Instead of providing the results for LRU-2, LRU-3, and LRU-4, we simply present LRU-3 as being the most representative for LRU-k. The following trends emerge from the investigation:

Table 1: Byte Hit Rates Under Various System Configurations

Trace	*cdt*	*campus*	*sm*
Configuration	*44 machines * 12 Mbytes (2.1%)*		
Perfect	*49.8%*	*52.7%*	*56.9%*
LRU	*43.2%*	*48.7%*	*52.4%*
LFU	*42.6%*	*47.9%*	*52.1%*
FIFO	*41.0%*	*46.7%*	*50.4%*
LRU-3	*44.7%*	*48.8%*	*53.5%*
histLRUpick	*45.0%*	*49.3%*	*53.4%*
Configuration	*44 machines * 25 Mbytes (4.3%)*		
Perfect	*64.8%*	*69.1%*	*71.5%*
LRU	*51.2%*	*59.6%*	*58.8%*
LFU	*53.7%*	*60.3%*	*60.4%*
FIFO	*49.2%*	*56.4%*	*58.7%*
LRU-3	*56.5%*	*62.4%*	*64.7%*
histLRUpick	*56.9%*	*62.5%*	*64.9%*
Configuration	*44 machines * 50 Mbytes (8.6%)*		
Perfect	*78.3%*	*83.9%*	*83.9%*
LRU	*60.5%*	*72.2%*	*68.6%*
LFU	*69.8%*	*72.8%*	*71.1%*
FIFO	*63.4%*	*67.6%*	*65.1%*
LRU-3	*72.4%*	*77.6%*	*76.5%*
histLRUpick	*72.9%*	*78.2%*	*76.8%*

- Larger global cache sizes increase the overall hit rate. As the *perfect* run for the 44*50M scenario indicates, it is possible to approach very high hit rates while employing a global cache size that is only 8.6% of the total file size.
- The difference between the replacement policies and *perfect* tends to be less pronounced with decreasing global cache size. This indicates that lack of storage

resource is more of a barrier to performance than replacement policies when the cache size is small.

- LRU-k and HistLRUpick (which is based on LRU-k) show the highest BHR values at large cache sizes. This can be attributed to the ability of LRU-k to exploit temporal locality better than the other caching policies. Examination of the traces revealed that user requests for a particular file tend to arrive at the web server in clusters. By saving the last k references and choosing on basis of the k-th reference, LRU-k ensures that movies which have been referenced multiple times recently are less likely to be removed from the cache since they will have smaller k-distances. Since these same movies will most probably be referenced soon in the future, LRU-k can achieve high hit rates.

4.2.2 Proxy Connections

Having inspected the overall BHR values of the caching algorithms, we now inspect their run-time load balancing performance. In order to examine dynamic load variations during a MiddleMan run, we graphed its *proxy connection* parameter. The proxy connection plot graphs the number of connections currently in the proxy system together with the *max connection*, the maximum number of connections currently at a proxy. Max connection is plotted on the *negative axis* for ease of comparison with the total number of connections in the system. Essentially, this plot indicates the current load imbalance. A well balanced proxy system will have a relatively small maximum connection, even if the total number of connections in the system is high. We define a well balanced system as one that evenly distributes load across proxies. At any given time, for example, the busy proxies in such a system will have similar loads.

The proxy connections plots for the system with LRU and HistLRUpick for the first six months of the trace are illustrated by figures 3 and 4 respectively. The black region in the plots show the total number of connections served by the proxy at any given time whereas the gray areas display the maximum number of connections on a proxy in the system at the same instant, on the negative axis. Ideally, we would like the gray areas to be small compared to the black regions. This would imply that since the most loaded proxy was not heavily burdened, the remaining load was distributed over the other proxies. We would also prefer the maximum connection plot to be relatively smooth. Otherwise, excessive fluctuations would hint at sudden load changes on proxies, an undesirable property for a load balancing algorithm.

Comparison of figures 3 and 4 show that with the exception of day 140, HistLRUpick produces a smoother max connection plot that is more bounded than LRU. On day 140, both LRU and HistLRUpick have the same maximum number of connections (5) on their maximally loaded proxy. However, the peak max values of LRU exceed this threshold a number of times (on days 5, 40, 60, and 170, for instance) whereas HistLRUpick always stays below (or in the case of day 140, equal to) this cutoff value.

