An Empirical Study of a Segment-based Streaming Proxy in an Enterprise Environment

Sumit Roy[1], Bo Shen[1], Songqing Chen[2], and Xiaodong Zhang[3]

[1] Streaming Media Systems Group
Hewlett-Packard Laboratories
Palo Alto, CA 94304
[2] Department of Computer Science
George Mason University
Fairfax, VA 22030
[3] Department of Computer Science
College of William and Mary
Williamsburg, VA 23187
zhang@cs.wm.edu

Abstract. Streaming media workloads have a number of desirable properties that make them good candidates for caching via proxy systems. The content does not get modified, and access patterns exhibit some locality of reference. However, media files tend to be much larger in size than traditional web pages, and users tend to view video clips only partially. Hence, segment-based strategies have been proposed to deliver large streaming media objects via a web-proxy. In this work, we evaluate the performance of such a segment-based streaming proxy using extensive trace driven simulation. We use representative workloads from enterprise media server logs, and evaluate the caching system performance regarding different cache sizes, different segment sizes, and different prefetching methods. Our results show that cost-effective caching requires only about 8 - 16 % of the total unique object size as proxy storage. Secondly, a segment size of around 200 Kbytes provides a good trade-off between cache object granularity and transaction overhead. Finally, a lazier prefetching schedule that provides a half-segment look-ahead has a significant performance gain when compared to a more aggressive one-segment look-ahead scheme.

1 Introduction

Recent years have seen abundant applications of streaming media objects on the global Internet, such as remote education, tele-medical treatment, entertainments, etc.. The increased amount of streaming data requires effective and efficient streaming delivery strategies. In an IP based environment, a continuous streaming session (often with a duration of minutes or hours, compared to milliseconds or seconds for traditional web pages) keeps consuming network bandwidth and disk bandwidth on the hosting server. Multiple concurrent streaming sessions can easily exhaust the available network bandwidth and overload the

media content server. Placing multimedia objects closer to clients is an effective solution that would relieve the network bottleneck and distributed the load across multiple streaming media servers. Hence, a lot of research has been done on proxy based streaming delivery approaches, because streaming media objects are generally in-variant, which makes them feasible for proxy caching without worrying about cache consistency [1, 2].

Streaming media object are usually large in size, of the order of multiple MBytes, rather than kBytes, as is more common with web-pages. While caching the complete media object would certainly work, it would require a considerable storage space. Moreover, unless this is augmented by smart read-through techniques, the startup-latency for a client that experiences a cache miss increases dramatically with the clip length. Studies of client access patterns to streaming media objects also show that most clips are only viewed partially [3, 4]. Hence, partial caching approaches are more effective at delivering quality streaming media to clients. In partial caching, the media object is divided or *segmented* into multiple parts. The proxy deals with individual segments for prefetching, replacement, and cache management issues. This has a number of advantages. First, in case of a cache miss, the startup-latency seen by a client is now dependent on the segment size, rather than the clip length. Second, clips that are viewed only partially do not have to be transferred in their entirety across the network. This reduces the load on the backbone infrastructure. Finally, cache utilization becomes more efficient, since the segments are smaller and can be treated as fixed size units. However, there are some challenges to designing a segment-based proxy cache. If a media client encounters a miss for a segment during play-back, there could be an interruption in smooth media delivery. This problem can be solved by providing large buffers at the client, the streaming server, or by using prefetching techniques in the cache. There are various trade-offs to be considered in deciding the segment size. A large segment size increases startup-latency on a miss, decreases the number of hand-shakes required with the back-end server, but might lead to inefficient cache utilization. Finally, it is interesting to evaluate the optimal cache size, based on an understanding of typical workloads.

In this paper, we evaluate the performance of a segment-based streaming proxy using extensive trace driven simulation. We use representative workloads from enterprise media server logs, and evaluate the system performance with regard to different cache sizes, different segment sizes, and different prefetching methods. Our results show that cost-effective caching requires only about 8 - 16 % of the total unique object size as proxy storage. Secondly, a segment size of around 200 Kbytes provides a good trade-off between cache object granularity and transaction overhead. Finally, a lazier prefetching schedule that provides a half-segment look-ahead has a significant performance gain when compared to a more aggressive one-segment look-ahead scheme.

The rest of this paper is organized as follows. We review related work in Section 2. We evaluate the system performance through extensive experiments in Section 3. We conclude the paper in Section 4.

