Segment-Based Proxy Caching for Internet Streaming Media Delivery

Songqing Chen
George Mason University

Haining Wang and Xiaodong Zhang
College of William and Mary

Bo Shen and Susie Wee
Hewlett-Packard Labs

The proliferation of multimedia content on the Internet poses challenges to existing content delivery networks. While proxy caching can successfully deliver traditional text-based static objects, it faces difficulty delivering streaming media objects because of the objects' sizes as well as clients' rigorous continuous delivery demands. We present two techniques supporting segment-based proxy caching of streaming media. We evaluated these techniques in simulations and real systems.

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 successfully delivers static text-based content, it has difficulty delivering streaming media content. One reason for this is that a media object's size is generally much larger than a text-based object. Thus, caching entire media 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 causes the client to experience playback jitter. This is annoying and could drive clients away from the streaming service. A download-before-watching solution certainly provides continuous playback, but it also introduces a tremendous startup delay. Additionally, it requires the client to have a large buffer space.

To overcome these hurdles in streaming media delivery, people have resorted to purchasing the services of proprietary content delivery networks (CDNs). These CDNs can smoothly deliver media content with their dedicated high-bandwidth networks and large storage capacities, but they can be costly.

At the same time, the success of proxy caching text-based Web objects has seen a number of deployed proxies (or proxy-like nodes) across the Internet. These intermediate proxies have plenty of resources—such as computing power, storage, and bandwidth—that can cache common-interest content to serve different clients more quickly than the client directly accessing the servers. As an alternative to expensive CDNs, these existing proxy resources can deliver media content inexpensively through effective resource management strategies, since the content of a media object doesn't change with time.

Researchers have tried using proxy caching to deliver streaming media. Several partial caching approaches[1-5] have been developed, because full-object caching isn't generally feasible. Nevertheless, these approaches can't really adapt to the dynamically changing popularity of objects and user access patterns. For example, when an object is 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 studied how to leverage existing proxy resources (including partial caching strategies) to efficiently deliver media content over the Internet. We proposed, evaluated, and implemented the following techniques to resolve existing problems:

- *Adaptive and lazy segmentation.* We use this to cache partial streaming media objects at the proxy and maximize cache usage.
- *Active prefetching.* We use this to actively prefetch uncached segments that a client is likely to access, thus providing the client with continuous streaming delivery.

Partial caching and its limitations

As we previously mentioned, many partial-caching approaches exist to divide media objects into smaller units so that only partial

1070-986X/05/$20.00 © 2005 IEEE

Published by the IEEE Computer Society

units are cached. Typically, we can consider one of 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,[4] uniform segmentation,[6] and exponential segmentation.[5] Prefix caching always caches the prefix of objects to reduce the client-perceived startup latency because the proxy can immediately serve the cached prefix 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 performance.[7]

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. Chae et al.[1] proposed a hybrid methodology 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[3] and the multiple-version caching approaches[8] belong to this category. Layered caching requires that we encode the object in layers. 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 connection. After detecting the connection type, the system streams the corresponding version to the client. Caching different versions is easy, but it consumes a lot of storage space.

These solutions improve the performance of proxy caching streaming media. However, partial caching along the quality domain requires various types of infrastructure support that aren't widely available yet, such as the online reassembly of multiple layers and the transport of multiple layers to the client. Partial caching along the time line doesn't have such infrastructure requirements. However, it can't address the following concerns:

- *Client accesses to media objects typically represent a skewed pattern.* Most accesses are for a few popular objects, and it's likely that clients will watch these in their entirety or near entirety. This is often true for movie content in a video-on-demand (VoD) environment and training videos in a corporate 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 doesn't account for the fact that most accesses are targeted to a few popular objects.[2]

- *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 because the system needs to retrieve the remaining segments for each client access. On the other hand, as the object's popularity diminishes, caching the initial segments may waste proxy resources. The lack of adaptability in the existing proxy-caching schemes might render proxy caching ineffective.

