Segment-based Proxy Caching for Internet Streaming Media Delivery

Songqing Chen[1] Haining Wang[2] Bo Shen[3] Susie Wee[3] Xiaodong Zhang[2]
[1]Department of Computer Science, George Mason University, Fairfax, VA 22030
[2]Department of Computer Science, College of William and Mary, Williamsburg, VA 23187
[3] Mobile and Media System Lab, Hewlett-Packard Labs, Palo Alto, CA 94304

Abstract

The proliferation of multimedia content on the Internet poses challenges on existing content delivery networks. While proxy caching proves to be a successful approach to delivering traditional text-based static objects, it faces difficulties in delivering streaming media objects due to much larger sizes of media objects and rigorous continuous delivery demands from clients. In this article, we present two new techniques supporting segment-based proxy caching of streaming media. These techniques have been evaluated experimentally in simulations and real systems.

1 Introduction

Streaming media content delivery has become increasingly important because of the media content proliferation in many application areas, such as education, medical treatment, and entertainment. Although proxy caching has been successful in delivering static text-based content, it has difficulties in delivering streaming media content due to two factors. One is that the size of a media object is generally much larger than a text-based object. Thus, caching entire media objects as static objects can quickly exhaust the proxy cache, making it infeasible. The other is that the client requesting a media object demands continuous streaming delivery. The occasional delays that occur when transferring data over the Internet are acceptable for text-based Web browsing, however, for streaming media data this transfer delay results in playback jitter at the client. The jitter not only is annoying, but also can drive clients away from the streaming service. A *download before watching* solution certainly provides continuous playback, but it introduces a tremendous startup delay. Additionally, it requires very large buffer space on the client.

To overcome these hurdles in streaming media delivery, people have resorted to purchasing the services of proprietary CDNs (Content Delivery Networks). These CDNs can smoothly deliver media content with their dedicated high bandwidth networks and large storage capacities, but this comes with high cost. At the same time, the success of proxy caching of text-based Web objects has seen a number of deployed proxies (or proxy-like nodes) across the public Internet today. These intermediate proxies have plenty of resources, such as computing power, storage, and bandwidth, that are used to cache common-interest content in order to serve different clients in a faster fashion. As an alternative to expensive CDNs, these existing proxy resources can be utilized to deliver media content at very low cost through effective resource management strategies, since media content is normally static and does not change with time.

Researchers have made some efforts on how to use proxy caching for delivering streaming media. Several partial caching approaches [1, 6, 8, 9, 12] have been developed since full-object caching approach is not feasible. Nevertheless, these approaches lack adaptiveness to the dynamically changing popularity of objects and users access patterns. For example, when an object is very popular, most or all of its data should be cached. When the object is unpopular, a small amount or none of its data should be cached. Furthermore, existing schemes have paid little attention to the client demand for a continuous streaming service.

To address these concerns, we have studied how to leverage existing proxy resources to efficiently deliver media content over the Internet. Specifically, we propose the following two techniques to resolve existing problems.

1. *adaptive and lazy segmentation strategy* for partially caching streaming media objects at the proxy and maximizing cache utilization.

2. *active prefetching technique* for actively prefetching uncached segments that a client is likely to access, thus providing the client with continuous streaming delivery.

In the reminder of this article, we first examine ex-

isting partial caching strategies, which motivated our work. Then we describe the adaptive and lazy segmentation strategy and the active prefetching technique. We discuss our ongoing work before making concluding remarks.

2 Partial Caching and Its Limitations

As aforementioned, a number of partial caching approaches have been proposed to divide media objects into smaller units so that only partial units are cached. Typically, there are two partitioning directions.

The first direction is to divide objects in the viewing time domain. We call these segment-based proxy caching approaches. Typical methods include prefix caching [9], uniform segmentation [7], and exponential segmentation [12]. Prefix caching always caches the prefix of objects to reduce the client perceived startup latency because the cached prefix can be immediately served from the proxy to clients. The proxy can retrieve the subsequent segments from the origin server while serving the prefix. In prefix caching, the prefix size plays a vital role in system's performance [11]. In uniform segmentation, objects are segmented according to a uniform length; while in exponential segmentation, objects are segmented in a way that the size of a succeeding segment can double the size of its preceding one. These segmentation-based strategies favor the caching of beginning segments of media objects. A hybrid methodology has been given in [1], in which uniform lengths and exponentially increasing lengths are both considered.

