

# Segment-Based Streaming Media Proxy: Modeling and Optimization

Songqing Chen, *Member, IEEE*, Bo Shen, *Senior Member, IEEE*, Susie Wee, and Xiaodong Zhang, *Senior Member, IEEE*

**Abstract**—Researchers often use segment-based proxy caching strategies to deliver streaming media by partially caching media objects. The existing strategies mainly consider increasing the byte hit ratio and/or reducing the client perceived startup latency (denoted by the metric delayed startup ratio). However, these efforts do not guarantee continuous media delivery because the to-be-viewed object segments may not be cached in the proxy when they are demanded. The potential consequence is playback jitter at the client side due to proxy delay in fetching the uncached segments, which we call *proxy jitter*. Thus, for the best interests of clients, a correct model for streaming proxy system design should aim to minimize proxy jitter subject to reducing the delayed startup ratio and increasing the byte hit ratio. However, we have observed two major pairs of conflicting interests inherent in this model: (1) one between improving the byte hit ratio and reducing proxy jitter, and (2) the other between improving the byte hit ratio and reducing the delayed startup ratio. In this study, we first propose and analyze prefetching methods for in-time prefetching of uncached segments, which provides insights into the first pair of conflicting interests. Second, to address the second pair of the conflicting interests, we build a general model to analyze the performance tradeoff between the second pair of conflicting performance objectives. Finally, considering our main objective of minimizing proxy jitter and optimizing the two tradeoffs, we propose a new streaming proxy system called *Hyper Proxy*. Synthetic and real workloads are used to evaluate our system. The performance results show that *Hyper Proxy* generates minimum proxy jitter with a low delayed startup ratio and a small decrease of byte hit ratio compared with existing schemes.

**Index Terms**—Content distribution, proxy caching, streaming media, system design.

## I. INTRODUCTION

**P**ROXY caching has been widely used to cache static objects (text/images) on the Internet so that subsequent requests to the same objects can be served directly from the proxy without contacting the server. However, the proliferation of multimedia

content makes caching challenging (e.g., [1]–[8]) due to the typical large sizes and the low-latency and continuous streaming demands of media objects.

To solve the problems caused by large-sized media objects, researchers have developed a number of segment-based proxy caching strategies (e.g., [7], [9]–[13]) that cache partial segments of media objects instead of entire media objects. The existing segment-based proxy caching strategies can be classified into the following two types based on their performance objectives. The first type focuses on reducing the client perceived startup latency (denoted by the delayed startup ratio) by always giving a higher priority to caching the beginning segments of media objects based on the observation [14], [15] that clients tend to watch the beginning portions. The second type aims at reducing network traffic and server workload by improving proxy caching efficiency, namely the byte hit ratio (see Section II for more details).

However, these segment-based proxy caching strategies cannot automatically ensure continuous streaming delivery to the client. In a segment-based proxy caching system, since only partial segments of objects are cached in the proxy, it is important for the proxy to fetch and relay the uncached segments in time whenever necessary. A delayed fetch of the uncached segments, which we call *proxy jitter*, causes the discontinuous delivery of media content. Proxy jitter aggregates onto the playback jitter at the client side. Once a playback starts, jitter is not only annoying but can also potentially drive the user away from accessing the content. Thus, for the best interest of clients, the highest priority must be given to minimize proxy jitter, and a correct model for media proxy cache design should aim to minimize proxy jitter subject to reducing the delayed startup ratio and increasing the byte hit ratio.

To reduce proxy jitter, one key is to develop prefetching schemes that can prefetch uncached segments in a timely manner. Some early work has studied the prefetching of multimedia objects (e.g., [10], [11], [16], [17]). Unfortunately, little prefetching work has been found to efficiently solve the proxy jitter problem in the context of segment-based proxy caching.

Improving the byte hit ratio increases proxy caching efficiency, while reducing proxy jitter provides clients with a continuous streaming service. Unfortunately, these two objectives conflict with each other. Furthermore, we have also observed that improving the byte hit ratio conflicts with reducing the delayed startup ratio [9]. These three conflicting objectives form two pairs of tradeoffs that complicate the design model. No previous work has been found to address the

Manuscript received February 27, 2005; revised October 3, 2005. This work was supported by NSF Grants CNS-0098055, CNS-0405909, and CNS-0509054/0509061, and a grant from Hewlett-Packard Laboratories. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Qing Li.

S. Chen is with the Department of Computer Science, George Mason University, Fairfax, VA 22030 USA (e-mail: sqchen@cs.gmu.edu).

B. Shen and S. Wee are with the Mobile and Media System Laboratory, Hewlett-Packard Laboratories, Palo Alto, CA 94304 USA (e-mail: boshen@hpl.hp.com; swee@hpl.hp.com).

X. Zhang is with the Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210 USA (e-mail: zhang@cse.ohio-state.edu).

Digital Object Identifier 10.1109/TMM.2005.864281

balancing of these tradeoffs, which are uniquely important to streaming media proxy systems.

In this study, we first propose an active prefetching method for the in-time prefetching of uncached segments, which not only gives an effective solution to address the proxy jitter problem, but also provides insights into the tradeoff between improving the byte hit ratio and reducing proxy jitter. Second, to effectively address the conflicting interests between reducing startup latency and improving byte hit ratio, we build a general model to analyze the performance tradeoff between the second pair of conflicting performance objectives and find an effective approach to balance them. Finally, considering our main objective of minimizing proxy jitter and balancing the two tradeoffs, we propose a new streaming proxy system called *Hyper Proxy* [18]. Synthetic and real workloads are used to systematically evaluate the system. The performance results show that the Hyper Proxy system generates minimum proxy jitter with a low delayed startup ratio and a small decrease of byte hit ratio compared with existing schemes.

The paper is organized as follows. Some related work is introduced in Section II. We propose active prefetching and provide insights into proxy jitter in Section III. The second pair of conflicting interests is addressed in Section IV. The modeling and analysis results are summarized in Section V to motivate the Hyper Proxy system design in Section VI. We evaluate it in Section VII and we make concluding remarks in Section VIII.

## II. RELATED WORK

Recently, proxy caching of streaming media has been explored in a number of settings. Researchers have observed that most clients tend to watch the initial portions of media objects, and make fewer accesses to later portions [14], [15]. Segment-based proxy caching strategies have been developed based on this observation. Segment-based caching strategies cache media objects in segments instead of in full. Among these schemes, prefix caching [7] was proposed earlier to segment the media object as a prefix segment and a suffix segment. Protocol consideration as well as partial sequence caching are studied in [2], [19]. More recently, two types of segmentation strategies had been developed according to how the object is divided. The first uses uniform sized segments. For example, Rejaie *et al.* [10] consider the caching of fixed sized segments of layer-encoded video objects. Adaptive-lazy segmentation strategies are studied in [9], [20], in which each object is segmented as late as possible with a uniform segment length that is determined according to the client access pattern. A uniform segment length is used for each object, while different objects may have a different segment length. The second type uses exponentially sized segments. In this strategy, media objects are segmented such that the size of a segment doubles that of its preceding one [12]. The intuition behind this strategy is that later segments of media objects are less likely to be accessed than earlier segments. From the perspective of optimization goals, these strategies emphasize either reducing the startup latency perceived by clients [7], [21], or improving caching efficiency [9], [10], [12], [13].

In video staging [8], a portion of bits from the video frames whose size is larger than a predetermined threshold is cut off and

prefetched to the proxy to reduce the bandwidth on the server proxy channel. In [6], [10], and [11], a similar idea is proposed for caching scalable video in co-operation with the congestion control mechanism. The prefetching of uncached layered video is done by always maintaining a prefetching window of the cached stream, and identifying and prefetching all the missing data within the prefetching window with a fixed time period ahead of their playback time. In [17], the proactive prefetch utilizes any partially fetched data due to the aborted connections to improve network bandwidth utilization. In [16], prefetching is used to prefetch a certain amount of data so that caching is feasible. In [22], the proposed approach attempts to select groups of consecutive frames with a selective caching algorithm, while in [4], the algorithm may select groups of nonconsecutive frames for caching in the proxy. The caching problem for layered encoded video is studied in [3]. Cache replacement of streaming media is studied in [5], [23].

## III. PREFETCHING AND INSIGHTS INTO PROXY JITTER

Prefetching schemes can reduce proxy jitter by fetching uncached segments before they are accessed. However, an efficient prefetching method should consider the following two conflicting interests in the proxy. On the one hand, proxy jitter occurs if the prefetching of uncached segments is delayed. To avoid jitter, the proxy should prefetch uncached segments as early as possible. But aggressive prefetching of uncached segments requires extra network traffic and storage space to temporarily store the prefetched data. Even worse, the client session may terminate before the prefetched segments are accessed. This observation indicates that the proxy should prefetch uncached segments as late as possible. This contradiction requires that the proxy accurately decides when to prefetch which uncached segment in a way that minimizes proxy jitter and resource usage (network and storage). In this section, we propose an *active prefetching* method, which jointly consider both objectives. Our subsequent analysis provides insights into the conflicting interests between reducing proxy jitter and improving the byte hit ratio.

### A. Active Prefetching

The objective of our active prefetching method is to determine when to prefetch which uncached segment so that the proxy jitter is minimized with the minimum amount of resource requirement. The following assumptions are made in our analysis.

- The object has been segmented and is accessed sequentially;
- The bandwidth of the proxy-client link is large enough for the proxy to stream the content to the client smoothly; and
- Each segment of the object can be fetched from the server (either the origin or a cooperative one) in a unicast channel.

Since the prefetching is per segment based, several related notations used in the discussion are listed in Table I.

Note that each media object has its inherent encoding rate, which is the playback rate. The rate is not a constant in variable bit rate video, but we use  $B_s$  to denote its average value.  $B_t$

TABLE I  
NOTATIONS FOR ACTIVE PREFETCHING

$B_s$	the average encoding rate of a certain object segment
$B_t$	the average network bandwidth of the proxy-server link
$k$	the total number of segments of the object
$n$	the number of cached segments of the object
$S_i$	the $i^{th}$ segment of the object
$L_i$	the length of the $i^{th}$ segment
$L_b$	the base segment length of the object, $L_b = L_1$

may vary dynamically when different segments are accessed. The proxy monitors  $B_t$  by keeping records of data transmission rate of the most recent prior session with the same server. The transmission rate is calculated by dividing the amount of transferred data by the transfer duration.