2 Related Work

The research on proxy caching of streaming media content has received much attention lately. Middleman [2] studied clusters of proxies for streaming media delivery. However, they ignored an important feature of streaming media accesses: It is found that continuous media objects such as video or music clips are often partially accessed. Based on this observation, *partial* caching approaches have been proposed to reduce the cache space requirement. The basic strategy is to cache segments of objects that are divided in the viewing time domain. Typical examples include prefix caching [5], uniform segmentation [6], exponential segmentation [7] and adaptive and lazy segmentation [8]. Prefix caching always caches the prefix of the objects to minimize the startup latency. The optimal prefix length can be calculated according to [9]. Its protocol consideration, as well as partial sequence caching, were studied in [10].

In uniform segmentation, objects are cached in uniform-size segments, while in exponential segmentation, the segment size doubles along the viewing direction. Considering limited resources available from a single cache, the Rcache [1] considers the usage of multiple proxies, focusing on the memory and disk utilization. Adaptive and lazy segmentation partitions objects when necessary by considering the user access pattern. These strategies are considered as partial caching strategies, in which only partial data of a media object are cached (Though, very popular objects can be fully cached). These strategies focus on protocol design or benefit analysis based on artificial workloads, emphasizing different performance aspects. Recently, authors in [11] proposed a flexible and scalable proxy testbed to support a wide and extensible set of advanced proxy streaming services. We proposed a segment-based streaming proxy design model in [12]. Compared with the previous work, in this study, we focus on evaluating different performance factors, particularly those related to the cache configurations, thus providing guidance on how to use segment-based streaming proxy in a cost-effective fashion.

The partial caching strategy can be extended to the quality domain. Layered caching techniques [13, 14] have demonstrated efficient usage of cache space by considering different QoS characteristics of client devices or connectivity. A comparison with multiple version caching is studied in [15] while a model of layered-encoded object distribution is studied in [16]. In [17], the proposed approach attempts to select groups of consecutive frames by the selective caching algorithm, while in [18], the algorithm may select groups of non-consecutive frames for caching in the proxy. A different idea is proposed in video staging [19], in which a portion of bits from the video frames whose size is larger than a predetermined threshold is cut off and prefetched to the proxy a priori to reduce the bandwidth on the server proxy channel. Recently, a fine grained, network aware and media adaptive rate control scheme is used in caching of scalable streaming content [20]. Most of partial caching schemes in quality domain require layered encoded objects or additional support from the proxy or client, and little work has been done to evaluate them against different performance factors. The work presented in this paper does not have this limitation.

3 Performance Evaluation

We evaluate the performance of a streaming proxy that uses uniform segmentation of the content, in the time domain. We first describe the evaluation methodology and the workloads used. We then show results of our simulation using different cache sizes, different segment sizes, and different prefetching methods.

3.1 Evaluation Methodology

We test the performance of the proxy by means of a trace driven cache simulator. We first convert a streaming media server log into a segmented trace, using the media object access time, the media offset and duration of the access, the segment length, and the encoding rate to construct a segment access schedule. This somewhat simplifies the actual segment access pattern, since real media files (like Quicktime or MP4) include regions with meta information, and they interleave audio and video data.

This synthesized trace is run through a cache simulator, for cache size varying from 1 % to 64 % of the total unique object size. For each cache size, we evaluate a range of segment sizes, from 20 KBytes to 2000 KBytes. We test three different methods for scheduling segment downloads from the original content server in these experiments. The *ondemand* method is entirely demand based, there is not prefetch. The *window* based prefetching uses a single segment look-ahead window. The access of one segment causes an immediate prefetch of the next sequential segment. This is an aggressive scheme and we expect it to perform well if a client accesses a media object in its entirety. On the other hand, clients that terminate early will reduce the effectiveness of prefetching, since the later segments will not be accessed. Finally, we also test the *half* method, where the next segment is prefetched after half of the current segment has been played back. Prefetching is beneficial for directly reducing client play-back jitter, measurement of this quantity is beyond the scope of this simulation base work.

3.2 Workload Selection

Streaming Media server log studies have shown that there can be considerably skew in terms of object popularity [4, 3]. We selected three different 12 hour traces from a multi-month enterprise media server log. The characteristics of these trace, called LIGHT, MEDIUM, and HEAVY, based on the skew, are detailed in Table 1.