- *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 popularity of objects and users' access patterns, existing schemes don't address the client's demand for a continuous streaming service. They can't always guarantee a continuous delivery because the to-be-viewed segments may not be in the proxy when clients access them. The problem exists for all segment-based proxy caching approaches, and the fetching delay results in proxy jitter. The aggregation of proxy jitter may result in playback jitter at the client side. As we previously noted, playback jitter is annoying and can potentially drive the client away from access-

ing the content. So it's important for a proxy to fetch and relay the demanded segments to the client in real time and without delay.

The key to removing the proxy jitter is to prefetch the uncached segments in a timely manner. Some previous work has studied the prefetching of multimedia objects.[3,9,10] For layered-encoded objects,[3] 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 Khan and Tao,[10] the prefetching preloads a certain amount of data to take advantage of the caching power. In Chesire et al.,[11] a proactive prefetching method uses any partially fetched data from a connection abortion to improve the network bandwidth usage.

To the best of our knowledge, so far little research exists on prefetching methods in the context of segment-based proxy caching. Particularly, no previous prefetching methods considered the conflicting interests that we note in delivering streaming media objects.

On the one hand, the late prefetching of uncached segments causes proxy jitter; this clearly suggests that the proxy should prefetch the uncached segments as early as possible. On the other hand, aggressively prefetching 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's 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.

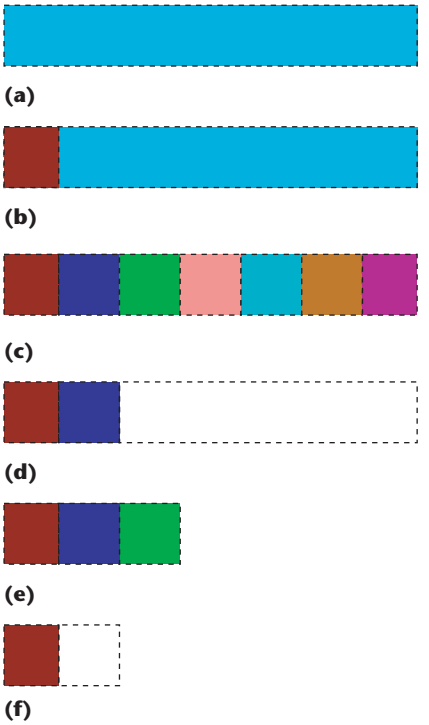


Figure 1. Adaptive and lazy segmentation. (a) Caching the entire object the first time a client accesses it. (b) Calculating the average access length when the server selects it as a victim. (c) Segmenting the object uniformly. (d) Replacing segments (other than the first two). (e) Admitting a segment when the average client access length increases. (f) Evicting a segment when the average client access length decreases.

Adaptive and lazy segmentation

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 Figure 1 shows, because of our aggressive admission policy, each object is fully cached when our system accesses it the first time. The system keeps the fully cached object in the proxy until the replacement policy chooses it as an eviction victim.

During this process, we can use the lazy segmentation strategy to segment the object and evict some segments according to the first phase of the two-phase iterative replacement policy. From then on, our aggressive admission and replacement policies can admit and replace object segments as needed.

Aggressive admission policy

Our system activates cache admission each time the object is accessed. The system fully admits the object if the object is accessed for the first time (see Figure 1a). The system fully caches the object for two reasons: it's unknown whether the object will be popular, and if it is popular, the portion that clients will access most is unknown.

After the first access, the system keeps track of

the average client access length by recording necessary information. Later, the system can segment the object with respect to the user access pattern. From that point, 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 we show in Figure 1e.

Lazy segmentation strategy

In current segmentation strategies, the system performs the segmentation with a predetermined base segment length before accessing an object for the first time, such as the exponential segmentation strategy. By contrast, our adaptive and lazy segmentation strategy segments the object as late as possible: when the system selects a victim to make room for some incoming objects, it waits before segmenting the selected victim.

As Figure 1b shows, the proxy uses the average client access length computed at that moment as the unit for segmentation—that is, the base segment length of this object. The system then segments the whole object uniformly according to its base segment length, as we illustrate in Figure 1c. Because 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 online users' access behaviors.