For a requested media object, assume there are  $n$  segments cached in the proxy. The goal is to determine when to schedule the prefetching of segment  $S_{n+1}$  so that proxy jitter is avoided. We denote the scheduling point as  $x$ .

Note that prefetching is not necessary when  $B_s \leq B_t$ , so the following discussion is based on  $B_s > B_t$ . At position  $x$ , the length of the to-be-delivered data from the cache is  $\sum_{i=1}^{i=n} L_i - x$ . To avoid proxy jitter, the time that the proxy takes to prefetch  $S_{n+1}$  must not exceed the time that the proxy takes to delivery the rest of the cached data plus the fetched data. That is, the following condition must be satisfied to avoid proxy jitter:

$$\frac{\sum_{i=1}^{i=n} L_i - x + L_{n+1}}{B_s} \geq \frac{L_{n+1}}{B_t}.$$

Therefore, the latest prefetching scheduling point to avoid proxy jitter is

$$x = \sum_{i=1}^{i=n} L_i - \frac{L_{n+1} \times (B_s - B_t)}{B_t}. \quad (1)$$

Referring back to our objectives, when  $x$  is selected as the prefetching scheduling point, the buffer size required for the prefetched data reaches a minimum:

$$\frac{\sum_{i=1}^{i=n} L_i - x}{B_s} \times B_t. \quad (2)$$

We now discuss the active prefetching method for two typical segment-based caching schemes by first determining the prefetching scheduling point and then discussing the prefetching scheme and resource requirements.

1) *Active prefetching for uniformly segmented object*: For the uniformly segmented object,  $L_i = L_1$ , based on (1), we have the latest scheduling points  $x$  as

$$x = (n + 1)L_1 - \frac{B_s}{B_t}L_1. \quad (3)$$

Equation (3) states that if  $n + 1 \geq (B_s/B_t)$ , the in-time prefetching of  $S_{n+1}$  is possible with the minimum required buffer size of

$$L_1 \times \frac{B_s - B_t}{B_s}. \quad (4)$$

However, (3) also indicates that if  $n + 1 < (B_s/B_t)$ , the in-time prefetching of  $S_{n+1}$  is not possible. Therefore, when  $n + 1 < (B_s/B_t)$  and the segments between  $n + 1$ th and  $\lceil (B_s/B_t) \rceil$ th are demanded, the proxy jitter is inevitable. To minimize future proxy jitter under this situation, the proxy needs to prefetch the  $\lceil (B_s/B_t) \rceil$ th segment instead of the  $n + 1^{th}$  segment.

For uniformly segmented objects, the active prefetching works as follows.

- $n = 0$ : No segment is cached. The proxy jitter (in this case, startup delay) is inevitable. To avoid future proxy jitter, the prefetching of the  $\lceil (B_s/B_t) \rceil^{th}$  segment is necessary. The minimum buffer size required is  $(1 - (B_t/B_s))L_1$ .
- $n > 0$  and  $n + 1 < (B_s/B_t)$ : The proxy starts to prefetch the  $\lceil (B_s/B_t) \rceil$ th segment once the client starts to access the object. If the segment between  $n + 1$ th and  $\lceil (B_s/B_t) - 1 \rceil$ th are demanded, they are fetched on demand, and the proxy jitter is inevitable. The minimum buffer size required is  $(1 - (B_t/B_s))L_1$ .
- $n > 0$  and  $n + 1 \geq (B_s/B_t)$ : The prefetching of  $S_{n+1}$  is scheduled when the streaming reaches the position of  $(n + 1 - (B_s/B_t))L_1$  of the first  $n$  cached segments. The proxy jitter can be completely eliminated in this case, and the minimum buffer size required is  $(1 - (B_t/B_s))L_1$ .

2) *Active prefetching for exponentially segmented object*: Through similar analysis, the active prefetching for exponentially segmented objects works as follows. Here, we assume  $B_s \leq 2B_t$ . When  $B_s \geq 2B_t$ , no prefetching of the uncached segments can occur in time for the exponentially segmented objects.

- $n = 0$ : No segment is cached. Thus, proxy jitter (in this case, startup delay) is inevitable. To avoid future proxy jitter, the prefetching of the  $\lceil 1 + \log_2((1/2 - (B_s/B_t))) \rceil$ th segment is necessary once the client starts to access the object. The minimum buffer size required is  $L_1 \times (B_t^2/2 \times B_s \times B_t - B_s^2)$ .
- $n > 0$  and  $n \leq \log_2((1/2 - *B_s/B_t))$ : The proxy starts to prefetch the  $\lceil 1 + \log_2((1/2 - (B_s/B_t))) \rceil$ th segment once the client starts to access this object. The proxy jitter is inevitable when the client accesses data of the  $n + 1$ th segment to the  $\lceil 1 + \log_2((1/2 - (B_s/B_t))) \rceil$ th segment. The minimum buffer size is  $L_i B_t / B_s$ , where  $i = \lceil 1 + \log_2((1/2 - (B_s/B_t))) \rceil$ .
- $n > 0$  and  $n > \log_2((1/2 - (B_s/B_t)))$ : The prefetching of the  $n + 1$ th segment starts when the client accesses to the  $1 - (2^n/2^n - 1) \times ((B_s/B_t) - 1)$  portion of the first  $n$  cached segment. The minimum buffer size is  $L_{n+1} \times (B_t/B_s)$  and increases exponentially for later segments.

Our proposed active prefetching method gives the optimal prefetching scheduling point whenever possible with the minimum resource usage. However, under certain conditions the prefetching of uncached segments may still be delayed as our analysis showed, for both the uniformly and exponentially segmented objects. Furthermore, the analysis also finds that the

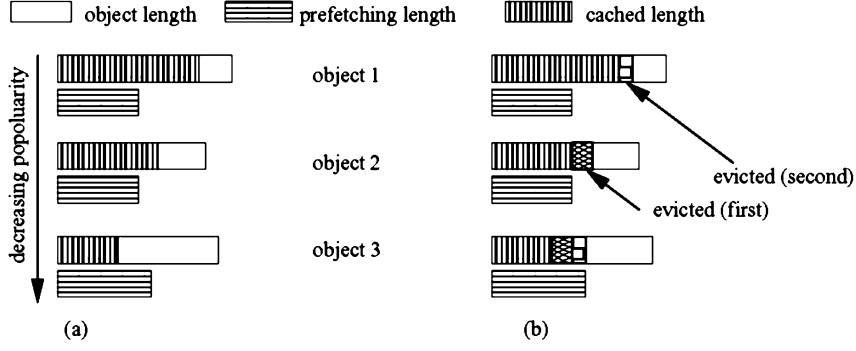


Fig. 1. Tradeoff between proxy jitter and byte hit ratio.

uniformly segmented object has advantages over the exponentially segmented object: it offers enhanced capability for in-time prefetching and the in-time prefetching can always begin in a later stage.

### B. Segment-Based Proxy Caching and Proxy Jitter Free Strategies

The previous section shows that active prefetching can not always guarantee continuous media delivery, which is one of the most important objectives for the streaming delivery. However, for any caching strategy, if there are always enough segments being cached in the proxy, prefetching of the uncached segments can always occur in time. To evaluate this situation, we define *prefetching length* as follows:

- *prefetching length*: the minimum length of data that must be cached in the proxy in order to guarantee continuous delivery when  $B_s > B_t$ . We denote  $m$  as the number of segments with the aggregated length equal to the prefetching length.

In-time prefetching must guarantee, in the worst case, that the prefetching of the rest of the segments is completed before the delivery of the whole object, that is

$$\frac{\sum_{i=1}^{i=k} L_i}{B_s} \geq \frac{\sum_{i=1}^{i=k} L_i - \sum_{i=1}^{i=m} L_i}{B_t}. \quad (5)$$

This indicates that the following condition must be satisfied to guarantee in-time prefetching:

$$\sum_{i=1}^{i=m} L_i \geq \sum_{i=1}^{i=k} L_i \left(1 - \frac{B_t}{B_s}\right). \quad (6)$$

- For the uniformly segmented object, since  $L_b$  is the base segment length, the minimum  $m$  to satisfy the above condition is

$$m = \left\lceil \frac{\left(1 - \frac{B_t}{B_s}\right) \sum_{i=1}^{i=k} L_i}{L_b} \right\rceil. \quad (7)$$

- For the exponentially segmented objects,<sup>1</sup> since  $L_i = 2L_{i-1}$ , the minimum  $m$  to satisfy the condition is

$$m = \left\lceil \log_2 \left( \frac{\left(1 - \frac{B_t}{B_s}\right) \sum_{i=1}^{i=k} L_i}{L_b} \right) \right\rceil + 1. \quad (8)$$

<sup>1</sup>To be consistent with the uniform segmentation case, the segments are counted from 1 instead of 0 as in [12].

### C. Tradeoff Between Low Proxy Jitter and High Byte Hit Ratio

We have calculated the minimum number of segments that must always be cached in the proxy to guarantee continuous delivery of the streaming media object. Thus we can estimate how much cache space we need to guarantee proxy-jitter-free delivery. However, in practice, we always have limited cache space and can not cache all these segments for each object.

In an actual segment-based proxy caching system, popular objects are always cached to reduce network traffic and server load. If an object is popular enough, all its segments can be cached in the proxy, possibly larger than its prefetching length. In Fig. 1(a), object 1 is in such a situation. If an object is not popular enough, some segments may get evicted and only a few of its segments are cached. The aggregated length of these segments may be less than its prefetching length, which causes proxy jitter when the uncached segments are demanded by the client. In Fig. 1(a), object 3 is in such a situation. Given a higher priority in reducing the proxy jitter, the proxy can choose to evict some segments of the object whose cached data length is larger than its prefetching length. The released cache space can be used to cache more segments of the object whose cached data length is less than its prefetching length so that the prefetching of its uncached segment can always be in time. In Fig. 1(b), we look for room from object 2 and then from object 1 since the spare cache space collected from object 2 is not enough. It is possible that segments of popular objects are evicted, which may reduce the byte hit ratio. However, since there are more objects with enough segments cached to avoid delayed prefetching, overall proxy jitter is reduced. From this example, we can see that the byte hit ratio can be traded for less proxy jitter.