The second direction is to divide objects in the quality domain. For example, the layered media caching [8] and the multiple version caching approaches [5] belong to this category. Layered caching requires the object be layered encoded. The base layer is always cached, while the proxy transfers the enhancement layer(s) if the network bandwidth is available. Multiple version caching statically caches different versions of a media object with different encoding rates. Each version corresponds to a specific type of client network connections. After detecting the type of client network connection, the corresponding version is streamed to the client. Caching different versions is simple, but it consumes a lot of storage space.

These partial caching solutions have improved the performance of proxy caching of streaming media. However, partial caching along the quality domain requires various types of infrastructure support that are not yet widely available, such as the online re-assembly of multiple layers and the transport of multiple layers to the client. Partial caching along the time line does not have such infrastructure requirements, but they cannot address the following concerns. (I) Client accesses to media objects typically represent a skewed pattern: most accesses are for a few popular objects, and these objects are very likely to be watched in their entirety or near entirety. This is often true for movie content in a VoD (Video on Demand) environment and training videos in a corporation environment. A heuristic segment-based caching strategy with a predefined segment size, exponential or uniform, always favorably caches the beginning segments of media objects and does not account for the fact that most accesses are targeted to a few popular objects [2]. (II) The access characteristics of media objects are dynamically changing. A media object's popularity and its most-watched portions may vary with time. For instance, some objects may be popular only for an initial time period when most users access the entire object. In this scenario, using a fixed strategy of caching the first several segments may not work. The reasons are as follows. When the object is popular, it may overload the network as remaining segments need to be retrieved for each client access. On the other hand, as the object popularity diminishes, caching their initial segments may waste proxy resources. The lack of adaptiveness in the existing proxy caching schemes may render proxy caching ineffective. (III) The uniform or the exponential segmentation methods always use the fixed base segment size to segment all the objects through the proxy. However, a proxy is always exposed to objects with a wide range of sizes from different categories, and the access characteristics to these objects can be quite diverse. Without an adaptive scheme, an overestimate of the base segment length may cause an inefficient use of cache space, while an underestimate may induce increased management overhead.

Besides lacking adaptiveness to the dynamically changing popularities of objects and users' access patterns, existing schemes have paid little attention to the client's demand for a continuous streaming service. They cannot always guarantee a continuous delivery because the to-be-viewed segments may not be in the proxy when they are accessed. The problem exists for all segment-based proxy caching approaches, and fetching delay is called *proxy jitter*. The aggregation of proxy jitter may result in *playback jitter* at the client side. Once a playback starts in a client, pausing in the middle due to the proxy jitter is not only annoying, but can also potentially drive the client away from accessing the content. It is very important for a proxy to fetch and relay the demanded segments to the client in time.

The key of removing the proxy jitter is to prefetch

the uncached segments in a timely manner. Some previous work has studied the prefetching of multimedia objects [3, 4, 8]. For layered-encoded objects [8], the prefetching of uncached layered video is conducted by maintaining a prefetching window of the cached stream. The proxy identifies and prefetches all the missing data within the prefetching window, whose length is fixed, before its playback time. In [4], the prefetching is employed to preload a certain amount of data so as to take advantage of the caching power. In [3], a proactive prefetching method utilizes any partially fetched data due to the connection abortion to improve the network bandwidth utilization.

To the best of our knowledge, prefetching methods have not been well studied in the context of segment-based proxy caching yet. Particularly, none of the previous prefetching methods have considered the following conflicting interests in delivering streaming media objects. On one hand, *proxy jitter* is caused by the late prefetching of uncached segments; this clearly suggests that the proxy should prefetch the uncached segments as early as possible. On the other hand, aggressively prefetching of uncached segments significantly increases the buffer space needed for temporarily storing the prefetched data and the network bandwidth needed for transferring this data. Also, it is quite possible that a client may abort an ongoing session before accessing the prefetched segments. The resource efficiency suggests that the proxy should prefetch the uncached segments as late as possible. Therefore, an effective media streaming proxy should be able to decide when to prefetch which uncached segments, subject to minimizing the *proxy jitter* with low resource overhead.