4.2.3 Reducing The Number of Proxies

In this subsection, we investigate the effects of reducing the number of proxies but increasing the storage space of each proxy so as to maintain the same global cache size. Figure 5 displays an alternate way of viewing the effects of reducing the number of proxies in MiddleMan. This graph shows overall load imbalance for each configuration by plotting the most aggregate bytes served by a proxy (Max) against the proxy with the least bytes served (Min). We show the results for the *campus* trace and 44*50, 22*100, 11*200 MByte configurations for MiddleMan. Under LRU, there is a great difference between the most and least loaded proxies. However, in each case, HistLRUpick reduces the overall disparity by reducing the traffic on the heavily loaded proxy and increasing traffic to the lightly loaded one.

5. Related Work

Current research on proxy caching has concentrated on caching HTML documents and images [1, 6, 11]. Relatively little prior work has been done on video proxy caching. Most of the current proxy caching work is not applicable to MiddleMan for the following reasons:

- *Document sizes:* web proxy designs and algorithms are optimized for HTML documents and images, which are generally much smaller than web video files [7, 22]. Video files are undesirable in these systems since if they were to be stored in the cache, they would potentially displace many HTML files. As HTML files are likely to be referenced much more frequently than video files, cache hit rates would suffer.
- *Proxy architectures:* MiddleMan has an unique architecture. The proxy architecture most frequently suggested for videos is a centralized server [14,18,19]. As discussed in section 3, this is not an efficient design for a video proxy server.

  Squid or Harvest [9, 24] use a hierarchical design based a tree structure. In these systems, if neighbors of a proxy cannot satisfy a request, the misses propagate upwards through the hierarchy. The miss and propagation combination can add significant latency to final response. Additionally, parents and children caches can potentially store the same files, leading to inefficient use of cache storage space. Hence, such designs have significant drawbacks for caching video.
- *Browsing patterns:* MiddleMan is optimized for video browsing patterns. It supports the notion of partial caching - only portions of video files may be present in the