Two-phase iterative replacement policy

Since the proxy has a limited cache space, the replacement is inevitable. Figures 1d and 1f show two phases of the replacement policy. The amount of replaced data at these two phases is different. A crucial aspect of the replacement policy lies in selecting a victim. The more appropriate the selected victim, the higher benefit the caching system gains. Instead of adopting a least recently used (LRU) algorithm to select the least recently used object as a victim (exponential segmentation uses this approach), our system considers the following factors:

- average number of accesses;
- average duration of accesses;
- length of the cached data, in terms of storage costs; and
- predicted probability of future accesses.

We can track the first three items based on client accesses, whereas the last one depends on predicting future accesses. However, the last one plays a more important role, because it predicts the trend of the object's popularity variations.

We take the following approach to come up with the prediction: When the object is first accessed, we denote the time as T_1. The current time is T_c, and the access time of the last (or most recent) access is T_r. The number of accesses so far is n.

Thus, the system computes the $T_c - T_r$, the time interval between now and the most recent access, and $(T_c - T_1)/n$, the average time interval for an access occurring in the past. If $T_c - T_r \geq (T_c - T_1)/n$, the possibility that a new request arrives soon for this object is small. Otherwise, it's 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 the average number of accesses, average duration of accesses, and predicted probability of future accesses. It's inversely proportional to the length of the cached data.

According to this utility function, the system always chooses the object with the smallest utility value as the victim. If the selected object turns out to be entirely cached, the system activates the first phase of the replacement policy. After completing the segmentation, the system keeps the first two segments in the cache, while it evicts all other segments. Figure 1d depicts this step.

The system keeps the first two segments because caching these two segments covers most client accesses that follow a normal distribution. If the selected victim is already partially cached, then the system activates the second phase of the replacement policy. As Figure 1f shows, it always evicts the last cached segment of the selected victim, and the system iteratively performs replacement until it finds sufficient cache space.

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

Typical performance in simulations

To test our design's performance, we used three synthetic workloads and one actual workload extracted from Hewlett-Packard (HP) Media Server

logs. The general trends reflected on these workloads are similar but the detailed variations depend on the individual workload. For brevity, we only present the results based on the real workload.

The actual workload of HP Corporate Media Solutions covers the period from 1–10 April 2001. It includes access to a total of 403 objects, and the unique object size amounts to 20 Gbytes. There were 9,000 requests. Our trace analysis revealed that 83 percent of the requests only view the objects for less than 10 minutes, and 56 percent only view the objects for less than 10 percent of their content. Only about 10 percent view the whole object. For synthetic workloads, we assume Poisson distribution for request interarrivals and Zipf-like distribution for object popularities.

Figures 2 and 3 show the performance results of the actual workload. We define the byte–hit ratio 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. For the uniform segmentation strategy, we used different segment sizes, and their results vary to 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. This proves the effectiveness of our proposed approach's adaptability.

When the cache size is in the range of 20 to 60 percent of the total object size, the performance improvement is much higher than when the cache size is larger than 70 percent of the total object size. The reason for this is because when the cache size reaches 70 percent, 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 isn't large enough, our proposed algorithm caches the segments according to their popularities. Other strategies try to cache the beginning segments of each object first, regardless of whether the object is popular, thus resulting in a lower cache performance (see Figure 2).

We can see similar performance trends when evaluating other synthetic workloads. Note that we generated the synthetic workloads based on previous research results on streaming media workload characterizations,[11] and our scheme's performance depends on the client access pattern. For workloads with the same object popularities and lengths, our scheme won't achieve a better performance.

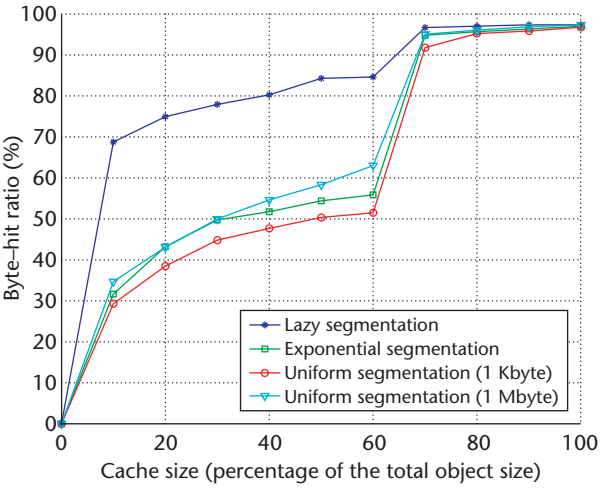


Figure 2 . Byte–hit ratio of an actual workload.