3 Adaptive and Lazy Segmentation

Targeting the limits of existing strategies, we first propose an adaptive and lazy segmentation based caching strategy, which responsively adapts to user access behaviors and lazily segments objects as late as possible. The scheme consists of an aggressive admission policy, a lazy segmentation strategy, and a two-phase iterative replacement policy. As shown in Figure 1, due to our aggressive admission policy, each object is fully cached when it is accessed for the first time. The fully cached object is kept in the proxy until it is chosen as an eviction victim by the replacement policy. At that time, the object is segmented using the lazy segmentation strategy and some segments are evicted according to the first phase of the two-phase iterative replacement policy. From then on, the segments of the object are

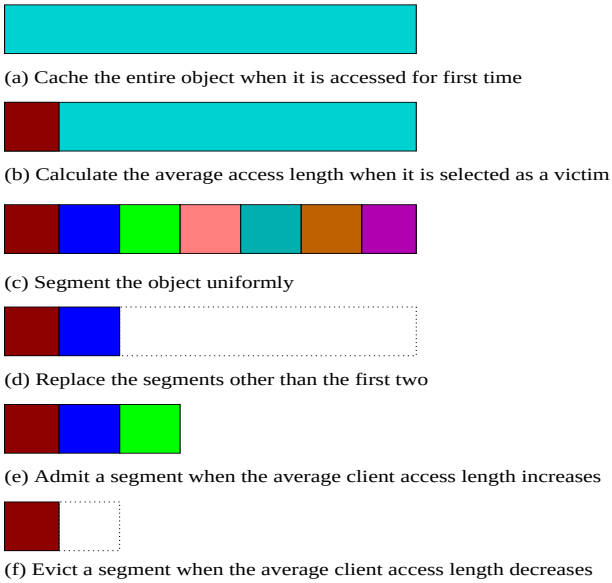

Figure 1: Adaptive and Lazy Segmentation

adaptively admitted by the aggressive admission policy or adaptively replaced as described in the second phase of the two-phase iterative replacement policy. We detail the three components of this strategy in the rest of the section.

3.1 Aggressive Admission Policy

Cache admission is activated each time when a media object is accessed. The object is fully admitted if it is accessed for the first time, which is shown in Figure 1(a). The system does the full caching because (1) whether this object will be popular or not is unknown; and (2) if it is popular, the portion which will be accessed most is also unknown. After the first access, the system keeps track of the average client access length by recording necessary information. Later on, the object will be segmented with respect to the user access pattern, then the admission policy works in segments. When the average users' access length changes, the admission policy adapts to the dynamics of user access patterns correspondingly, which is shown in Figure 1(e).

3.2 Lazy Segmentation Strategy

In current segmentation strategies, the segmentation is performed with a pre-determined base segment length before an object is accessed for the first time, such as the exponential segmentation strategy. By contrast, the adaptive and lazy segmentation strategy segments the object as late as possible: when a victim is selected to make room for some incoming objects, the selected vic-

tim is not segmented yet. As shown in Figure 1(b), the proxy uses the average client access length computed at that moment as the unit for segmentation, i.e., the base segment length of this object. The whole object is then segmented uniformly according to its base segment length, which is illustrated in Figure 1(c). Since the lazy segmentation strategy delays the segmentation process as late as possible, the proxy can gather a sufficient amount of accessing statistics to properly segment each media object. Moreover, the proxy can adaptively set different base segment lengths for different media objects according to on-line users' access behaviors.

3.3 Two-Phase Iterative Replacement Policy

Since the proxy has a limited cache space, the replacement is inevitable. Figures 1(d) and 1(f) show two phases of the replacement policy. The amount of replaced data at these two phases are different.

A crucial aspect of the replacement policy lies in selecting a victim. The more appropriate the victim is selected, the higher benefit the caching system gains. Instead of adopting a LRU (least recently used) to select the least recently used object as a victim, which is used in exponential segmentation, our system considers the following factors.