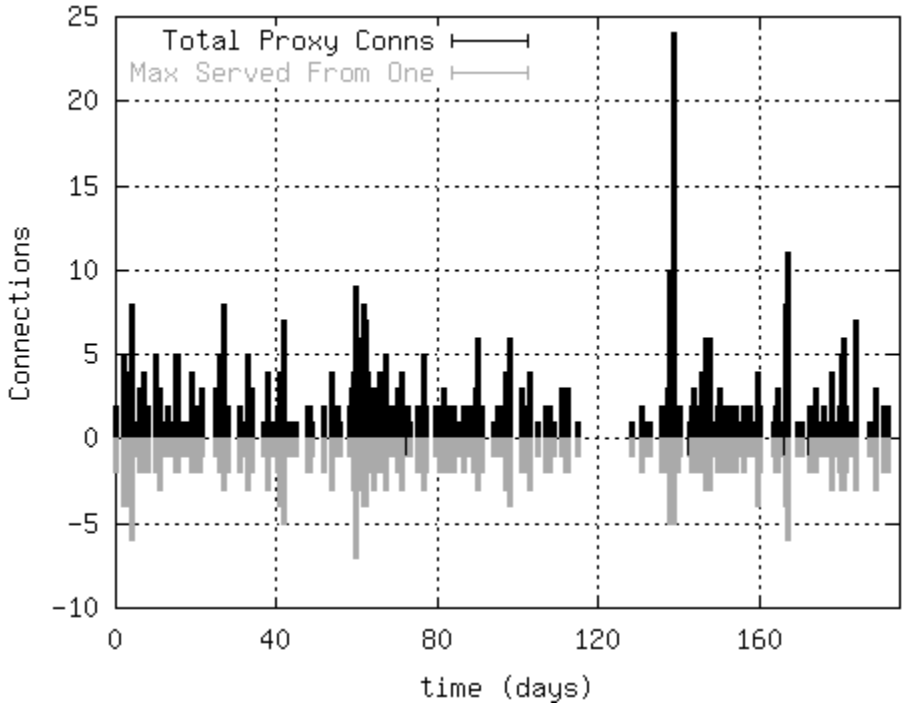

Figure 3: Proxy Connection, LRU, 44*50 MByte

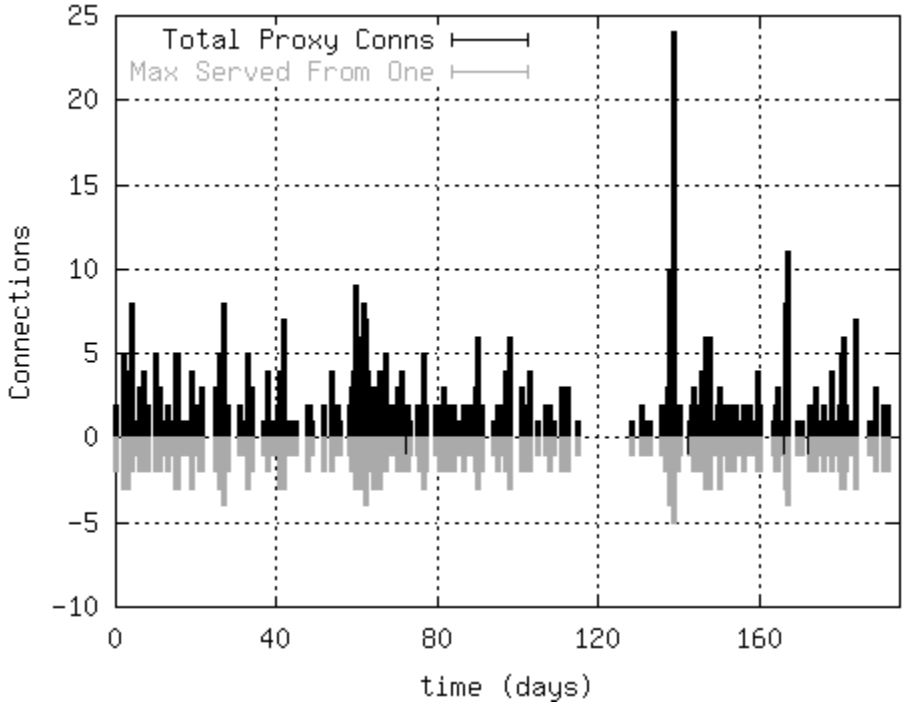

Figure 4: Proxy Connection, histLRUpick, 44*50 MByte

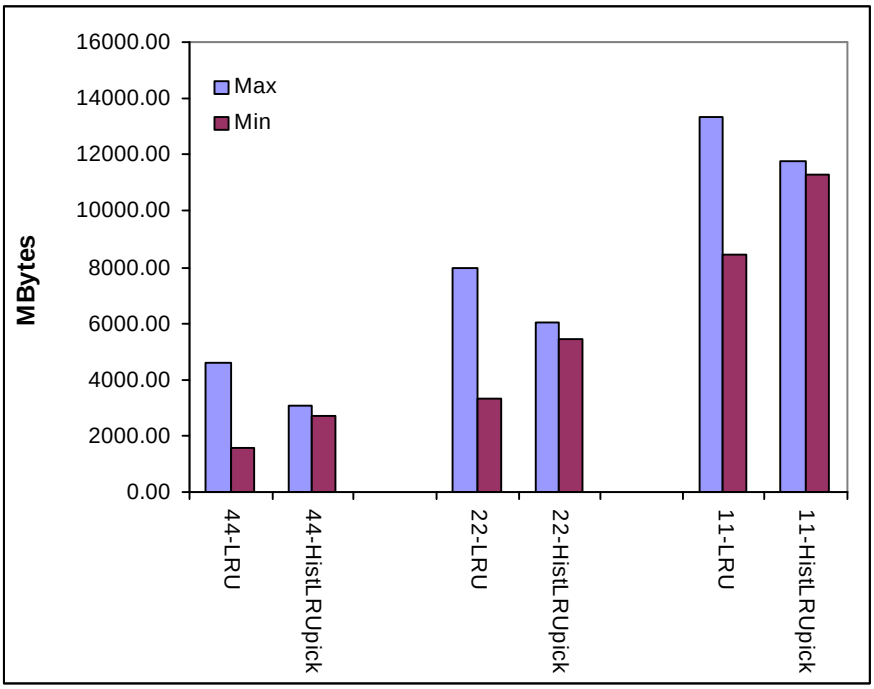

Figure 5: Overall max/min load report for *campus* trace

system. Such an approach is not acceptable for HTML documents or images.