Table 1. Summary of Media Server traces used for evaluation

Workload Name	Num of Requests	Num of Objects	Num of Skewed Accesses (%)	Unique Object Size (GB)	Accessed Data Size (GB)	Accessed Object Size (GB)
LIGHT	655	83	89 (14 %)	2.08	3.232	19.854
MEDIUM	1086	85	376 (35 %)	2.02	3.909	13.314
HEAVY	1840	91	1156 (63 %)	2.44	11.676	34.686

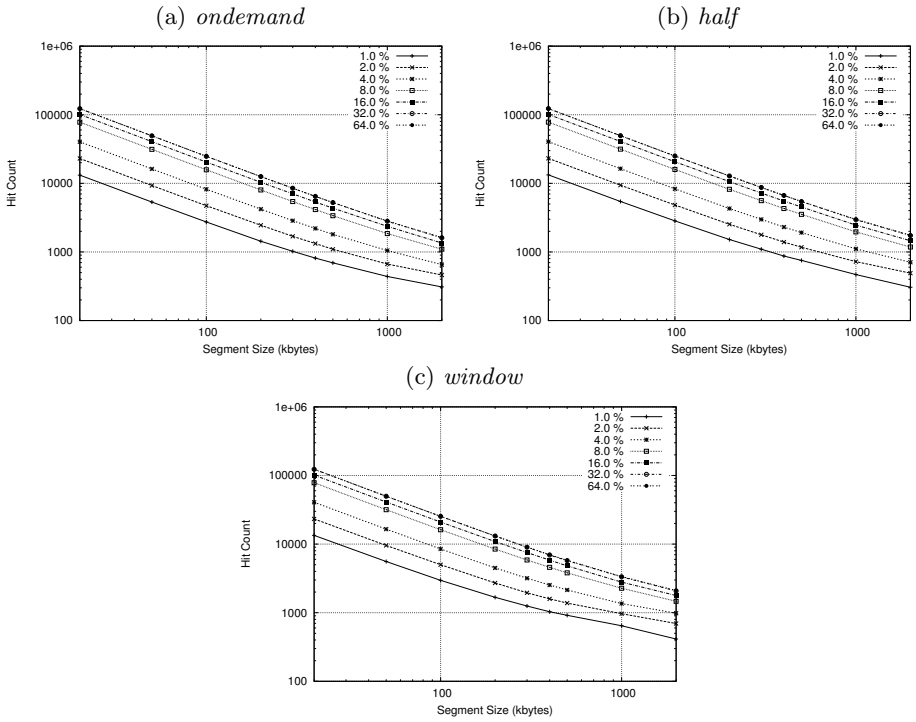

Fig. 1. Cache Hit Counts for LIGHT, different Cache capacities

Unique object size is used as reference for cache capacity. Accessed object size shows the total of accessed objects (full size). Accessed data shows the the portion of it that is actually accessed by the client. The average access duration for these three workloads are 337, 247 and 311 seconds, respectively.

3.3 Simulation Results

The goal of the simulation was to determine the behavior of the proxy with respect to cache size, segment size, and segment scheduling method. We show the condensed results in the next few sub-sections.

Cache Size The proxy cache size is an important design consideration. For every run, we assume a cold-start cache. Intuitively, a large cache performs better, since it can potentially eliminate misses induced due to limited capacity. A cost-effective cache would be sized such that the short term working set completely fits in the cache. Thus, all misses would be cold-start misses, and segment evictions would be for content that is no longer required. Our simulator uses the standard LRU algorithm for replacing cached segments.