## IV. BYTE HIT RATIO VERSUS DELAYED STARTUP RATIO

From previous study [9], we have observed that segment-based proxy caching strategies, typically the lazy-adaptive segmentation and exponential segmentation, always perform well in the byte hit ratio, but perform not so well in the delayed startup ratio, or vice versa. We present some details of our observation first. This observation leads us to conjecture that there are some conflicting interests between the objectives of improving the byte hit ratio and reducing the delayed startup ratio. We must understand these insights before we can design a correct system according to our design model.

TABLE II  
WORKLOAD SUMMARY

Workload Name	Num of Request	Num of Object	Size (GB)	$\lambda$	$\theta$	Range (minute)	Duration (day)
WEB	15188	400	51	4	0.47	2-120	1
PART	15188	400	51	4	0.47	2-120	1
REAL	9000	403	20	-	-	6 - 131	10

In this section, we formalize the problem and mathematically analyze this tradeoff after we review some performance results. An analytical model is built to analyze the two representatives: exponential segmentation and adaptive-lazy segmentation for the ideal situation where the replacement policy always finds the victim with the least utility value for being replaced. Thus the effect of other factors can be excluded so that we can understand the performance insights.

#### A. Representative Comparison Results

In the previous study, experiments have been conducted based on several workloads, including both synthetic workloads and a real workload extracted from enterprise media server logs. Two synthetic workloads assume a Zipf-like distribution ( $p_i = f_i / \sum_{i=1}^N f_i$ ,  $f_i = 1/i^\theta$ ) with a skew factor  $\theta$  for the popularity of the media objects and request inter arrival follows the Poisson distribution with a mean interval  $\lambda(p(x, \lambda) = e^{-\lambda} \lambda^x / x!, x = 0, 1, 2, \dots)$ .

The first synthetic workload, named WEB, simulates accesses to media object in the Web environment in which the length of the video varies from short ones to longer ones. The second simulates the Web accesses where clients accesses to objects are incomplete, that is, a started session terminates before the full media object is delivered. It simulates this behavior by designing a partial viewing workload based on the WEB workload, and thus called PART. In this workload, 80% of the sessions terminate before 20% of the object is delivered.

For the real workload named as REAL, it is extracted from server logs of HP Corporate Media Solutions, covering the period from April 1 through April 10, 2001. There is a total of 403 objects, and the unique object size accounts to 20 GB. There is a total of 9000 requests during this period. Table II lists some characteristics of these workloads.

By using a complete viewing workload WEB, when the cache size is 10%, 20%, and 30% of the total object size, the byte hit ratios achieved by the lazy segmentation and the exponential segmentation are about 50% and 13%, 67%, and 29%, 75%, and 39%, respectively (Note that in the following context, we always use lazy segmentation and exponential segmentation to represent their corresponding strategies.). The performance gap is more than 30% in average. At the same time, the delayed start request ratios achieved by the two strategies are about 44% and 12%, 34%, and 3%, 29%, and 2%, respectively. The gap is also about 30%.

Using a partial viewing workload PART, when the cache size is 10%, 20% and 30% of the total object size, the byte hit ratio achieved by the lazy segmentation is 28, 42, and 7 percentage points, respectively, more than those of the exponential segmentation. At the same time, the delayed start request ratio achieved by the lazy segmentation is 34, 24, and 13, percentage points,

TABLE III  
NOTATIONS FOR ANALYSIS

$n$	the number of objects
$p_i$	the $i$ th object popularity
$\theta$	skew factor for Zipf-like distribution
$\lambda$	access arrival rate for Poisson distribution
$L_i$	the full length of the $i$ th object
$L_{ave}$	the average length of the objects
$C$	the total cache size
$C_{prefix}$	the reserved space for caching startup length of objects
$C_{rest}$	$C_{rest} = C - C_{prefix} = (1 - \beta)C$
$\alpha$	the percentage of startup length of an object
$\beta$	the percentage of reserved space for caching startup length

respectively, more than those of the exponential segmentation. Similar trends are for another workload REAL.

#### B. Analytical Model

To formalize the problem, we assume the following.

- 1) The popularity of the objects follows a Zipf-like distribution, which models the probability set  $p_i$ , where  $p_i = (f_i / \sum_{i=1}^{i=n} f_i)$ , ( $i = 1, 2, \dots, n$ ,  $n$  is the total number of objects) and  $f_i = (1/i^\theta)$ , where  $\theta > 0$  and is the skew factor.
- 2) The request arrival interval process follows Poisson distribution with a mean arrival rate  $\lambda$ . The request arrival interval process to each individual object is independently sampled from the aggregate arrival interval process based on probability set  $p_i$ , where  $\sum_{i=1}^{i=n} p_i = 1$ .
- 3) The clients view the requested objects completely. This is to simplify the analysis and does not affect the conclusion.

These assumptions indicate that the mean arrival rate for each object is

$$\lambda_i = \lambda p_i = \lambda \times \frac{\frac{1}{i^\theta}}{\sum_{i=1}^{i=N} \frac{1}{i^\theta}}. \quad (9)$$

To evaluate the delayed startup ratio, we define the following notation:

- *startup length*: the length of the beginning part of an object. If this portion of the object is cached, no startup delay is perceived by clients when the object is accessed. We use  $\alpha$  to denote the percentage of the startup length with respect to the full object length<sup>2</sup>.

All notations used in the discussion are listed in Table III.

Consider the ideal case that the cache space is always allocated to cache the most popular objects. If we sort the objects

<sup>2</sup>Note that instead of caching the first  $\alpha$  percent, caching a constant length of the prefix segment for each object leads to the same results.

according to their decreasing popularities, the ideal case indicates that  $C_{\text{prefix}}$  is used to cache the segments (within startup length) of the first  $t$  most popular objects. Thus, ideally, for exponential segmentation, assuming the  $C_{\text{prefix}}$  can cache the first  $t$  objects' prefix segments,  $t$  must satisfy the following condition:

$$\sum_{i=1}^{i=t} L_i \times \alpha \leq C_{\text{prefix}} \quad \text{and} \quad \sum_{i=1}^{i=t+1} L_i \times \alpha > C_{\text{prefix}}. \quad (10)$$

Ideally, assuming the rest of the cache size  $C_{\text{rest}}$  can cache the first  $m$  object's remaining segments,  $m$  must satisfy the following condition:

$$\sum_{i=1}^{i=m} L_i \times (1 - \alpha) \leq C_{\text{rest}} \quad \text{and} \quad \sum_{i=m+1}^{i=n} L_i \times (1 - \alpha) > C_{\text{rest}}. \quad (11)$$

For lazy segmentation, no cache space is allocated separately to cache the initial segments of the object, thus, ideally, assuming the whole cache could be used to cache the first  $k$  objects according to the popularity,  $k$  must satisfy the following condition:

$$\sum_{i=1}^{i=k} L_i \leq C \quad \text{and} \quad \sum_{i=k+1}^{i=n} L_i > C. \quad (12)$$

To this end, we express the delayed start request ratios for exponential segmentation and lazy segmentation, as follows:

$$P_{\text{delay-E}} = \frac{\sum_{i=t+1}^{i=n} \lambda_i}{\sum_{i=1}^{i=n} \lambda_i} \quad (13)$$

and

$$P_{\text{delay-L}} = \frac{\sum_{i=k+1}^{i=n} \lambda_i}{\sum_{i=1}^{i=n} \lambda_i} \quad (14)$$

respectively.

Without considering the misses when the object is accessed for the first time, their corresponding byte hit ratios are

$$P_{\text{hit-E}} = 1 - \frac{\sum_{i=t+1}^{i=n} \lambda_i \times L_i \times \alpha + \sum_{i=m+1}^{i=n} \lambda_i \times L_i \times (1 - \alpha)}{\sum_{i=1}^{i=n} \lambda_i \times L_i} \quad (15)$$

and

$$P_{\text{hit-L}} = 1 - \frac{\sum_{i=k+1}^{i=n} \lambda_i \times L_i}{\sum_{i=1}^{i=n} \lambda_i \times L_i}. \quad (16)$$

respectively (based on that  $\text{hit\_ratio} + \text{miss\_ratio} = 1$ ).

### C. Performance Objective Analysis

In order to find the reasons for the unbalanced performance results as we observed, we analyze the performance objectives one by one based on the model we have built.

1) *Delayed Start Request Ratio*: Equations (13) and (14) indicate that the relationship between  $t$  and  $k$  determines which technique performs better in terms of the delayed start request ratio.

Based on (10) and (12), by comparing

$$\sum_{i=1}^{i=t} L_i \leq \frac{C \times \beta}{\alpha} \quad \text{and} \quad \sum_{i=1}^{i=k} L_i \leq C$$

we can get that if  $(\beta/\alpha) > 1$ , it will lead to  $t > k$ . Through (13) and (14),  $t > k$  means that the exponential segmentation has a better (less) delayed start request ratio. Otherwise,  $t \leq k$ , and lazy segmentation will perform better.

Exponential segmentation always caches beginning segments of all objects, which leads to  $k < t$ . Thus, in terms of the delayed start request ratio, exponential segmentation normally performs better than lazy segmentation.

2) *Byte Hit Ratio*: Based on (15) and (16), we can see the relationships among  $t$ ,  $m$ , and  $k$  are deterministic to the byte hit ratio, while their relationships are decided by  $\alpha$ ,  $\beta$ ,  $n$ , and  $L_i$ .

We now provide a thorough evaluation of all possible situations as follows.

- $m = t$ :

Equation (15) can be written as

$$P_{\text{hit-E}} = 1 - \frac{\sum_{i=t+1}^{i=n} \lambda_i \times L_i}{\sum_{i=1}^{i=n} \lambda_i \times L_i}. \quad (17)$$

Compared with (16), the problem once again comes to the relationship of  $t$  and  $k$ . If  $t > k$ , then exponential segmentation performs better. If  $t < k$ , lazy segmentation performs better.

When  $m = t$ , there are  $m$  or  $t$  objects cached by the exponential segmentation totally. Thus, we can get  $t = k$ . Thus, lazy segmentation will perform the same as exponential segmentation in terms of the byte hit ratio.