1. the average number of accesses;
2. the average duration of accesses;
3. the length of the cached data, i.e., the cost of the storage; and
4. the predicted probability of the future access.

The first three items are tracked from client accesses, whereas the last one depends on predicting of future accesses. However, the last one plays a more important role since it predicts the trend of the object's popularity variations. The prediction is done as follows. When the object is accessed for the first time, the time is termed as T_1. The current time is termed as T_c, and the access time of the last (or most recent) access is termed as T_r. The number of accesses so far is termed as n. Thus, the system computes the $T_c - T_r$, the time interval between now and the most recent access, and the $\frac{T_r - T_1}{n}$, the average time interval for an access happened in the past. If $T_c - T_r > \frac{T_r - T_1}{n}$, the possibility that a new request arrives soon for this object is small. Otherwise, it is very likely that a new request may arrive in the near future. In our utility function, the utility value of an object is thus proportional to items (1), (2), (4), and inversely proportional to item (3).

According to this utility function, the object with the smallest utility value is always chosen as the victim. If the selected object turns out to be entirely cached, the first phase of replacement policy is activated: after the segmentation is done, the first two segments are kept in the cache, while all the rest segments are evicted. Figure 1(d) depicts this step. The first two segments are kept because caching these two segments covers most client accesses that follow a normal distribution. If the selected victim is already partially cached, then the second phase of the replacement policy is activated: as shown in Figure 1(f), it always evicts the last cached segment of the selected victim, and the system iteratively performs replacement until sufficient cache space is found.

The design of the two-phase iterative replacement policy reduces the chance of making a wrong decision of the replacement, and gives a fair chance to the replaced segments so that they can be cached back into the proxy again by the aggressive admission policy if they become popular again.

3.4 Typical Performance in Simulations

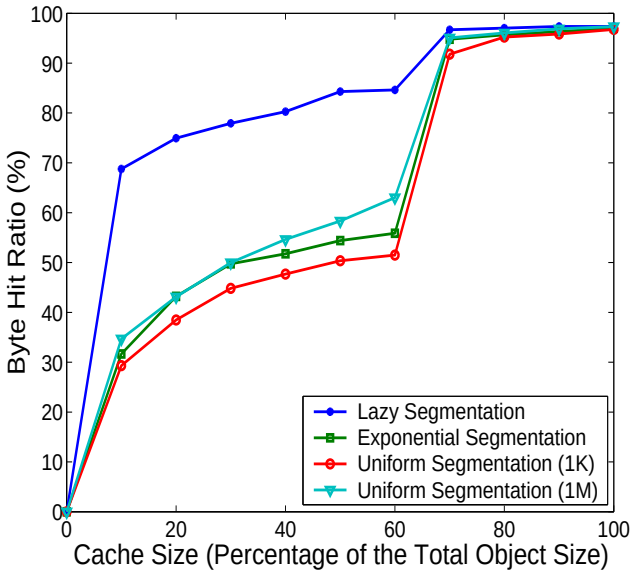

Figure 2: Byte Hit Ratio of an Actual Workload

To test the performance of our design, we use three synthetic workloads and one actual workload extracted from HP Media Server logs. The general trends reflected on these workloads are similar but the detailed variations depend on the individual workload. We only present the results based on the real workload for brevity.

The actual workload of HP Corporate Media Solutions covers the period from April 1 to April 10, 2001. It includes accesses to a total of 403 objects, and the unique object size amounts to 20 GB. There are 9000 requests. Our trace-analysis reveals that 83% requests only view the objects for less than 10 minutes and 56% requests only view the objects for less than 10% of their content. Only about 10% requests view the whole objects. For synthetic workloads, we assume Poisson distribution for request inter-arrivals and Zipf-like distribution for object popularities.