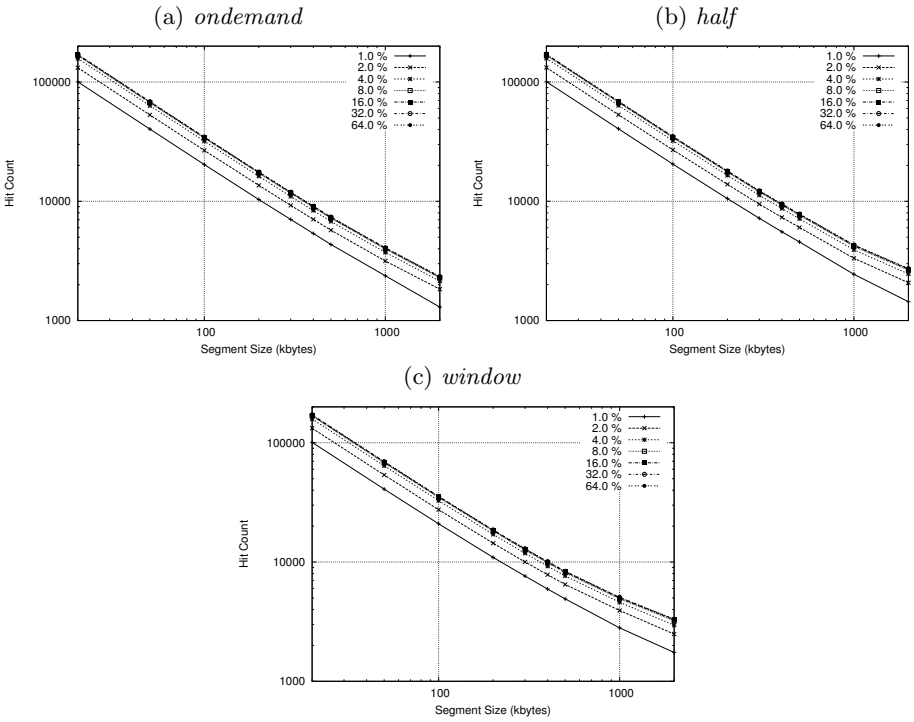

Fig. 2. Cache Hit Counts for MEDIUM, different Cache capacities

Figures 1, 2, and 3 show the Segment Hit Count for the LIGHT, MEDIUM, and HEAVY traces. The Cache capacity is shown as a percentage of the total *unique* object size. It is seen that the Hit Count approximately doubles when the cache size is doubled, up to about a cache size of 8 % for LIGHT, 4 % for MEDIUM, and 2 % for HEAVY. This behavior is consistent across a large range of segment sizes, and is independent of the segment scheduling method. Hence, a cache size of only 8 % of the total unique object size is sufficient for providing good hit rates in the proxy. Note that this translates into a real size of approximately 512 MBytes. This suggests that streaming proxies could be implemented with solid state storage devices like FlashMemory or battery backed DRAM.

Segment Granularity The evaluation of an optimal segment sizes is very important. Smaller segment sizes use the cache more efficiently, and provide lower startup-latencies to a client when there is a cache miss. However, very small segment sizes can cause excessive transactions with the media content server.

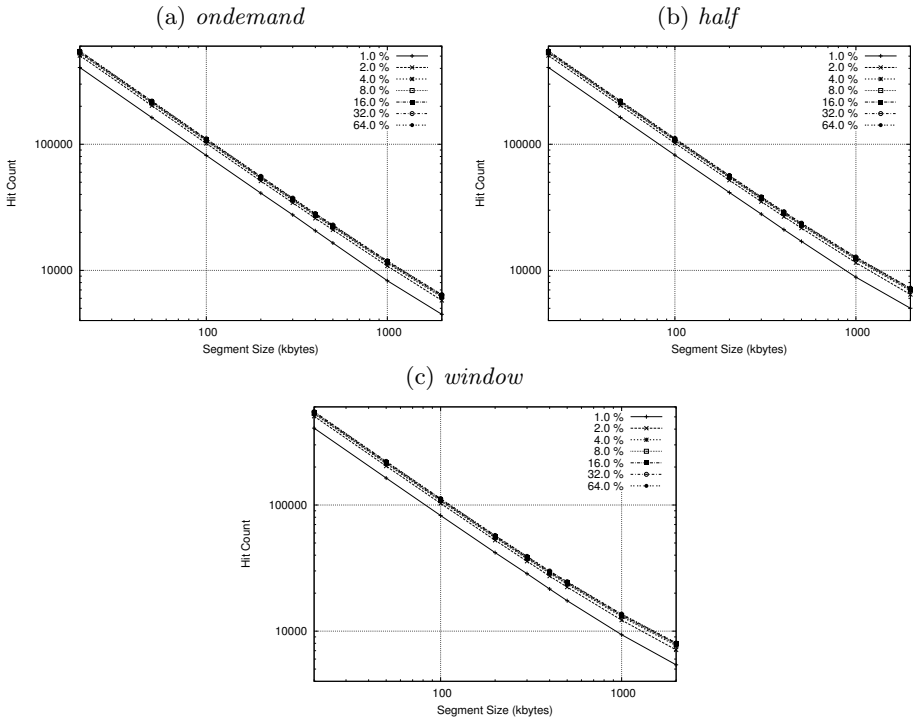

Fig. 3. Cache Hit Counts for HEAVY, different Cache capacities

Figure 4 shows the number of Bytes transferred due to cache misses for the different workloads. Beyond a segment size of 100 - 200 KBytes, the curves start sloping up, which means that extra data is transferred due to the larger segment granularity. This effect is even more pronounced when we look at an additional metric for methods that include prefetching. When a segment schedule includes prefetches, it is possible that a client terminates the session, without ever accessing the contents of the prefetched segment. The amount of data transferred and not used is shown in Figures 5 and 6 for the *window* and *half* method respectively.