- $m < t$ :

Equation (15) can be written as

$$P_{\text{hit-E}} = 1 - \frac{\sum_{i=t+1}^{i=n} \lambda_i \times L_i + \sum_{i=m+1}^{i=t} \lambda_i \times L_i \times (1 - \alpha)}{\sum_{i=1}^{i=n} \lambda_i \times L_i}. \quad (18)$$

Equation (16) can be written as

$$P_{\text{hit-L}} = 1 - \frac{\sum_{i=k+1}^{i=t} \lambda_i \times L_i + \sum_{i=t+1}^{i=n} \lambda_i \times L_i}{\sum_{i=1}^{i=n} \lambda_i \times L_i}. \quad (19)$$

Thus, we must compare  $\sum_{i=m+1}^{i=t} \lambda_i \times L_i \times (1 - \alpha)$  and  $\sum_{i=k+1}^{i=t} \lambda_i \times L_i$ . If  $\sum_{i=m+1}^{i=t} \lambda_i \times L_i \times (1 - \alpha) > \sum_{i=k+1}^{i=t} \lambda_i \times L_i$ , lazy segmentation performs better. Otherwise, exponential segmentation performs better.

When  $m < t$ , there are  $m$  objects fully cached for exponential segmentation. Thus, for lazy segmentation, there are more objects fully cached and we can get that  $k > m$ . Therefore, we further analyze the case when  $k > m$  and  $t > m$  as follows.

Since

$$\begin{aligned} & \sum_{i=m+1}^{i=t} \lambda_i \times L_i \times (1 - \alpha) \\ &= \left( \sum_{i=m+1}^{i=k} \lambda_i \times L_i + \sum_{i=k+1}^{i=t} \lambda_i \times L_i \right) \times (1 - \alpha) \end{aligned} \quad (20)$$

and

$$\begin{aligned} & \sum_{i=k+1}^t \lambda_i \times L_i \\ &= \sum_{i=k+1}^t \lambda_i \times L_i \times \alpha + \sum_{i=k+1}^t \lambda_i \times L_i \times (1 - \alpha), \end{aligned} \quad (21)$$

we must compare  $\sum_{i=m+1}^{i=k} \lambda_i \times L_i \times (1 - \alpha)$  and  $\sum_{i=k+1}^t \lambda_i \times L_i \times \alpha$ .

From (10)–(12), we have

$$t = \frac{C}{L_{\text{ave}}^t} \times \frac{\beta}{\alpha}, m = \frac{C}{L_{\text{ave}}^m} \times \frac{1 - \beta}{1 - \alpha}, k = \frac{C}{L_{\text{ave}}^k} \quad (22)$$

where  $L_{\text{ave}}^t$  denotes the average value of  $L_1$  to  $L_t$ ,  $L_{\text{ave}}^m$  denotes the average value of  $L_1$  to  $L_m$ , and  $L_{\text{ave}}^k$  denotes the average value of  $L_1$  to  $L_k$ .

Assume objects are of equal length, then  $L_{\text{ave}}^t = L_{\text{ave}}^m = L_{\text{ave}}^k = L_{\text{ave}}$ . Note that this assumption simplifies the condition. Thus

$$t = k \times \frac{\beta}{\alpha}, m = k \times \frac{1 - \beta}{1 - \alpha}.$$

Since  $\lambda_i \geq \lambda_{i+1}$ , we get

$$\begin{aligned} \sum_{i=m+1}^{i=k} \lambda_i \times L_i \times (1 - \alpha) &\geq \sum_{i=m+1}^{i=k} \lambda_{k+1} \times L_i \times (1 - \alpha) \\ &= \lambda_{k+1} k (\beta - \alpha) L_{\text{ave}} \end{aligned} \quad (23)$$

and

$$\begin{aligned} \sum_{i=k+1}^t \lambda_i \times L_i \times \alpha &\leq \sum_{i=k+1}^t \lambda_{k+1} \times L_i \times \alpha \\ &= \lambda_{k+1} k (\beta - \alpha) L_{\text{ave}}. \end{aligned} \quad (24)$$

Based on (23) and (24), it is clear that exponential segmentation performs worse in byte hit ratio when  $m < t$ . This confirms the performance comparisons in Section VII.

- $m > t$ :

Equation (15) can be written as

$$P_{\text{hit-E}} = 1 - \frac{\sum_{i=t+1}^{i=m} \lambda_i \times L_i \times \alpha + \sum_{i=m+1}^{i=n} \lambda_i \times L_i}{\sum_{i=1}^{i=n} \lambda_i \times L_i}. \quad (25)$$

Equation (16) can be written as:

$$P_{\text{hit-L}} = 1 - \frac{\sum_{i=k+1}^{i=m} \lambda_i \times L_i + \sum_{i=m+1}^{i=n} \lambda_i \times L_i}{\sum_{i=1}^{i=n} \lambda_i \times L_i}. \quad (26)$$

Thus, we must compare  $\sum_{i=t+1}^{i=m} \lambda_i \times L_i \times \alpha$  and  $\sum_{i=k+1}^m \lambda_i \times L_i$  (Note that if  $k + 1 > m$ ,  $\sum_{i=k+1}^m \lambda_i \times L_i$  is defined as  $-\sum_{i=m+1}^k \lambda_i \times L_i$ ). If  $\sum_{i=t+1}^{i=m} \lambda_i \times L_i \times \alpha > \sum_{i=k+1}^m \lambda_i \times L_i$ , exponential segmentation performs worse. Otherwise, lazy segmentation performs worse.

When  $m > t$ , there are only  $t$  objects that are fully cached for exponential segmentation, so it must be  $k > t$ . Thus, the further analysis will be done when  $m > t$  and  $k > t$  as follows.

Since

$$\begin{aligned} & \sum_{i=t+1}^{i=m} \lambda_i \times L_i \times \alpha \\ &= \sum_{i=t+1}^{i=k} \lambda_i \times L_i \times \alpha + \sum_{i=k+1}^{i=m} \lambda_i \times L_i \times \alpha \end{aligned} \quad (27)$$

and

$$\begin{aligned} & \sum_{i=k+1}^m \lambda_i \times L_i \\ &= \sum_{i=k+1}^m \lambda_i \times L_i \times \alpha + \sum_{i=k+1}^m \lambda_i \times L_i \times (1 - \alpha) \end{aligned} \quad (28)$$

it comes to compare  $\sum_{i=t+1}^{i=k} \lambda_i \times L_i \times \alpha$  and  $\sum_{i=k+1}^m \lambda_i \times L_i \times (1 - \alpha)$ .

Since  $\lambda_i \geq \lambda_{i+1}$ , with the same assumption as before that objects are of same length, it is easy to get

$$\begin{aligned} \sum_{i=t+1}^{i=k} \lambda_i \times L_i \times \alpha &\geq \sum_{i=t+1}^{i=k} \lambda_{k+1} \times L_i \times \alpha \\ &= (\alpha - \beta) k \lambda_{k+1} L_{\text{ave}} \end{aligned} \quad (29)$$

and

$$\begin{aligned} \sum_{i=k+1}^m \lambda_i \times L_i \times (1 - \alpha) &\leq \sum_{i=k+1}^m \lambda_{k+1} \times L_i \times (1 - \alpha) \\ &= (\alpha - \beta) k \lambda_{k+1} L_{\text{ave}}. \end{aligned} \quad (30)$$

Based on (29) and (30), we can get that when  $m > t$ , lazy segmentation performs worse in terms of byte hit ratio under these assumptions. However,  $m > t$  leads to  $k > t$ , recall the analysis conclusion in IV-C1, when  $k > t$ , lazy segmentation will perform better in terms of the delayed start request ratio. This shows that there is always a tradeoff between the byte hit ratio and the delayed startup ratio. In reality, exponential segmentation always caches all objects' beginning segments, thus,  $m > t$  is always true.

The results of the above analysis show that the performance of segment-based caching strategies is always a tradeoff between the byte hit ratio and the delayed start request ratio. They are affected by the relationships of  $t, m$ , which are determined by  $\alpha, \beta, n$ , and  $L_{ave}$ . For the lazy segmentation strategy, in the sense that we do not reserve a part of the cache space for caching the beginning segments of objects,  $\beta = 0$ ; however, if counting the cache space used for caching the beginning segments of objects, a dynamically changing nonzero  $\beta$  is used. For the prefix caching,  $\beta$  is set to 100%. Based on the analysis results, if  $m$  is decreased, the achieved byte hit ratio is reduced. However, the decrease of  $m$  leads to decrease of  $t$ , which results in a reduced delayed start request ratio. This seems to indicate that we can use the byte hit ratio to trade for delayed start request ratio. Whether this is true or not is critical to if we can find out an effective way to balance these two performance objectives. We will answer these questions heuristically after we derive the performance bounds for each performance objective.

#### D. Performance Bound Analysis

We have learned that these two performance objectives are always a tradeoff between each other. However, how much performance can be optimized is not answered yet. In this section, we will give performance bounds based on the model so that they can guide the performance optimization under certain conditions as our assumptions state. We also assume the objects are of equal length as before. That is,  $L_{ave}^t = L_{ave}^m = L_{ave}^k = L_{ave}$ .

1) *Delayed Start Request Ratio*: For the exponential segmentation strategy, substituting (9) in (13), we get

$$P_{\text{delay-E}} = \frac{\sum_{i=t+1}^{i=n} \lambda \times \frac{\eta \times \frac{1}{i^\theta}}{\sum_{i=1}^{i=n} \eta \times \frac{1}{i^\theta}}}{\sum_{i=1}^{i=n} \lambda \times \frac{\eta \times \frac{1}{i^\theta}}{\sum_{i=1}^{i=n} \eta \times \frac{1}{i^\theta}}} = \frac{\sum_{i=t+1}^{i=n} \frac{1}{i^\theta}}{\sum_{i=1}^{i=n} \frac{1}{i^\theta}}. \quad (31)$$

Carefully using the series theory and integration on (31):

- $\theta = 1$

$$P_{\text{delay-E}} = \frac{\sum_{i=t+1}^{i=n} \frac{1}{i}}{\sum_{i=1}^{i=n} \frac{1}{i}} \leq \frac{\int_t^n \frac{1}{i} di}{\int_1^{n+1} \frac{1}{i} di} = \frac{\ln n - \ln t}{\ln(n+1)}$$

and

$$P_{\text{delay-E}} = \frac{\sum_{i=t+1}^{i=n} \frac{1}{i}}{\sum_{i=1}^{i=n} \frac{1}{i}} \geq \frac{\int_{t+1}^{n+1} \frac{1}{i} di}{1 + \int_1^n \frac{1}{i} di} = \frac{\ln \frac{n+1}{t+1}}{1 + \ln n}.$$

Having  $t = (C/L_{ave}) \times (\beta/\alpha), U = (C/L_{ave})$ , we have

$$P_{\text{delay-E}}^{\text{Max}} = \frac{\ln n - \ln U \times \frac{\beta}{\alpha}}{\ln(n+1)} \quad (32)$$

and

$$P_{\text{delay-E}}^{\text{Min}} = \frac{\ln(n+1) - \ln(U \times \frac{\beta}{\alpha} + 1)}{1 + \ln n}. \quad (33)$$

For (32) and (33), the larger the value of  $\beta$ , the smaller the values of  $P_{\text{delay-E}}^{\text{Max}}$  and  $P_{\text{delay-E}}^{\text{Min}}$ , and the smaller the value of  $\beta$ , the larger the values of  $P_{\text{delay-E}}^{\text{Max}}$  and  $P_{\text{delay-E}}^{\text{Min}}$ .