The performance results of the actual workload are shown in Figures 2 and 3. Byte hit ratio is defined as the amount of data delivered from the proxy, normalized by the total amount of data demanded by clients. In these figures, *lazy segmentation* represents our proposed strategy, while the others are self-explained. Note that for the uniform segmentation strategy, different segment sizes are used and their results vary in a certain extent. We can see that lazy segmentation always achieves the best performance in terms of the byte hit ratio, leading to the highest network traffic reduction and evidencing the effectiveness of the adaptiveness of our proposed approach. Particularly, when the cache size is in the range of 20% - 60% of the total object size, the performance improvement is much higher than that when the cache size is larger than 70% of the total object size. This is due to the fact that when the cache size reaches 70% of the total object size, the cache space is large enough to accommodate the beginning portions and most popular segments in the workload. Thus, the byte hit ratio improvement is trivial. However, when the available cache is not large enough, our proposed algorithm caches the segments according to their popularities; while other strategies try to cache the beginning segments of each object first, regardless of whether the object is popular or not, thus resulting in a lower cache performance as shown on Figure 2.

Similar performance trends are observed when other synthetic workloads are evaluated. Note the synthetic workloads are generated based on the previous research results on streaming media workload characterizations [2], and the performance of our scheme depends on the client access pattern. For workloads with the same object popularities and lengths, our scheme cannot achieve a better performance.

Figure 3 shows the performance of different schemes with respect to startup delay. Delayed start request ratio is defined as the number of requests with a start delay normalized by the total number of requests. We can see that the proposed adaptive-lazy scheme performs almost similar as the best existing scheme.

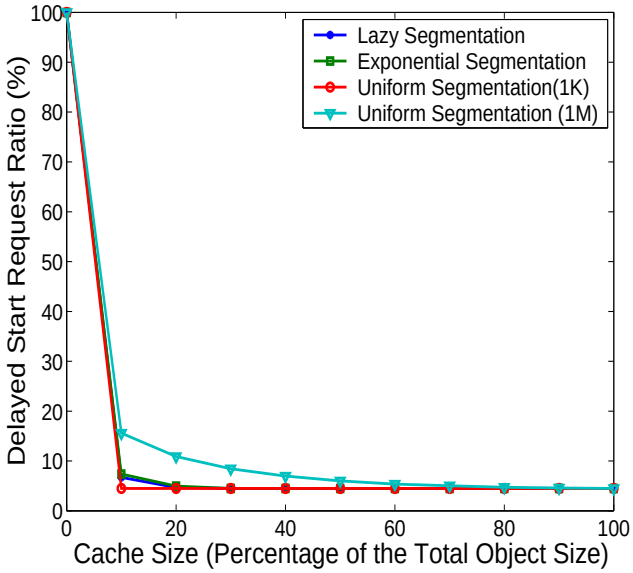

Figure 3: Delayed Start Request Ratio of an Actual Workload

4 Active Prefetching

Streaming media systems are designed to provide streaming services to clients. Thus, the performance metric, such as the byte hit ratio, is only important for a proxy caching system from the system point of view, but it is not the ultimate concern of the client. From the client point view, a quality streaming is jitter free with small startup latency. Thus, the client always expects the streaming delivery to start immediately after the link is clicked. More important, during the delivering service, the playback is expected to be continuous without any interruption. On the other hand, these client side metrics are tied to the system side metrics, and a better system resource utilization may not result in a better service quality to clients: if the beginning segments of an object is cached in the proxy when it is accessed by the client, the client experiences no delay. However, to cache all the beginning portions of each object in the proxy costs a huge amount of cache space and leaves less space for caching popular segments, thus decreasing the cache performance. Similarly, if the later segments of an object is not cached while they are accessed by the client, the delay, resulting in playback jitter at the client side, is inevitable if the network link bandwidth from the proxy to the origin server is not sufficient. But to cache the later segments of an object may need to evict the popular segments or the beginning segments of other objects. An efficient system needs to balance these goals in the design.

In order to provide a continuous streaming delivery

to clients, we propose an *active prefetching* scheme to prefetch the uncached segments from the content server in anticipation of the client's continuous access. Compared with the passive fetching, in which the fetching of uncached segments happens when the client misses it, our scheme is called *active prefetching* because the proxy prefetches the uncached segments that are likely to be accessed based on the prediction of future client accesses. In this scheme, the aforementioned conflicting objectives are carefully balanced so that *proxy jitter* is removed without wasting network and proxy resource.

To conduct our prefetching scheme, we have made the following assumptions.