For the previously determined optimal cache size of approximately 8 %, it is seen that the amount of useless data increases rapidly for segment sizes greater than approximately 200 KBytes, independent of the access pattern. Hence, the optimal segment size is expected to be around 100 - 200 Kbytes.

Prefetching Effectiveness The prefetching methods *window* and *half* primarily affect client perceived jitter, whose evaluation is beyond the scope of this paper. However, overly aggressive prefetching can lead to the following problem: A prefetched segment might be evicted from the cache due to capacity misses, thus

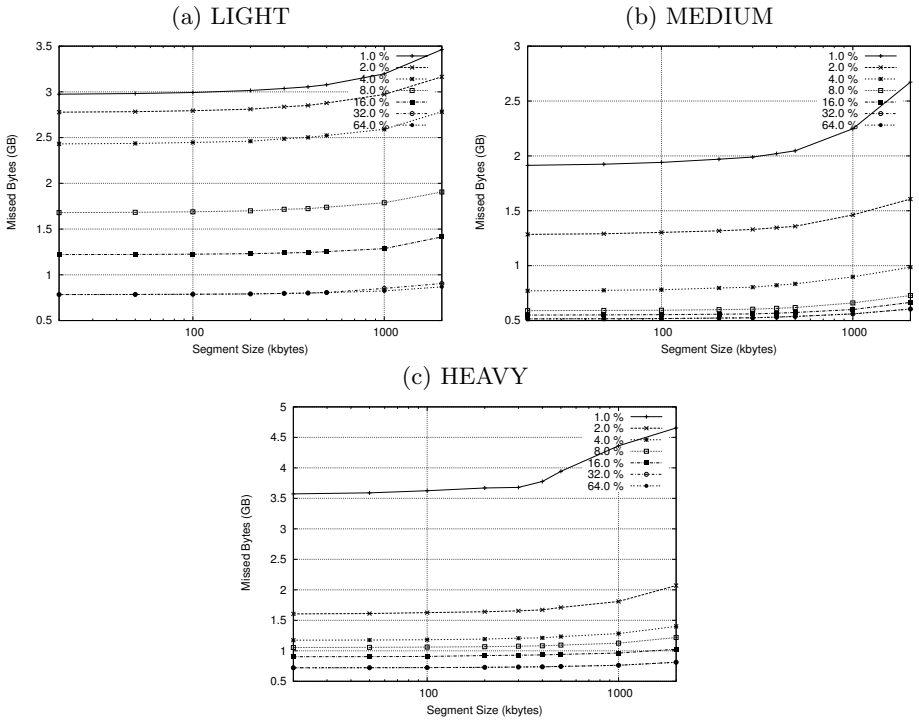

Fig. 4. Missed Bytes of *ondemand* method, different Cache capacities

causing a completely unnecessary data transfer from the media content server. Similarly, a client may terminate its session, before accessing the prefetched segment. The amount of data transferred needlessly this way is shown in Figures 7 and 8 for *window* and *half*, respectively.

The *half* method causes less than 50 % of the wasted data transfer compared to the *window* method for the optimal cache size and segment size determined previously. Since the schedule is less aggressive, when a client terminates its session early, fewer segments are prefetched. This is particularly visible for the LIGHT workload, where there is a more uniform access distribution to the objects. Hence a lazier prefetch method, for example *half* provides a much better performance when measured at the cache. Note that the benefit is independent of the segment size and workload. The amount of data never used, shown previously in Figures 5 and 6, also agree with this assessment.

4 Conclusion

Streaming media workloads are amenable to proxy caching via segmentation. In this paper we evaluate the performance of a segment-based streaming proxy