- $\theta \neq 1$

$$P_{\text{delay-E}} = \frac{\sum_{i=t+1}^{i=n} \frac{1}{i^\theta}}{\sum_{i=1}^{i=n} \frac{1}{i^\theta}} \leq \frac{\int_t^n \frac{1}{i^\theta} di}{\int_1^{n+1} \frac{1}{i^\theta} di} = \frac{n^{1-\theta} - t^{1-\theta}}{(n+1)^{1-\theta} - 1},$$

and

$$P_{\text{delay-E}} = \frac{\sum_{i=t+1}^{i=n} \frac{1}{i^\theta}}{\sum_{i=1}^{i=n} \frac{1}{i^\theta}} \geq \frac{\int_{t+1}^{n+1} \frac{1}{i^\theta} di}{1 + \int_1^n \frac{1}{i^\theta} di} = \frac{(n+1)^{1-\theta} - (t+1)^{1-\theta}}{n^{1-\theta} - \theta}.$$

Having  $t = (C/L_{ave}) \times (\beta/\alpha), U = (C/L_{ave})$ , we have

$$P_{\text{delay-E}}^{\text{Max}} = \frac{n^{1-\theta} - \left(U \times \frac{\beta}{\alpha}\right)^{1-\theta}}{(n+1)^{1-\theta} - 1} \quad (34)$$

and

$$P_{\text{delay-E}}^{\text{Min}} = \frac{(n+1)^{1-\theta} - \left(U \times \frac{\beta}{\alpha} + 1\right)^{1-\theta}}{n^{1-\theta} - \theta}. \quad (35)$$

$t^{1-\theta}$  is an *increasing* function when  $0 < \theta < 1$ , and a *nonincreasing* function when  $\theta > 1$ . Thus, the larger the value of  $\beta$ , the smaller the values of  $P_{\text{delay-E}}^{\text{Max}}$  and  $P_{\text{delay-E}}^{\text{Min}}$ , and the smaller the value of  $\beta$ , the larger the values of  $P_{\text{delay-E}}^{\text{Max}}$  and  $P_{\text{delay-E}}^{\text{Min}}$ .

For the lazy segmentation strategy, substituting (9) in (14), we get

$$P_{\text{delay-L}} = \frac{\sum_{i=k+1}^{i=n} \lambda \times \frac{\eta \times \frac{1}{i^\theta}}{\sum_{i=1}^{i=n} \eta \times \frac{1}{i^\theta}}}{\sum_{i=1}^{i=n} \lambda \times \frac{\eta \times \frac{1}{i^\theta}}{\sum_{i=1}^{i=n} \eta \times \frac{1}{i^\theta}}} = \frac{\sum_{i=k+1}^{i=n} \frac{1}{i^\theta}}{\sum_{i=1}^{i=n} \frac{1}{i^\theta}}. \quad (36)$$

Carefully using the series theory and integration on (36):

- $\theta = 1$

$$P_{\text{delay-L}}^{\text{Max}} = \frac{\ln n - \ln U}{\ln(n+1)} \quad (37)$$

and

$$P_{\text{delay-L}}^{\text{Min}} = \frac{\ln(n+1) - \ln(U+1)}{1 + \ln n}. \quad (38)$$

- $\theta \neq 1$

$$P_{\text{delay-L}}^{\text{Max}} = \frac{n^{1-\theta} - (U)^{1-\theta}}{(n+1)^{1-\theta} - 1} \quad (39)$$

and

$$P_{\text{delay-L}}^{\text{Min}} = \frac{(n+1)^{1-\theta} - (U+1)^{1-\theta}}{n^{1-\theta} - \theta}. \quad (40)$$

Equations (32) and (34) and (37) and (39) give the upper bounds for the exponential segmentation and the lazy segmentation strategies, respectively, with different  $\theta$  conditions. Equations (33) and (35) and (38) and (40) give lower bounds for them in the ideal situation.

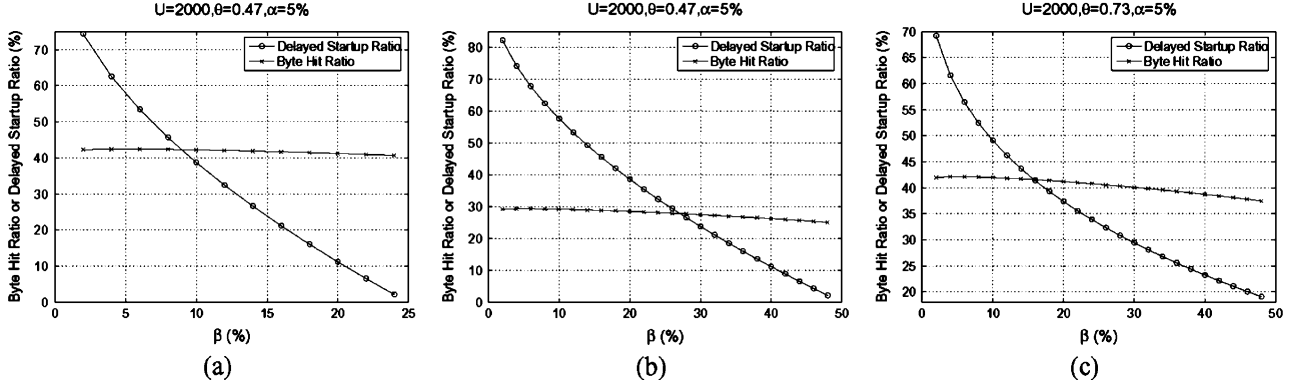


Fig. 2. Byte hit ratio versus delayed startup ratio.

2) *Byte Hit Ratio*: For exponential segmentation, based on (15), substituting the  $\lambda_i$  from (9), we get

$$P_{\text{hit-E}} = 1 - \frac{\alpha \times \sum_{i=t+1}^{i=n} \frac{1}{i^\theta} + (1-\alpha) \times \sum_{i=m+1}^{i=n} \frac{1}{i^\theta}}{\sum_{i=1}^{i=n} \frac{1}{i^\theta}}. \quad (41)$$

Carefully using the series theory and integration on (41):

- $\theta = 1$

$$P_{\text{hit-E}}^{\text{Max}} = 1 - \frac{\alpha \times \ln \frac{n+1}{U \times \frac{\beta}{\alpha} + 1} + (1-\alpha) \times \ln \frac{n+1}{U \times \frac{1-\beta}{1-\alpha} + 1}}{1 + \ln n} \quad (42)$$

and

$$P_{\text{hit-E}}^{\text{Min}} = 1 - \frac{\alpha \times \ln \frac{n}{U \times \frac{\beta}{\alpha}} + (1-\alpha) \times \ln \frac{n}{U \times \frac{1-\beta}{1-\alpha}}}{\ln(n+1)}. \quad (43)$$

- $\theta \neq 1$

$$P_{\text{hit-E}}^{\text{Max}} = 1 - \left( \frac{(n+1)^{1-\theta} - \alpha \times \left( \frac{\beta}{\alpha} \times U + 1 \right)^{1-\theta}}{n^{1-\theta} - \theta} - \frac{(1-\alpha) \times \left( \frac{(1-\beta)}{(1-\alpha)} \times U + 1 \right)^{1-\theta}}{n^{1-\theta} - \theta} \right) \quad (44)$$

and

$$P_{\text{hit-E}}^{\text{Min}} = 1 - \left( \frac{n^{1-\theta} - \alpha \times \left( \frac{\beta}{\alpha} \times U \right)^{1-\theta}}{(n+1)^{1-\theta} - 1} - \frac{(1-\alpha) \times \left( \frac{(1-\beta)}{(1-\alpha)} \times U \right)^{1-\theta}}{(n+1)^{1-\theta} - 1} \right). \quad (45)$$

For lazy segmentation, based on (16), substituting the  $\lambda_i$  from (9), we get

$$P_{\text{hit-L}} = 1 - \frac{\sum_{i=1}^{i=n} \frac{1}{i^\theta} - \sum_{i=1}^{i=k} \frac{1}{i^\theta}}{\sum_{i=1}^{i=n} \frac{1}{i^\theta}} = \frac{\sum_{i=1}^{i=k} \frac{1}{i^\theta}}{\sum_{i=1}^{i=n} \frac{1}{i^\theta}}. \quad (46)$$

Carefully using the series theory and integration on (46):

- $\theta = 1$

$$P_{\text{hit-L}}^{\text{Max}} = \frac{1 + \ln U}{\ln(n+1)} \quad (47)$$

and

$$P_{\text{hit-L}}^{\text{Min}} = \frac{\ln(U+1)}{1 + \ln n}. \quad (48)$$

- $\theta \neq 1$

$$P_{\text{hit-L}}^{\text{Max}} = \frac{U^{1-\theta} - \theta}{(n+1)^{1-\theta} - 1} \quad (49)$$

and

$$P_{\text{hit-L}}^{\text{Min}} = \frac{(U+1)^{1-\theta} - 1}{n^{1-\theta} - \theta}. \quad (50)$$

Equations (42), (44) (47), and (49) give the upper bounds for the exponential segmentation and the lazy segmentation strategies, respectively, with different  $\theta$  values. Equations (43), (45)(48), and (50) give the lower bounds for the ideal situation.

It is important to note that these upper and lower bounds are based on the model we built for ideal situations. Thus, the upper bounds we found here are valid upper bounds for general situations, while the lower bounds are only valid for ideal situations.