- The streaming is faster than the data transfer. Otherwise, prefetching is less critical;
- A media object has been segmented and is accessed sequentially;
- The bandwidth of the proxy-client link is large enough for the proxy to deliver the content to a client smoothly; and
- Each segment of the object is delivered by a media server in a unicast way.

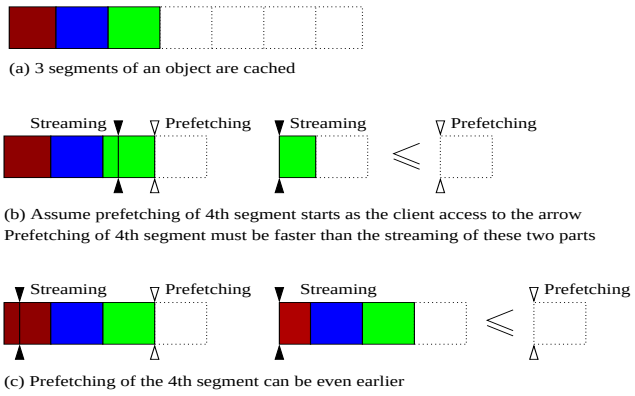

Figure 4: An Example for Segment Prefetching

Figure 4 shows a case study in prefetching. Figure 4(a) illustrates that at the beginning, the first three segments of an object have been cached. When a client accesses the object, Figure 4(b) shows one possible scenario where the proxy prefetches the fourth (uncached) segment after the client starts to access the third one. This is one segment ahead prefetch. In this scenario, to guarantee a continuous streaming, the prefetching of the entire fourth segment must be faster than the streaming of the rest of third segment plus the fourth segment. This requirement is shown on the right side of Figure 4(b). If the prefetching is always performed by one segment ahead, it is straightforward to infer that the streaming speed cannot be two times faster than the prefetching delay. Otherwise, a continuous streaming service could be interrupted. So, if the time difference between the streaming speed and the prefetching delay is large, the proxy has to prefetch earlier. Figure 4(c) shows such a scenario, in which the prefetching of the fourth segment should be as early as the client just starts accessing the first segment. Overall, based on the streaming speed and prefetching delay, the starting point of the prefetch can be accurately calculated.

The prefetching delay depends on the available bandwidth between the proxy and the content server. The bandwidth can be periodically measured using tools such as PCAP (Packet CAPture) library.

Correspondingly, we can figure out the temporary storage requirement for prefetching. Figure 5 shows how to precisely calculate the storage size. Figure 5(a) indicates the time point when the prefetching is scheduled to start, while Figure 5(b) illustrates the situation when the streaming of the prefetched data starts. If the storage is a circular buffer, the amount of prefetched data between time 1 and time 2 is the maximum buffer size that the proxy needs. Therefore, if the client terminates before viewing any prefetched data, the resource wastage is maximized. The maximum wastage includes the network bandwidth for data transferring and the storage for storing this amount of data.

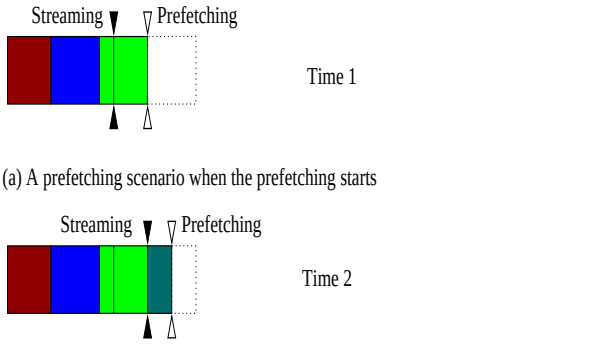

Figure 5: Storage Size Needed for Prefetching

4.1 Typical Performance in Real Systems

We have implemented the system as presented before. To evaluate its performance, we reproduce the three-month actual workload, which is also used in previous simulations. Among the reproduced workload, a 12-hour trace is extracted to run in the deployed system.