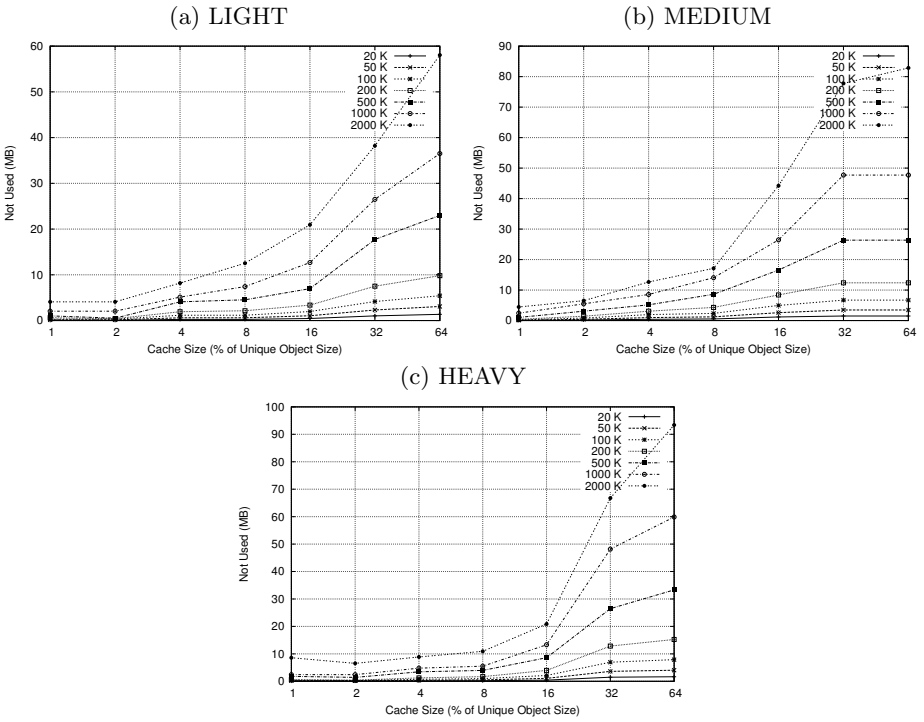

Fig. 5. Data prefetched but never used, *window* method, different Segment sizes

using a trace driven simulation. We use three different traces, which have light, medium, and heavy skew in terms of object popularity. We first determine the optimum cache size by looking at the hit counts for different sizes. Our definition of optimality is based on the marginal performance gain due to an increase in size. Based on this measure, our results show that cost-effective caching requires only about 8 - 16 % of the total unique object size as proxy storage. This is an interesting result, since it suggests that streaming proxies could be implemented with solid state data stores for the enterprise workloads that were considered for this paper.

Next we evaluate the performance of the system using different segment sizes. In this case, we suggest that an increase in segment size should not cause an undue increase in total bytes that are transferred on cache misses. At the same time, we propose that unused data due to prefetches should be kept minimal. For the optimal cache size determined earlier, a segment size of around 200 Kbytes provides a good trade-off.

Finally, we look at two different prefetch schemes, and evaluate them based on the amount of *unnecessary* data transferred due to prefetches. This metric is particularly relevant to the interaction between the proxy cache, and the original

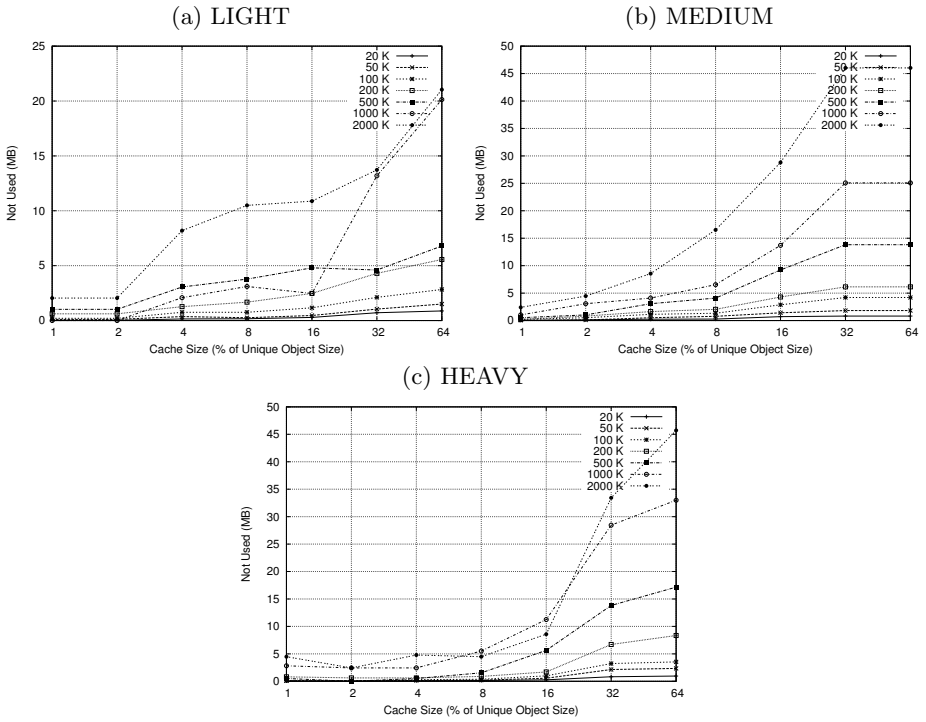

Fig. 6. Data prefetched but never used, *half* method, different Segment sizes

media content server. The less aggressive prefetching schedule issues a request after serving *half* of a segment. It is seen that this method causes less than 50 % of needless data transfers when compared to the more aggressive, *window* scheme.