### E. Analytical Results

To give an intuition into the dynamic nature of the two performance objectives, Fig. 2 gives some examples based on (34) and (44). Here, in all three figures, we assume a cache size 20% of the total object size. This is because that existing experimental studies show that a cache size of 8%–16% is sufficient to achieve very good cache efficiency [24]. Furthermore,  $\theta$  and  $\alpha$  are set as 0.47 and 5%, respectively. For media objects, the popularity study in current research [14], [15] shows a  $\theta$  value of 0.47 for the Zipf-like distribution and is improving. A value of 0.73 is the typical results for Web content [25]. In this figures,  $U$  is set as 2000 object units.

Fig. 2(a) shows when the number of objects is 10 000. As shown in the figure, the decrease of the byte hit ratio is much slower than the decrease of the delayed startup ratio when  $\beta$  increases. Therefore, we can use a small decrease of byte hit ratio to trade for a significantly large reduction in the delayed startup ratio. The trend is clearer in Fig. 2(b), where the number

of objects is doubled. With the potentially more and more popular media objects, Fig. 2(c) shows the result when  $\theta$  is 0.73. As shown in the figure, the relative decreasing speed of the byte hit ratio and delayed startup ratio suggests that trading cache performance for improving client perceived startup latency is possible and beneficial.

Mathematically, the partial derivative of  $P_{\text{delay}}^{\text{Max}}$  with respect to  $\beta$  yields  $|\Delta_{\text{Delay}}|$  which denotes the change of the delayed startup ratio. The partial derivative of  $P_{\text{hit}}^{\text{Max}}$  with respect to  $\beta$  yields  $|\Delta_{\text{Hit}}|$  which denotes the change of the byte hit ratio. Therefore, we have

$$\frac{|\Delta_{\text{Delay}}|}{|\Delta_{\text{Hit}}|} = \frac{1}{\alpha} \times \frac{\frac{N^{1-\theta}-\theta}{(N+1)^{1-\theta}-1}}{\left(\frac{1-\beta}{1-\alpha} \times \frac{\alpha}{\beta} + \frac{\alpha}{U\beta}\right)^{-\theta} + \left(\frac{\alpha}{U\beta} + 1\right)^{-\theta}}. \quad (51)$$

It can be shown that  $|\Delta_{\text{Delay}}|/|\Delta_{\text{Hit}}|$  is always greater than 1 when  $\alpha$  and  $\beta$  are less than 50%.

The above analysis provides us with a solid basis to restructure the adaptive-lazy segmentation strategy in [9] by giving a higher priority to caching the startup length of objects in the replacement policy. The objective is to effectively address the conflicting interests between improving the byte hit ratio and reducing the delayed startup ratio for the best quality of media delivery. The analysis leads to the following improved replacement policy design.

#### F. Improved Adaptive-Lazy Segmentation Strategy

In order to significantly reduce the startup latency with a small decrease of the byte hit ratio as suggested by our previous analysis result, a three-phase iterative replacement policy is re-designed as follows.

Based on a utility function defined similarly as in [9], upon an object admission, if there is not enough cache space, the proxy selects the object with the smallest utility value at that time as the victim, and the segment of this object is evicted in one of the two phases as follows. 1) First Phase: If the object is fully cached, the object is segmented by the lazy segmentation method [9]. The first two segments are kept and the remaining segments are evicted right after the segmentation is completed. 2) Second Phase: If the object is partially cached with more than 1 segment, the last cached segment of this object is evicted. 3) Third Phase: If the victim has only the first segment and is to-be-replaced, then its startup length and the base segment length,  $L_b$ , is compared. If its startup length is less than the base segment length, the startup length is kept and the rest is replaced. Otherwise, it will be totally replaced. The utility value of the object is updated after each replacement and this process repeats iteratively until the required space is found. This restructured adaptive and lazy segmentation strategy has shown its effectiveness in [20] by well balancing the two performance objectives.

### V. PRINCIPLE TO DESIGN SEGMENT-BASED STREAMING MEDIA PROXY

In Sections III and IV, we have analyzed the two pairs of conflicting performance objectives in designing segment-based

TABLE IV  
HYPER PROXY DATA STRUCTURE

$T_1$	the time instance the object is firstly accessed
$T_r$	the last reference time of the object
$T_c$	the current time instance
$L_{sum}$	the sum of each access duration to the object
$n_a$	the number of accesses to the object
$L_b$	the length of the base segment
$n$	the number of the cached segments of the object
$FG_{adm}$	the admission flag for admitting segments
$L_{thd}$	the threshold length used in the replacement policy
$L_{avg}$	the average access duration of an object
$F$	the access frequency

streaming media proxy. The results show that to comprehensively consider the client performance and proxy caching efficiency, some less popular segments should be cached to minimize the proxy jitter. On the other hand, to reduce the client perceived startup latency, it is possible to slightly reduce the byte hit ratio to trade for a larger reduction of the client startup ratio. Having the answers to balance the two pairs of conflicting performance objectives in the previous sections, we can design a high quality media streaming proxy system.

Compared with the traditional Web caching studies, where the goal is always to improve the cache performance, thus reducing the network traffic and client response time, the caching of Internet media objects cannot simply follow such a principle. Our study indicates that the standard objective of improving the byte hit ratio commonly used in proxy caching is not suitable to streaming media delivery and this study provides an initial exploration on how to improve the Internet media delivery systems.

### VI. HYPER PROXY SYSTEM

In this section, we design a system, called Hyper Proxy, following our design model. The design consists of four components, which cooperate to implement the balance between the two pair of conflicting performance objectives. In our design, for any media object accessed through the proxy, a data structure containing the following items in Table IV is created and maintained. This data structure is called the access log of the object.

For each object, the  $L_{thd}$  is calculated after the segmentation (see Section VI-C). It is equal to  $\max(\text{startup length, prefetching length}, 2L_b)$  and its value varies due to the dynamic nature of  $B_s$  and  $B_t$ . In the system, two object lists (*premium list* and *basic list*) are maintained. The *basic list* contains all the objects whose length of cached segments is larger than its  $L_{thd}$  while the *premium list* contains all the objects whose cached data length is equal to or less than its  $L_{thd}$ .  $FG_{adm}$  is the flag used to indicate the priority of new segment admission. Items  $L_{avg}$  and  $F$  can be derived from the items above. They are used as measurements of access activities to each object. At time instance  $T_c$ , the access frequency  $F$  is  $n_a/(T_r - T_1)$ , and the average access duration  $L_{avg}$  is  $L_{sum}/n_a$ . When an object is accessed for the first time, it is fully cached and linked to the *basic list* according to the admission policy. A fully cached object is kept in the cache until it is chosen as an eviction victim according to the replacement

policy. At that time, the object is segmented and some of its segments are evicted. The object is also transferred to the *premium list*. Once the object is accessed again, the proxy uses the active prefetching method to determine when to prefetch which uncached segment. Then the segments of the object are adaptively admitted by the admission policy or adaptively replaced by the replacement policy.

We now present the detailed description of four major modules in the Hyper Proxy caching system.

#### A. Priority-Based Admission Policy

For any media object, cache admission is considered whenever the object is accessed.

- A requested object with no access log indicates that the object is accessed for the first time. The object is then cached in full regardless of the request's accessing duration. The replacement policy (see Section VI-D) is activated if there is not sufficient space. The victim is selected from objects in the *basic list*, or *premium list* when the *basic list* is empty. In the *premium list*, objects with *PRIORITY* flag are searched if no object with *NON-PRIORITY* flag is in *premium list*. The fully cached object is linked to the *basic list* and an access log is created for the object and the recording of the access history begins. If an access log exists for the object (not the first access to the object), but the log indicates that the object is fully cached, the access log is updated. No other action is necessary.
- If an access log exists for the object, and its  $FG_{adm}$  is *PRIORITY* (see Section VI-B), the proxy considers the admission of the next uncached segment or segments determined by its *prefetching length*. Whether the segment(s) can be admitted or not depends on if the replacement policy can find a victim or not. Victim selection is limited to objects in the *basic list* or *premium list* with *NON-PRIORITY* flag if *basic list* is empty. Note that for this admission, the system does not need to compare the caching utility value of this object with the victim's, but only to find a victim with the smallest utility value.
- If an access log exists for the object, and its  $FG_{adm}$  is *NON-PRIORITY* (see Section VI-B), the next uncached segment is considered for admission only if  $L_{avg} \geq (n+1)L_b/L_{thd}$ . (Note  $L_{avg}$  is changing dynamically.) The inequality indicates that the average access duration is increasing to the extent that the cached  $n$  segments can not cover most of the requests while a total of  $n+1$  segments can. Whether the next uncached segment is eventually admitted or not depends on whether or not the replacement policy can find a victim whose caching utility is less than this object. The victim selection is limited to the *basic list* only.

After the admission, the object will be transferred to the *basic list* if it is in the *premium list* and its cached data length is larger than its  $L_{thd}$ .

In summary, using the priority-based admission, the object is fully admitted when it is accessed for the first time. Then the

admission of this object is considered segment by segment with the higher priority given to the admissions that are necessary for in-time prefetching.

#### B. Active Prefetching

After the object is segmented and some of its segments are replaced (see Section VI-D), the object becomes partially cached. Then, upon each subsequent access, active prefetching is activated to determine when to prefetch which segment once the object is accessed according to the following various conditions.

- $n = 0$ : No segment is cached. The prefetching of the  $\lceil (B_s/B_t) \rceil$ th segment is considered. The  $FG_{adm}$  of this object is set to be *PRIORITY*.
- $n > 0$  and  $n + 1 < (B_s/B_t)$ : The proxy starts to prefetch the  $\lceil (B_s/B_t) \rceil$ th segment once the client starts to access the object. If the segments between  $n + 1$ th and  $\lceil (B_s/B_t) - 1 \rceil$ th are demanded, proxy jitter is inevitable and the  $FG_{adm}$  of this object is set to be *PRIORITY*.
- $n > 0$  and  $n + 1 \geq (B_s/B_t)$ : The prefetching of  $n + 1$ th segment starts when the client accesses to the position of  $(n + 1 - (B_s/B_t))L_b$  of the first  $n$  cached segments. The  $FG_{adm}$  of this object is set to be *NON-PRIORITY*.

Note that  $B_s$  and  $B_t$  are sampled when each segment is accessed. As a result,  $L_{thd}$  is also updated accordingly.