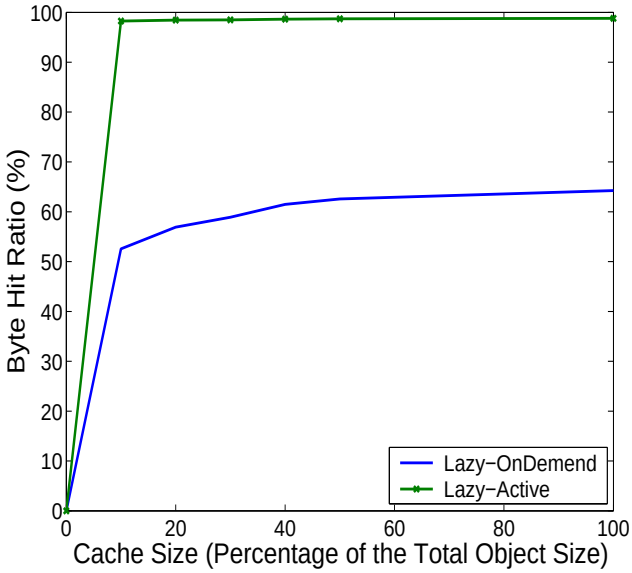

Figure 6: Byte Hit Ratio from a Real System

In the 12-hour workload, file sizes range from 1 minute to 100 minutes, and the object encoding rates include 28, 56, 112, 156, 180, and 256 Kbps. The unique object size amounts to 8.826G. The experiments are done by putting the system in a local network. Both the server and the proxy are of 2G Hz Pentium III CPU with 1 GB memory. Our experimental results are shown in Figure 6. *Lazy-OnDemand* represents the scheme without the prefetching support, and *Lazy-Active* represents the scheme with the active prefetching support. The performance of *Lazy-OnDemand* in Figure 6 shows that in this 12-hour trace, the client access is not skewed, while the performance gain by *Lazy-Active* with respect to *Lazy-OnDemand* indicates the importance of prefetching. Clearly, with the assistance of active prefetching, a much larger percentage of uncached data can be delivered to the clients in time, reflected as the higher byte hit ratio. The higher byte hit ratio implies that more data can be served from the proxy to clients in time, resulting in the less playback jitter to the clients. Active prefetching is effective because the uncached segments are prefetched at a right time, which considers the streaming rate and the available bandwidth for the prefetching. Compared with other strategies, such as intuitive sliding-window based prefetching, the scheduling needs some computation and the efficiency is much higher. Note that the computation of the starting point of prefetching only costs 0.1% of the total CPU cycles at the proxy in our experiments. On the other hand, compared with the simulation results shown in Figure 2, the implemented system achieves less but realistic gains. The reasons are a shorter trace and some trade-offs made in the implementation. Since the real workload where the 12-hour trace is extracted from is a workload containing a lot of premature client terminations, where the client only accesses a small portion of the object before termination, the byte hit ratio achieved with the active prefetching quickly reaches its plateau when the cache size is only a small percentage of its total object size.

5 Current Efforts

Today a client can use a PDA, a cell-phone or other mobile devices, instead of high-end PCs, to access the Internet. The wide usage of mobile devices further complicates the Internet streaming media delivery. Due to the different screen sizes and color depths, a media object that is appropriate to a desktop computer may not be appropriate to a PDA. The media delivery network has to distinguish and adapt to different client devices. According to the type of a client device, an appropriate version should be chosen and streamed. Moreover, the usage of mobile devices induces the mobility problem. The media delivery network has to consider the mobility of clients, as the client may frequently move from one place to another. Thus, we are trying to address the following two questions.

1. How to convey different versions of a media object to different client devices?
2. How to deal with the client mobility?

The following sections present our initiative efforts and clues to address these two problems.

5.1 Proxy Enabled Transcoding

To deal with the diversity of client browsing devices, we seek the proxy enabled transcoding as a solution. A proxy can cache the transcoded media objects and deliver them for a variety of future client references, which prevents the repeated transcoding operations. The challenge here is that, the media delivery network must be able to distinguish and adapt to different client devices. So far, there are two different ways to perform continuous media transcoding. The first one is to store multiple versions of an object in advance, called offline-transcoding, in which different versions for all kinds of devices are well prepared before their streaming service is available. Its drawback is the huge storage consumption for storing all versions of an object. The second solution is online-transcoding, where the transcoding and delivering are conducted simultaneously. While the storage requirement is moderate, this approach burns a

large amount of CPU cycles on the fly. Based on media segments, we are considering a hybrid solution that meets the diversity requirement of media delivery with reasonable overhead at a proxy server.