References

1. Chae, Y., Guo, K., Buddhikot, M., Suri, S., Zegura, E.: Silo, rainbow, and caching token: Schemes for scalable fault tolerant stream caching. In: IEEE Journal on Selected Areas in Communications. (2002)
2. Acharya, S., Smith, B.: Middleman: A video caching proxy server. In: Proc. of ACM NOSSDAV, Chapel Hill, NC (2000)
3. Cherkasova, L., Gupta, M.: Characterizing locality, evolution, and life span of accesses in enterprise media server workloads. In: Proc. of ACM NOSSDAV, Miami, FL (2002)
4. Chesire, M., Wolman, A., Voelker, G., Levy, H.: Measurement and analysis of a streaming media workload. In: Proc. of the 3rd USENIX Symposium on Internet Technologies and Systems, San Francisco, CA (2001)
5. Sen, S., Rexford, J., Towsley, D.: Proxy prefix caching for multimedia streams. In: Proc. of IEEE INFOCOM, New York City, NY (1999)

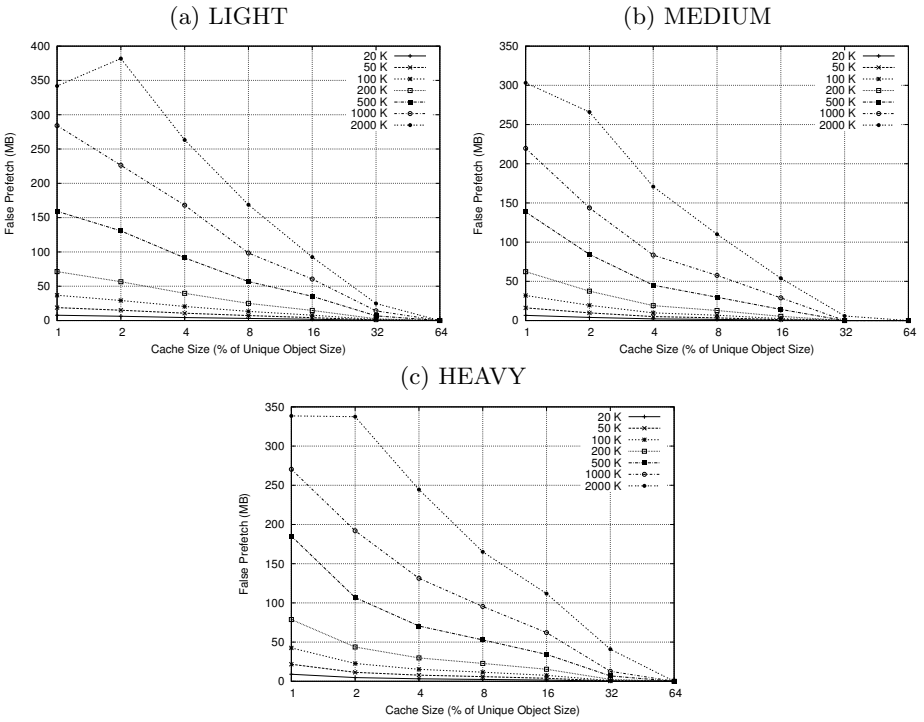

Fig. 7. Data prefetched but evicted before use, *window* method, different Segment sizes

6. Rejaie, R., Handley, M., Yu, H., Estrin, D.: Proxy caching mechanism for multimedia playback streams in the internet. In: Proc. of International Web Caching Workshop, San Diego, CA (1999)
7. Wu, K., Yu, P.S., Wolf, J.: Segment-based proxy caching of multimedia streams. In: Proc. of WWW, Hongkong, China (2001)
8. Chen, S., Shen, B., Wee, S., Zhang, X.: Adaptive and Lazy Segmentation Based Proxy Caching for Streaming Media Delivery. In: Proc. of ACM NOSSDAV, Monterey, CA (2003)
9. Wang, B., Sen, S., Adler, M., Towsley, D.: Proxy-based distribution of streaming video over unicast/multicast connections. In: Proc. of IEEE INFOCOM, New York City, NY (2002)
10. Gruber, S., Rexford, J., Basso, A.: Protocol considertations for a prefix-caching for multimedia streams. In: Computer Network. Volume 33(1-6). (2000) 657–668
11. Zhang, X., Bradshaw, M., Guo, Y., Wang, B., Kurose, J., Shenoy, P., Towsley, D.: Amps: A flexible, scalable proxy testbed for implementing streaming services. Technical report, Department of Computer Science, University of Massachusetts (2004)
12. Chen, S., Shen, B., Wee, S., Zhang, X.: Designs of high quality streaming proxy systems. In: Proc. of IEEE INFOCOM, Hong Kong, China (2004)