#### C. Lazy Segmentation Policy

The key of the lazy segmentation strategy is as follows. Once there is no cache space available and replacement is needed, the replacement policy calculates the caching utility of each cached object (see Section VI-D). Subsequently, the object with the smallest utility value is chosen as the victim if it is not active (no request is accessing it). If the victim object turns out to be fully cached, the proxy segments the object as follows. The average access duration  $L_{avg}$  at current time instance is calculated. It is used as the length of the base segment, that is,  $L_b = L_{avg}$ . Note that the value of  $L_b$  is fixed once it is determined. The object is then segmented uniformly according to  $L_b$ . After that, the first  $\lceil (L_{thd}/L_b) \rceil$  segments are kept in cache, while the rest are evicted (see Section VI-D). The number of cached segments,  $n$ , is updated in the access log of the object accordingly. This lazy segmentation scheme allows better determination of  $L_b$ .

#### D. Differentiated Replacement Policy

The replacement policy is used to re-collect cache space by evicting selected victims. First of all, a utility function (same as in [9]) is defined below to help the victim selection process by identifying the least valuable object as the victim:

$$F \times \frac{L_{sum}}{n_a} \times \min \left( 1, \frac{T_r - T_1}{T_c - T_r} \right) \quad (52)$$

In the above equation, the caching utility value is proportional to

- 1)  $F$ , which estimates the average number of future accesses;
- 2)  $(L_{sum}/n_a)$ , which estimates the average duration of future access;

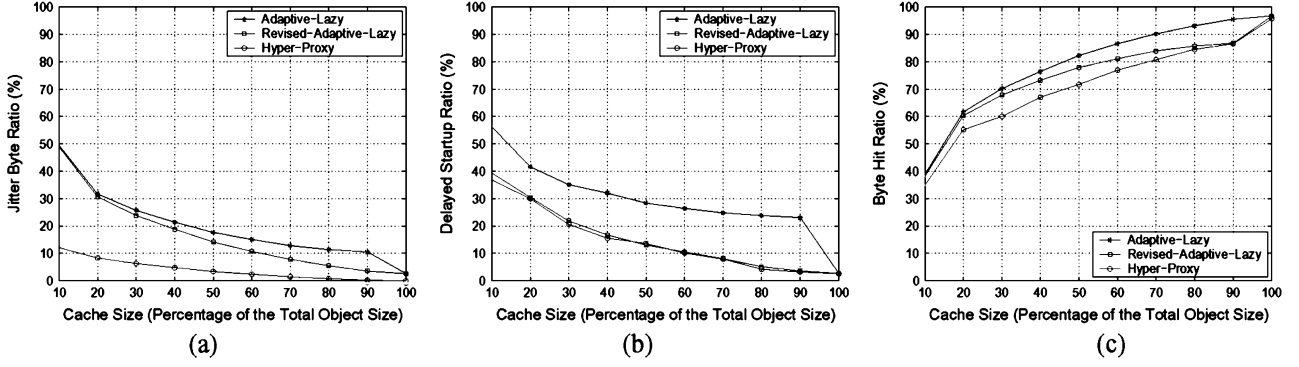


Fig. 3. Performance of the WEB workload: (a) Proxy jitter bytes, (b) delayed startup ratio, and (c) byte hit ratio.

- 3)  $\min(1, ((T_r - T_1/n_a)/T_c - T_r))$ , which estimates the possibility of future accesses<sup>3</sup>; and inversely proportional to
- 4)  $nL_b$ , which represents the disk space required.

Corresponding to the different situations of admission, when there is not enough space, the replacement policy selects the victim with the smallest utility value from different lists in the order as designated in Section VI-A. Then partially cached data of the victim is evicted as follows.

- If the victim is fully cached in the *basic list*, the object is segmented as described in Section VI-C. The first  $\lceil (L_{\text{thd}}/L_b) \rceil$  segments are kept and the remaining segments are evicted right after the segmentation is completed. The object is removed from the *basic list* and linked to the *premium list*.
- If the victim is partially cached in the *basic list*, the last cached segment of this object is evicted. After the eviction, the object will be linked to the *premium list* if its cached data length is less than or equal to its  $L_{\text{thd}}$ . Note this object can be selected as victim again if no sufficient space is found in this round.
- If the victim is in the *premium list*, the last cached segment of this object is evicted. If no data of this object is cached, it is removed from the *premium list*.

The utility value of the object is updated after each replacement and this process repeats iteratively until the required space is found.

The design of the differentiated replacement policy gives a higher priority for reducing proxy jitter, reduces the erroneous decision of the replacement and gives fair chances 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. Note that after an object is fully evicted, the system still keeps its access log. If not, once the object is occasionally accessed again, it should be fully cached again. Since media objects tend to have diminishing popularities as the time goes on, if the system caches the object in full again, this results in an inefficient use of the cache space. Our design enhances the resource utilization by avoiding this kind of situation. By setting

<sup>3</sup>The system compares the  $T_c - T_r$ , the time interval between now and the most recent access, and the  $(T_r - T_1/n_a)$ , the average time interval between accesses occurring in the past. If  $T_c - T_r > (T_r - T_1/n_a)$ , the possibility that a new request arrives soon for this object is small. Otherwise, it is highly possible that a request is coming soon.

a large enough time-out threshold, the proxy deletes the access logs of unpopular objects eventually.

## VII. PERFORMANCE EVALUATION

To evaluate the performance of the Hyper Proxy system, we use the same workloads as listed in Table II. In this section, we present detailed performance results.

In the simulation experiments, the streaming rate of accessed objects is set randomly in the range from half to four times that of the link capacity between the proxy and the server. We use the *jitter byte ratio* to evaluate the quality of the continuous streaming service provided by the proxy system. It is defined as the amount of data that is not prefetched in time by the proxy normalized by the total bytes demanded by the streaming sessions. Delayed prefetching causes potential playback jitter at the client side. A good proxy system should have small jitter byte ratio. The second metric we use is the *delayed startup ratio*, which is the number of requests that are served with a startup latency normalized by the total number of requests. The last metric we use is the *byte hit ratio*, which is the amount of data delivered to the client from the proxy cache normalized by the total bytes the clients demand.

We evaluate these three metrics in three designs of a segment-based proxy caching system. The *Adaptive-Lazy* represents the adaptive-lazy segmentation based proxy caching system [9] with active prefetching. This scheme aims at improving the byte hit ratio. The *Revised-Adaptive-Lazy* represents the improved adaptive-lazy segmentation based proxy caching system with active prefetching. This scheme is designed to reduce the delayed startup ratio subjective to improving the byte hit ratio. The *Hyper-Proxy* represents our proposed *Hyper Proxy* system in this paper, aiming at minimizing proxy jitter subjective to minimizing the delayed startup ratio while maintaining a high byte hit ratio.

For the WEB workload, the object encoding rate ranges in the 28 Kbps–256 Kbps, while the available network bandwidth for its uncached segments prefetching is randomly selected in the range of half to twice of its encoding rate. Fig. 3(a) shows that *Hyper-Proxy* always provides the best continuous streaming service to the client while *Adaptive-Lazy* that aims at increasing byte hit ratio, performs worst. Specifically, when cache size is 20% of total object size, *Hyper-Proxy* reduces proxy jitter by more than 50%.

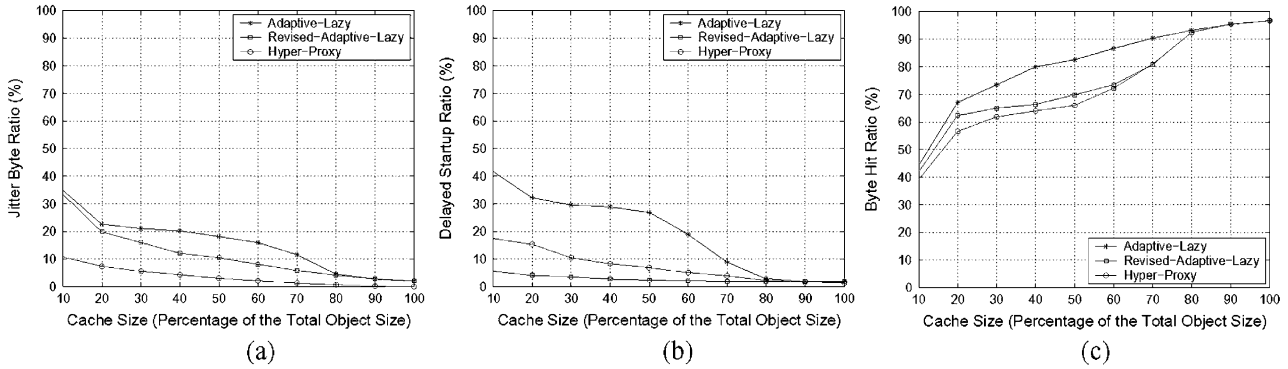


Fig. 4. Performance of the part workload: (a) jitter byte ratio, (b) delayed startup ratio, and (c) byte hit ratio.

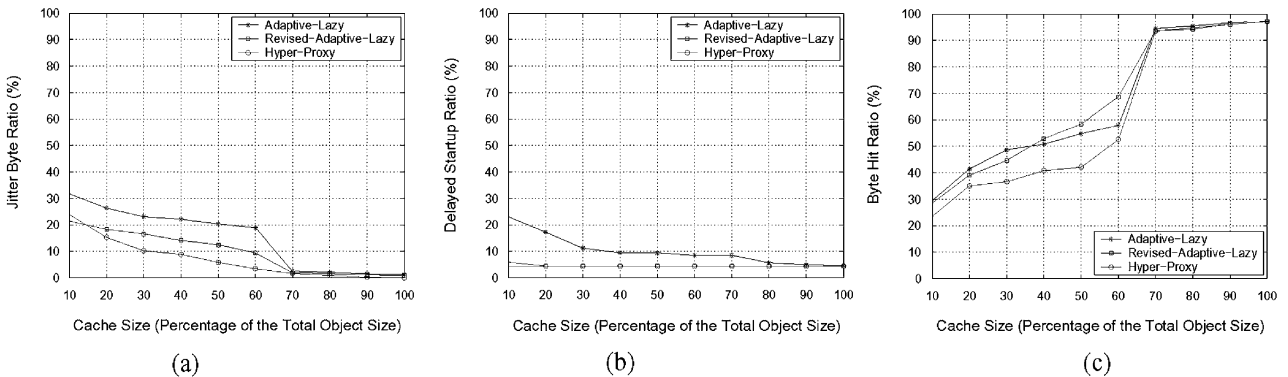


Fig. 5. Performance of the real workload: (a) jitter byte ratio, (b) delayed startup ratio, and (c) byte hit ratio.