5.2 Proxy Hand-off

The usage of mobile devices not only incurs the aforementioned transcoding problem, but also introduces the mobility problem. When a client holding a PDA or cell-phone reads the video-based news, the client may be in a moving train, or may walk on the street. Thus, the media delivery network should provide a nomadic streaming service. This implies one streaming proxy is not capable of providing a continuous streaming service, given each base station is associated with a proxy. To cope with the client mobility, we resort to cooperative proxy caching techniques to reduce the number of expensive proxy hand-offs. However, how to coordinate different proxies for continuous media delivery is difficult, since the hand-off between the base stations and the client can easily disrupt the continuity of streaming media delivery.

There are other problems related to proxy based streaming delivery, such as live streaming, and the power consumption in mobile devices. It is more difficult to exploit the current proxy caching techniques for delivering live streams, since the real-time requirement of live stream delivery is even more rigorous. Authors in [10] has investigated how to reduce the energy consumption of streaming media at mobile devices. A proxy-based power-friendly transformation technique is used to limit the energy consumption of receiving and decoding stream media.

6 Conclusion

This article presents problems and solutions on the proxy-based streaming media delivery. Specifically, the adaptive and lazy segmentation scheme is presented to promptly identify and cache the most popular segments of each media object in the proxy, thereby maximizing the cache utilization. Additionally, the active prefetching technique is presented to pro-actively prefetch the uncached segments which clients are very likely to access in the future, thus providing a continuous streaming service to clients. Currently we are investigating enhanced proxy functions to efficiently handle the diversity and mobility of client devices.

7 Acknowledgments

Our research activities are supported by NSF and Hewlett-Packard Labs.

References

[1] Y. Chae, K. Guo, M. Buddhikot, S. Suri, and E. Zegura. Silo, rainbow, and caching token: Schemes for scalable fault tolerant stream caching. In *IEEE Journal on Selected Areas in Communications, Special Issue on Internet Proxy Services*, September 2002.

[2] M. Chesire, A. Wolman, G. Voelker, and H. Levy. Measurement and analysis of a streaming media workload. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, San Francisco, CA, March 2001.

[3] J. Jung, D. Lee, and K. Chon. Proactive web caching with cumulative prefetching for large multimedia data. In *Proceedings of WWW*, Amsterdam, Netherland, May 2000.

[4] J. I. Khan and Q. Tao. Partial prefetch for faster surfing in composite hypermedia. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, San Francisco, CA, March 2001.

[5] T. Kim and M. H. Ammar. A comparison of layering and stream replication video multicast schemes. In *Proceedings of ACM NOSSDAV 2001*, Port Jefferson, NY, June 2001.

[6] Z. Miao and A. Ortega. Scalable proxy caching of video under storage constraints. In *IEEE Journal on Selected Areas in Communications*, volume 7, pages 1315–1327, Sept. 2002.

[7] R. Rejaie, M. Handley, H. Yu, and D. Estrin. Proxy caching mechanism for multimedia playback streams in the internet. In *Proceedings of International Web Caching Workshop*, San Diego, CA, March 1999.

[8] R. Rejaie, H. Yu, M. Handely, and D. Estrin. Multimedia proxy caching mechanism for quality adaptive streaming applications in the internet. In *Proceedings of IEEE INFOCOM*, Tel-Aviv, Israel, March 2000.

[9] S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *Proceedings of IEEE INFOCOM*, New York City, NY, March 1999.

[10] P. Shenoy and P. Radkov. Proxy-assisted power-friendly streaming to mobile devices. In *Proceedings of the 2003 Multimedia Computing and Networking Conference*, Santa Clara, CA, January 2003.

[11] B. Wang, S. Sen, M. Adler, and D. Towsley. Proxy-based distribution of streaming video over unicast/multicast connections. In *Proceedings of IEEE INFOCOM*, New York City, NY, June 2002.

[12] K. Wu, P. S. Yu, and J. Wolf. Segment-based proxy caching of multimedia streams. In *Proceedings of WWW*, Hongkong, China, May 2001.