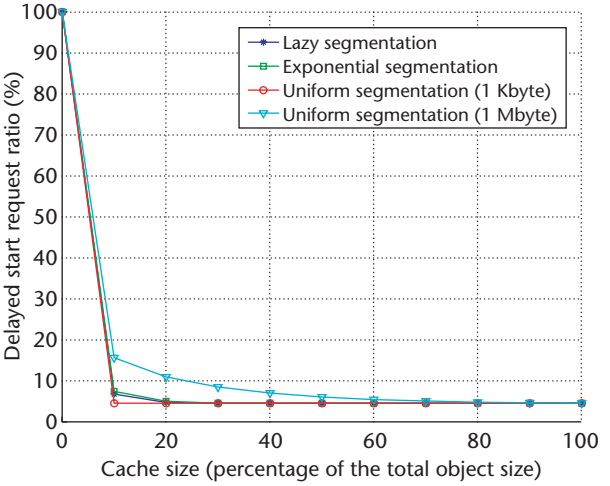


Figure 3. Delayed start–request ratio of an actual workload.

Figure 3 shows the performance of different schemes with respect to startup delay. We define a delayed start–request ratio 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 as well as the best existing scheme.

Active prefetching

Streaming media systems 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's

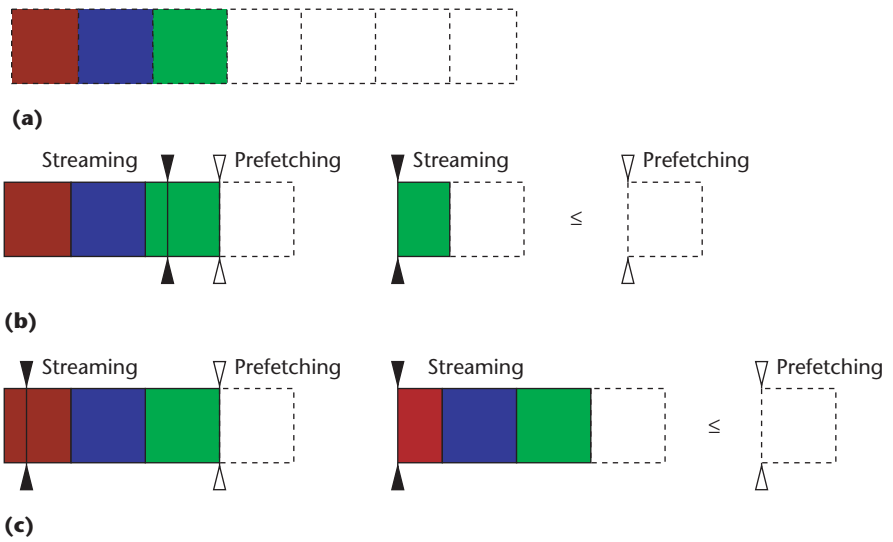


Figure 4. Example for segment prefetching. (a) Caching three object segments. (b) Assume that prefetching of the fourth segment begins as the client accesses the third segment. Prefetching the fourth segment must be faster than the streaming of the rest of the third segment plus the fourth segment. (c) The system needs to prefetch the segment earlier, when the client accesses the first segment.

point of view, but it's not the client's ultimate concern.

From the client's viewpoint, quality streaming is jitter free and has small startup latency. Thus, the client always expects the streaming delivery to start immediately after clicking the link. More importantly, during the delivery, the client expects playback to be continuous and without interruption.