Fig. 3(b) shows that *Hyper-Proxy* achieves the lowest delayed startup ratio. *Revised-Adaptive-Lazy* achieves results close to *Hyper-Proxy*. This is expected as we have analyzed in the [20].

Fig. 3(c) shows *Hyper-Proxy* achieves a relatively low byte hit ratio, which indicates a smaller reduction of network traffic. This is the price to pay for less proxy jitter and the smaller delayed startup ratio as shown in Fig. 3(a) and (b).

In PART, the object encoding rate and the available network bandwidth to prefetch its uncached segments are set as in WEB. Similar results are observed for the PART workload as shown in Fig. 4. When cache size is 20% of total object size, *Hyper-Proxy* reduces proxy jitter by 50% by giving up less than 5 percentage points in the byte hit ratio. Fig. 4(b) shows that *Revised-Adaptive-Lazy* achieves the best performance in reducing the delayed startup ratio. The result is expected since this scheme is specifically designed to prioritize reducing the delayed startup ratio. On the other hand, since *Hyper-Proxy* proactively prevents proxy jitter by keeping more segments, more cache space is used for segments that may not be requested due to early termination. This in turn makes *Hyper-Proxy* perform not well in reducing the delayed startup ratio.

In a more realistic setup, we use the REAL workload to evaluate performance. The encoding rate for an object in REAL is the same as recorded in the log, while we take the client connection link bandwidth as the available bandwidth for its uncached segment prefetching. As shown in Fig. 5, *Hyper-Proxy* performs best in reducing proxy jitter and delayed startup. The performance degradation in byte hit ratio is also acceptable. For this evaluation, it is interesting that the byte hit ratio achieved

by the *Adaptive-Lazy* system is not as high as without considering bandwidths. Studying different situations, we find that in our simulation the available bandwidth of the proxy-server link is typically much smaller than the object encoding rate, causing a large number of byte misses in *Adaptive-Lazy* due to request bursty, which however would not have happened without considering the bandwidth constraint.

## VIII. CONCLUSION

Proxy caching has been widely used for text-based content delivery over the Internet. With more and more Internet media contents, using proxies to support media delivery is cost-effective, but challenging due to the nature of media, including large media sizes and low-latency and continuous streaming demand. Most existing research on proxy-based media delivery has been conducted to address some of these challenges. However, there is no research that has comprehensively studied the proxy-based media delivery system. In this paper:

- we have presented an optimization model to guide the design of effective media proxy system and to ensure high delivery quality to clients, which aims to minimize proxy jitter subject to reducing the startup latency and increasing the byte hit ratio;
- we have provided insights into the model by analyzing two pairs of conflicting interests and tradeoffs inherent in this model;
- we have proposed to build a new media proxy caching system called Hyper Proxy. This system addresses the

interests from the perspectives of both clients and Internet resource management with a high priority given to clients. We have shown that the Hyper Proxy system minimizes proxy jitter with a low delayed startup ratio and low network traffic compared with other existing caching schemes.

#### ACKNOWLEDGMENT

The authors appreciate the critical and constructive comments from the anonymous referees.

#### REFERENCES

- [1] E. Bommaiah, K. Guo, M. Hofmann, and S. Paul, "Design and implementation of a caching system for streaming media over the Internet," in *Proc. IEEE Real Time Technology and Applications Symp.*, Washington, DC, May 2000, pp. 121.
- [2] M. Y. Chiu and K. H. Yeung, "Partial video sequence caching scheme for vod systems with heterogeneous clients," in *Proc. 13th Int. Conf. Data Engineering*, Birmingham, U.K., Apr. 1997.
- [3] J. Kangasharju, F. Hartanto, M. Reisslein, and K. W. Ross, "Distributing layered encoded video through caches," *IEEE Trans. Comput.*, vol. 51, no. 6, pp. 622–636, Jun. 2002.
- [4] Z. Miao and A. Ortega, "Scalable proxy caching of video under storage constraints," in *IEEE J. Sel. Areas Commun.*, vol. 20, Sep. 2002, pp. 1315–1327.
- [5] M. Reisslein, F. Hartanto, and K. W. Ross, "Interactive video streaming with proxy servers," in *Proc. First Int. Workshop on Intelligent Multimedia Computing and Networking*, Atlantic City, NJ, Feb. 2000.
- [6] R. Rejaie, M. Handley, and D. Estrin, "Quality adaptation for congestion controlled video playback over the internet," in *Proc. ACM SIGCOMM*, Cambridge, MA, Sep. 1999.
- [7] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proc. IEEE INFOCOM*, New York, Mar. 1999.
- [8] Z. Zhang, Y. Wang, D. Du, and D. Su, "Video staging: A proxy-server based approach to end-to-end video delivery over wide-area networks," in *IEEE Trans. Networking*, vol. 8, Aug. 2000, pp. 429–442.
- [9] S. Chen, B. Shen, S. Wee, and X. Zhang, "Adaptive and lazy segmentation based proxy caching for streaming media delivery," in *Proc. ACM NOSSDAV*, Monterey, CA, Jun. 2003.
- [10] R. Rejaie, M. Handley, H. Yu, and D. Estrin, "Proxy caching mechanism for multimedia playback streams in the internet," in *Proc. Int. Web Caching Workshop*, San Diego, CA, Mar. 1999.
- [11] R. Rejaie, M. H. H. Yu, and D. Estrin, "Multimedia proxy caching mechanism for quality adaptive streaming applications in the Internet," in *Proc. IEEE INFOCOM*, vol. 2, Israel, Mar. 2000, pp. 980–989.
- [12] K. Wu, P. S. Yu, and J. Wolf, "Segment-based proxy caching of multimedia streams," in *Proc. WWW*, Hong Kong, May 2001.
- [13] 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 J. Sel. Areas Commun.*, vol. 20, Sep. 2002, pp. 1328–1344.
- [14] L. Cherkasova and M. Gupta, "Characterizing locality, evolution, and life span of accesses in enterprise media server workloads," in *Proc. ACM NOSSDAV*, Miami, FL, May 2002.
- [15] M. Chesire, A. Wolman, G. Voelker, and H. Levy, "Measurement and analysis of a streaming media workload," in *Proc. 3rd USENIX Symp. Internet Technologies and Systems*, San Francisco, CA, Mar. 2001.
- [16] J. I. Khan and Q. Tao, "Partial prefetch for faster surfing in composite hypermedia," in *Proc. 3rd USENIX Symp. Internet Technologies and Systems*, San Francisco, CA, Mar. 2001.
- [17] J. Jung, D. Lee, and K. Chon, "Proactive web caching with cumulative prefetching for large multimedia data," in *Proc. WWW*, Amsterdam, Netherlands, May 2000.
- [18] S. Chen, B. Shen, S. Wee, and X. Zhang, "Designs of high quality streaming proxy systems," in *Proc. IEEE INFOCOM*, vol. 3, Hong Kong, Mar. 2004, pp. 1512–1521.
- [19] S. Gruber, J. Rexford, and A. Basso, "Protocol considerations for a prefix-caching for multimedia streams," in *Comput. Network*, vol. 33, Jun. 2000, pp. 657–668.
- [20] S. Chen, B. Shen, S. Wee, and X. Zhang, "Investigating performance insights of segment-based proxy caching of streaming media strategies," in *Proc. ACM/SPIE Conf. Multimedia Computing and Networking*, San Jose, CA, Jan. 2004.
- [21] B. Wang, S. Sen, M. Adler, and D. Towsley, "Proxy-based distribution of streaming video over unicast/multicast connections," in *Proc. IEEE INFOCOM*, New York City, Jun. 2002.
- [22] W. Ma and H. Du, "Reducing bandwidth requirement for delivering video over wide area networks with proxy server," in *Proc. Int. Conf. Multimedia and Expo.*, New York, Jul. 2000.
- [23] R. Tewari, A. D. H. Vin, and D. Sitaram, "Resource-based caching for web servers," in *Proc. ACM/SPIE Conf. Multimedia Computing and Networking*, San Jose, CA, Jan. 1998.
- [24] S. Roy, B. Shen, S. Chen, and X. Zhang, "An empirical study of a segment-based streaming proxy in an enterprise environment," in *Proc. 9th Int. Workshop on Web Content Caching and Distribution (WCW'04)*, Beijing, China, Oct. 2004.
- [25] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *Proc. IEEE INFOCOM*, vol. 1, New York, Mar. 1999, pp. 126–134.

**Songqing Chen** (M'03) received the Ph.D. degree in computer science from the College of William and Mary, Williamsburg, VA.

He is an Assistant Professor in the Department of Computer Science, George Mason University, Fairfax, VA. His research interests include high performance computing, operating systems, and distributed systems.

**Bo Shen** (M'97–SM'04) received the B.S. degree in computer science from Nanjing University of Aeronautics and Astronautics, China, and the Ph.D. degree in computer science from Wayne State University, Detroit MI.

He is a Senior Research Scientist in Streaming Media Systems Group at Hewlett-Packard Laboratories, Palo Alto, CA. His research interests include multimedia signal processing, multimedia networking, and content distribution systems. He has published over 40 papers in prestigious technical journals and conferences and holds four U.S. patents with many pending.

Dr. Shen has been on the program committee for a number of technical conferences, including SIGMM 2005.

**Susie Wee** received the Ph.D. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge.

She is currently the Lab Director of the Mobile and Media Systems Lab, Hewlett Packard Laboratories, Palo Alto, CA, and is a consulting Assistant Professor at Stanford University, Stanford, CA. Her research interests include multimedia networking and secure streaming.

Dr. Wee received the Technology Review's Top 100 Young Investigators award in 2002. She is currently an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY and formerly was an Associate Editor of the IEEE TRANSACTIONS ON IMAGE PROCESSING.

**Xiaodong Zhang** (SM'94) received the B.S. degree in electrical engineering from the Beijing Polytechnic University in 1982, and the Ph.D. degree in computer science from the University of Colorado, Boulder, in 1989.

He is the Robert M. Critchfield Professor in Engineering, and Chair of Department of Computer Science and Engineering at The Ohio State University, Columbus.

Dr. Zhang served as the Program Director of Advanced Computational Research at the National Science Foundation from 2001 to 2004. He is the Associate Editor-in-Chief of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, and is serving on the Editorial Boards of the IEEE TRANSACTIONS ON COMPUTERS and *IEEE Micro*.