- *Cache coherence:* due to the WORM nature of the vast majority of web videos, this is not a factor for video proxy caches. However, other types of web documents experience high turnover rates and standard proxy caches must deal seriously with cache consistency issues.

The work by Brubeck and Rowe [8] is closest to ours. They introduce the concept of multiple video servers that can be accessed via the web. These video servers manage other tertiary storage systems - popular movies are cached on their local disks. They also pioneer the concept of a movie being comprised of media objects scattered over proxy servers. Other existing work on video proxies [14,18,19] concentrates more on keeping selected segments of videos locally in a central proxy and, for subsequent requests, blending this data with the original stream to ensure smoother viewing. Such an approach is orthogonal to our work. and in fact could be used in conjunction with MiddleMan to improve playback.

Tewari et al [21] present a resource based caching algorithm for web proxies and servers that is able to handle a variety of object types based on their size and bandwidth requirements. However, they still assume a central non-cooperating architecture.

In addition to multimedia research, distributed file systems work also influenced our design. In particular, we used the xFS [5] concepts of network striping and independent location of file blocks in MiddleMan. Additionally, Fox et al [10] provided justification in our decision to centralize the caching/load balancing in the proxy cluster.

6. Conclusion And Future Work

In this paper, we have investigated the performance of MiddleMan, a video caching web proxy system. We found LRU-k to provide the highest hit rates but a variation, HistLRUpick, yielded good hit rates as well as effective load balancing. A relatively small global cache size of 2.14 GBbytes (44*50 Mbytes, about 8.6% of total file sizes) resulted in very high byte hit rates. From the point of view of the server, MiddleMan dramatically reduces load by intercepting a large number of server accesses. Hence, the net effect of MiddleMan is to greatly increase the effective bandwidth of the entire video delivery system by a factor between three and ten, allowing more clients to be serviced at any given time.

MiddleMan shows promise but raises a number of issues that need to be addressed. These include:

- *Fault Tolerance:* the current architecture of MiddleMan renders it susceptible to proxy or coordinator crashes. In particular, a coordinator crash causes the system to become unusable since only the coordinator maintains global system state. Standard techniques such as reliable backup servers may provide a possible solution but leads to two further problems. First, proxies must be able to detect the location of the new coordinator. Second, the process of switching from one coordinator to

another might stall ongoing proxy-client connections. One possible approach to the first problem is to maintain a separate multicast channel within the cluster which can be used to inform all proxies of any configuration changes. The second problem might be avoided by the proxy bypassing MiddleMan and fetching the next block directly from the WWW server.

- *Fast-forward/Rewind Support:* clients may wish to fast forward or rewind through video material. Currently, MiddleMan does not explicitly support such functionality. However, both the proxies and the cache replacement policy can be altered so that once the proxy detects a fast-forward or rewind request from the client, it is able to request the right sequence of blocks from the coordinator.
- *Security/Authentication:* the proxy cache might contain "pay per view" type movies which, if not checked, might allow clients to access titles without authorization from the original movie provider. Hence, an authentication scheme is necessary which would allow MiddleMan to verify whether a client is allowed to retrieve a certain title from the cache. It might also be necessary to encrypt cache contents to prevent unauthorized access to files.
- *Proxy Cluster Cooperation:* in this paper we have investigated the functioning of a single proxy cluster. An obvious next step would be to increase the scope of the system by allowing multiple proxy clusters to interact. In this scenario, if a file was not available in the local proxy, the coordinator might redirect the request to a proxy in a different cluster which does have the file cached. One possible method for achieving such cooperation is for coordinators to periodically exchange data about cache contents on some well known multicast channel.