On the other hand, these client-side metrics are tied to the system-side metrics, and better system resource usage might not result in better service quality to clients. For instance, if an object's beginning segments are already cached in the proxy when the client accesses the object, the client experiences no delay. However, to cache all the beginning portions of each object in the proxy requires a huge amount of cache space and leaves less space for caching popular segments, thus decreasing the cache performance.

Similarly, if an object's later segments aren't already cached when the client tries to access them, the delay—resulting in playback jitter at the client side—is inevitable if the network link bandwidth from the proxy to the server is insufficient. But to cache an object's later segments, the system 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.

Therefore, to provide a continuous streaming delivery to clients, we propose an *active prefetching scheme*. This scheme can prefetch the uncached segments from the content server in anticipation of the client's continuous access. In comparison to passive fetching, in which the fetching of uncached segments happens when the client misses it, our scheme's active prefetching proxy prefetches the uncached segments that are likely to be accessed based on the prediction of future client accesses. In this scheme, the system carefully balances the aforementioned conflicting objectives so that it removes proxy jitter without wasting network and proxy resources.

To conduct our prefetching scheme, we made the following assumptions:

- Streaming is faster than data transfer. Otherwise, prefetching is less critical.
- Media objects are already segmented and accessed sequentially.
- Bandwidth of the proxy–client link is large enough for the proxy to deliver the content to a client smoothly.
- A media server delivers each object segment in a unicast way.

Figure 4 shows a case study in prefetching. Figure 4a illustrates that at the beginning the system caches the first three object segments. When a client accesses the object, Figure 4b 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* prefetching. In this scenario, to guarantee a continuous stream, the prefetching of the entire fourth segment must be faster than the streaming of the third segment plus the fourth segment.

If the system always performs the prefetching one segment ahead, we can infer that the streaming speed can't be two times faster than the prefetching delay. Otherwise, a continuous

stream 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 4c shows such a scenario, in which the prefetching of the fourth segment should occur when the client begins accessing the first segment. Overall, based on the streaming speed and prefetching delay, the system can accurately calculate the starting point of the prefetch.

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

Correspondingly, we can figure out the temporary storage requirement for prefetching. Figure 5 shows how to precisely calculate the storage size. Figure 5a indicates the time when the system schedules prefetching to start, while Figure 5b illustrates when the actual streaming of the prefetched data begins. 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, it maximizes the resource wastage. The maximum wastage includes the network bandwidth for data transferring and the storage size for storing this amount of data.

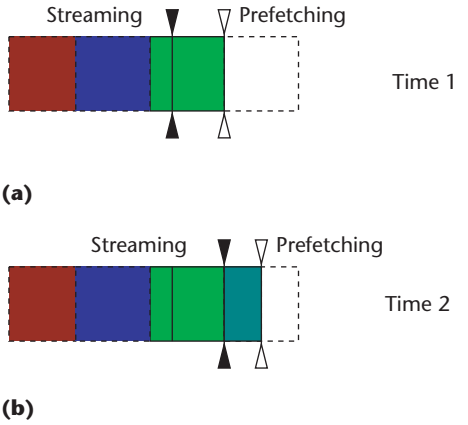


Figure 5. Storage size needed for prefetching. (a) A prefetching scenario when the prefetching starts. (b) Prefetching continues until the system begins streaming the prefetched data.

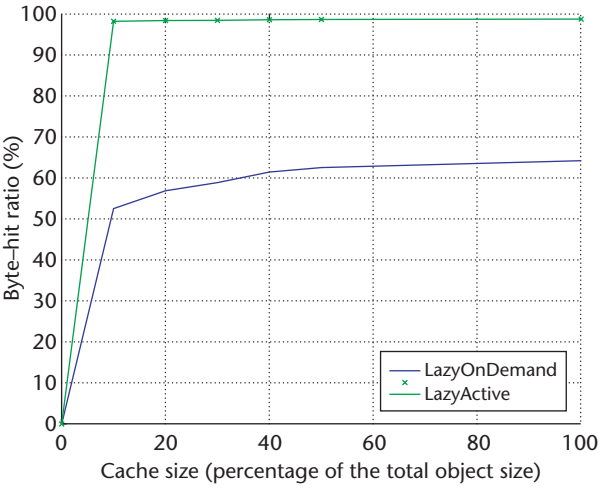


Figure 6. Byte–hit ratio from a real system.