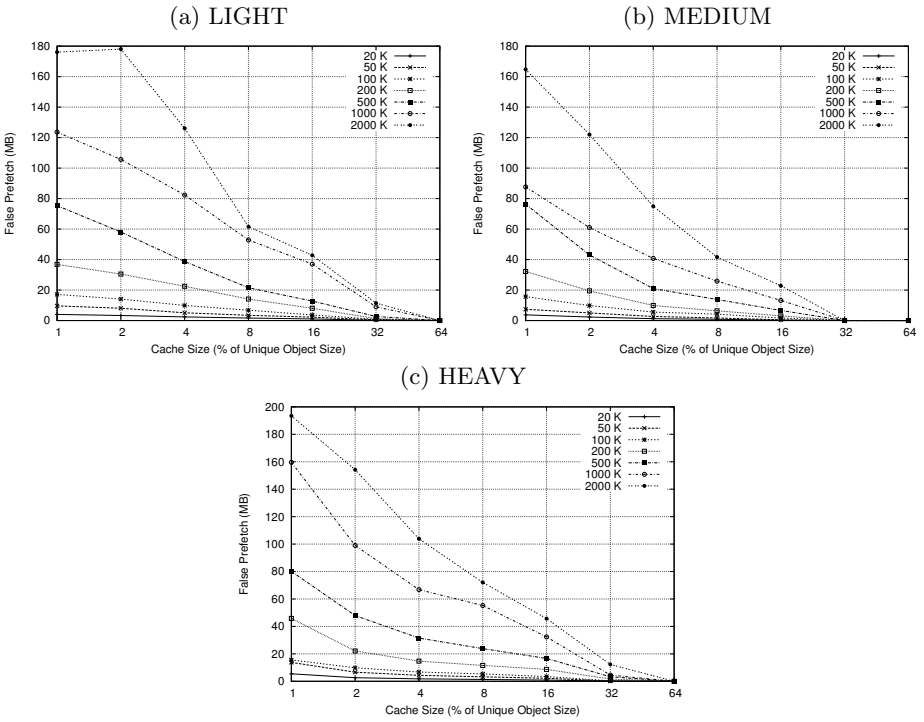

Fig. 8. Data prefetched but evicted before use, *half* method, different Segment sizes

13. Rejaie, R., H. Yu, M.H., Estrin, D.: Multimedia proxy caching mechanism for quality adaptive streaming applications in the internet. In: Proc. of IEEE INFOCOM, Tel-Aviv, Israel (2000)
14. Rejaie, R., Handely, M., Estrin, D.: Quality adaptation for congestion controlled video playback over the internet. In: Proc. of ACM SIGCOMM, Cambridge, MA (1999)
15. Kim, T., Ammar, M.H.: A comparison of layering and stream replication video multicast schemes. In: Proc. of ACM NOSSDAV 2001, Port Jefferson, NY (2001)
16. Kangasharju, J., Hartanto, F., Reisslein, M., Ross, K.W.: Distributing layered encoded video through caches. In: Proc. of IEEE INFOCOM, Anchorage, AK (2001)
17. Ma, W., Du, H.: Reducing bandwidth requirement for dilivering video over wide area networks with proxy server. In: Proc. of IEEE ICME. Volume 2. (2000) 991–994
18. Miao, Z., Ortega, A.: Scalable proxy caching of video under storage constraints. In: IEEE Journal on Selected Areas in Communications. (2002)
19. Zhang, Z., Wang, Y., Du, D., Su, D.: Video staging: A proxy-server based approach to end-to-end video delivery over wide-area networks. In: IEEE Transactions on Networking. Volume 8. (2000) 429–442
20. Liu, J., Chu, X., Xu, J.: Proxy cache management for fine-grained scalable video streaming. In: Proc. of IEEE INFOCOM, Hong Kong, China (2004)