Future work on MiddleMan will focus on addressing these issues as well as building and deploying a prototype.

References

[1] M. Abrams et. al, *Caching Proxies: Limitations and Potentials*, 4th International World-Wide Web Conference, pp 119-133, December 1995.

[2] S. Acharya, B. Smith, *An Experiment To Characterize Videos On The World Wide Web*, Proceedings of ACM/SPIE Multimedia Computing and Networking 1998 (MMCN'98), San Jose, January 1998.

[3] S. Acharya, B. Smith, *Characterizing User Access To Videos On The World Wide Web*, Multimedia Computing and Networking 2000, San Jose, January 2000.

[4] V. Almeida et al, *Characterizing Reference Locality in the WWW*, Technical Report TR-96-11, Department of Computer Science, Boston University, 1996

[5] T. Anderson, M. Dahlin, J. Neefe, D. Roselli, D. Patterson, and R. Wang. *Serverless Network File Systems*. ACM Transactions on Computer Systems 14, 1, Feb. 1996.

[6] A. Bestavros et al, *Application Level Document Caching in the Internet*, in Proceedings of Workshop on Services and Distributed Environments, June 1995

[7] T. Bray, *Measuring the Web*, in Proc. 4th International World Wide Web Conference, Paris, France, May 1996, pp 994-1005.

[8] D. W. Brubeck, L. A. Rowe, *Hierarchical Storage Management in a Distributed VOD System*, IEEE MultiMedia, Fall 1996, Vol. 3, No. 3

[9] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, K. Worrell, *A Hirerchical Internet Object Cache*, Proceedings of the 1996 USENIX Technical Conference, January 1996

[10] A. Fox, S. Gribble et al, *Extensible Cluster-based Scalable Network Service*s, SOSP-16, Nov 1997, France.

[11] A. Luotonen, K. Altis, *World-wide Web Proxies*, Computer Networks and ISDN Systems 27(2). 1994.

[12] A. Luotonen, *Web Proxy Servers*, Prentice Hall, 1998.

[13] S. McCanne, V. Jacobson, *vic: a Flexible Framework For Packet Video*, Proceedings of ACM Multimedia '95, Nov 1995

[14] Z. Miao, A. Ortega, *Proxy Caching for Efficient Video Services over the Internet*, In Ninth International Packet Video Workshop (PVW '99), New York, April 1999.

[15] E. O'Neil, P. O'Neil, G. Weikum, *The LRU-k Page Replacement Algorithm For Database Disk Buffering*, Proceedings of International Conference on Management of Data, May, 1993

[16] P. Parnes , M. Mattsson, K. Synnes, D. Schefstrom, *mMOD - The multicast Media-On-Demand system*, extended abstract, January, 1997. <URL: http://www.cdt.luth.se/~peppar/docs/mMOD97/mMOD.ps>

[17] P. Parnes , K. Synnes, D. Schefstrom, *The CDT mStar Environment: Distributed Collaborative Teamwork in Action*, Third IT-conference in the Barit region, September 16-17, 1997, Luleå, Sweden

[18] R. Rejaie et al, *Proxy Caching Mechanism for Multimedia Playback Streams in the Internet*, 4th Web Cache Workshop, San Diego, CA., March 1999.

[19] S. Sen et al, *Proxy Prefix Caching for Multimedia Streams*, Proc. IEEE INFOCOM, March 1999.

[20] A. Silberschatz, J. Peterson, P. Galvin, *Operating System Concepts*, Addison Wesley, 1992

[21] R. Tewari, H.M. Vin, A. Dan, D. Sitaram, *Resource-based Caching for Web Servers*, Proceedings of ACM/SPIE Multimedia Computing and Networking 1998 (MMCN'98), San Jose, Pages 191-204, January 1998

[22] A. Woodruff, P. M. Aoki, E. Brewer, P. Gauthier, L. A. Rowe, *An Investigation of Documents from the World Wide Web*, in Proc. 5th International World Wide Web Conference, Paris, France, May 1996, pp 963--979.

[23] http://www.w3.org/Protocols/HTTP/1.1/

[24] http://squid.nlanr.net/squid

[25] http://www.microsoft.com

[26] http://www.netscape.com