Typical performance in real systems

We implemented the active prefetch system in our system. To evaluate the performance of active prefetch, we reproduced the three-month actual workload. Among the reproduced workload, we extracted a 12-hour trace to run in the deployed system. In the 12-hour workload, file sizes ranged from 1 to 100 minutes, and the object encoding rates include 28, 56, 112, 156, 180, and 256 Kbps. The unique object size amounted to 8.826 Gbytes.

We conducted the experiments by putting the system in a local network. Both the server and the proxy used a 2-GHz Pentium III with 1 Gbyte of memory. We show our experimental results 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 isn't skewed. Meanwhile, the performance gain by Lazy-Active with respect to Lazy-OnDemand shows the importance of prefetching. Clearly, with the assistance of active prefetching, the system can deliver a much larger percentage of uncached data to the clients in time, resulting in a higher byte–hit ratio.

The higher byte–hit ratio implies that the system can serve more data from the proxy to clients in time, resulting in the client experiencing less playback jitter. Active prefetching is effective because then the system uncaches segments at the right time, and considers the streaming rate and available bandwidth for prefetching.

Compared to other strategies, such as intuitive sliding-window-based prefetching, the scheduling for active prefetching needs some computational adjustments and the efficiency is much higher. Note that the computation of the starting point of prefetching only costs 0.1 percent of the total CPU cycles at the proxy in our experiments.

On the other hand, compared with the simulation results in Figure 2, the implemented sys-

tem achieves fewer gains (although these gains are more realistic). The reasons are a shorter trace and some tradeoffs made in the implementation. Because we extracted the real workload with the 12-hour trace from a workload containing a lot of premature client terminations (where the client only accesses a small portion of the object before terminating it), 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.

Current efforts

Today a client can use a PDA, cell phone, or other mobile device (instead of a high-end personal computer) to access the Internet. The wide use of mobile devices further complicates Internet streaming media delivery. Because of different screen sizes and color depths, a media object that's appropriate to a desktop computer might not be appropriate to a PDA. The media delivery network has to distinguish and adapt to different client devices. In accordance with the type of client device, networks will need to choose an appropriate version for streaming.

Moreover, using mobile devices induces the mobility problem. The media delivery network must consider the mobility of clients, as the client could frequently move from one place to another. Thus, we're trying to address the following questions:

- How can we convey different versions of a media object to different client devices?
- How can we deal with clients' mobility?

Proxy-enabled transcoding

To deal with the diversity of client browsing devices, we're considering 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 repeatedly transcoding operations.

The challenge here is that the media delivery network must be able to distinguish and adapt to different client devices. So far, two different ways exist 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. The drawback, though, is huge storage consumption for holding onto all of the object versions.

The second solution is *online transcoding*, where the network transcodes and delivers simultaneously. While the storage requirement is moderate, this approach burns a large amount of CPU cycles on the fly. Based on media segments, we're considering a hybrid solution that meets the diversity requirement of media delivery with reasonable overhead at a proxy server.

Proxy hand-off

Using 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 video-based news, the client might be on a train, or walking on the street. Thus, the media delivery network should provide a nomadic streaming service.

This implies one streaming proxy isn't capable of providing a continuous streaming service, given that each base station is associated with a proxy. To cope with the client's mobility, we resort to cooperative proxy caching techniques to reduce the number of expensive proxy hand-offs. However, coordinating different proxies for continuous media delivery is difficult because the hand-off between the base stations and the client can easily disrupt the continuity of streaming media delivery.

Numerous problems exist related to proxy-based streaming delivery, such as live streaming or power consumption in mobile devices. It's more difficult to exploit current proxy caching techniques for delivering live streams because the real-time requirements of live stream delivery are even more rigorous.

Conclusion

We've shown significant performance benefits of our segment-based caching methods and their effectiveness in practice for delivering streaming media over the Internet. We're currently making efforts to further improve the quality of Internet streaming in a cost-effective way. **MM**

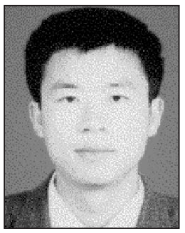
Acknowledgments

Our research activities are supported by the National Science Foundation and Hewlett-Packard Labs.

References

1. Y. Chae et al., "Silo, Rainbow, and Caching Token: Schemes for Scalable Fault Tolerant Stream

Caching," *IEEE J. Selected Areas in Comm.*, vol. 20, no. 7, 2002, pp. 1328-1344.
2. Z. Miao and A. Ortega, "Scalable Proxy Caching of Video under Storage Constraints," *IEEE J. Selected Areas in Comm.*, vol. 20, no. 7, 2002, pp. 1315-1327.
3. R. Rejaie et al., "Multimedia Proxy Caching Mechanism for Quality Adaptive Streaming Applications in the Internet," *Proc. IEEE Conf. Computer Comm.* (Infocom), IEEE CS Press, 2000, pp. 3-10.
4. S. Sen, J. Rexford, and D. Towsley, "Proxy Prefix Caching for Multimedia Streams," *Proc. IEEE Conf. Computer Comm.* (Infocom), IEEE CS Press, 1999, pp. 36-44.
5. K. Wu, P.S. Yu, and J. Wolf, "Segment-Based Proxy Caching of Multimedia Streams," *Proc. World Wide Web*, ACM Press, 2001, pp. 36-44.
6. R. Rejaie et al., "Proxy Caching Mechanism for Multimedia Playback Streams in the Internet," *Proc. Int'l Web Caching Workshop*, 1999.
7. B. Wang et al., "Proxy-Based Distribution of Streaming Video over Unicast/Multicast Connections," tech. report UM-CS-2001-005, Univ. of Massachusetts, 2001.
8. T. Kim and M.H. Ammar, "A Comparison of Layering and Stream Replication Video Multicast Schemes," *Proc. ACM Network and Operating System Support for Digital Audio and Video* (Nossdav 2001), ACM Press, 2001, pp. 63-72.
9. J. Jung, D. Lee, and K. Chon, "Proactive Web Caching with Cumulative Prefetching for Large Multimedia Data," *Proc. World Wide Web*, Elsevier, 2000.
10. J.I. Khan and Q. Tao, "Partial Prefetch for Faster Surfing in Composite Hypermedia," *Proc. 3rd Usenix Symp. Internet Technologies and Systems*, 2001, pp. 13-24.
11. M. Chesire et al., "Measurement and Analysis of a Streaming Media Workload," *Proc. 3rd Usenix Symp. Internet Technologies and Systems*, 2001, pp. 1-12.



Songqing Chen is an assistant professor in the Department of Computer Science at George Mason University. His research interests include operating systems and distributed systems. Chen received a PhD in computer science from the College of William and Mary.

Haining Wang is an assistant professor of computer science at the College of William and Mary. His research interests include networking, security, and distributed computing. Wang received a PhD in computer science and engineering from the University of Michigan at Ann Arbor.

Xiaodong Zhang is the Lettie Pate Evans Professor of Computer Science and the department chair at the College of William and Mary. His research interests include high-performance and distributed systems. Zhang received a PhD in computer science from the University of Colorado at Boulder. He is an associate editor for *IEEE Transactions on Computers* and *IEEE Micro* magazine.

Bo Shen is a senior research scientist with Hewlett-Packard Laboratories. His research interests include image/video processing and multimedia systems. Shen received a PhD in computer science from Wayne State University. He's a senior member of the IEEE.

Susie Wee manages the Multimedia Communications and Networking Department at Hewlett-Packard Laboratories and is a consulting assistant professor at Stanford University. Her research interests include multimedia networking and secure streaming. Wee received a PhD in electrical engineering from the Massachusetts Institute of Technology. She received the *Technology Review's* Top 100 Young Investigators award in 2002 and is associate editor of *IEEE Transactions on Image Processing*.

Readers may contact Songqing Chen at sqchen@cs.gmu.edu.