Adaptive and Lazy Segmentation Based Proxy Caching for Streaming Media Delivery

Songqing Chen
Dept. of Computer Science
College of William and Mary
Williamsburg, VA 23187
sqchen@cs.wm.edu

Bo Shen
Mobile and Media System Lab
Hewlett-Packard Laboratories
Palo Alto, CA 94304
boshen@hpl.hp.com

Susie Wee
Mobile and Media System Lab
Hewlett-Packard Laboratories
Palo Alto, CA 94304
swee@hpl.hp.com

Xiaodong Zhang
Dept. of Computer Science
College of William and Mary
Williamsburg, VA 23187
zhang@cs.wm.edu

ABSTRACT

Streaming media objects are often cached in segments. Previous segment-based caching strategies cache segments with constant or exponentially increasing lengths and typically favor caching the beginning segments of media objects. However, these strategies typically do not consider the fact that most accesses are targeted toward a few popular objects. In this paper, we argue that neither the use of a predefined segment length nor the favorable caching of the beginning segments is the best caching strategy for reducing network traffic. We propose an adaptive and lazy segmentation based caching mechanism by delaying the segmentation as late as possible and determining the segment length based on the client access behaviors in real time. In addition, the admission and eviction of segments are carried out adaptively based on an accurate utility function. The proposed method is evaluated by simulations using traces including one from actual enterprise server logs. Simulation results indicate that our proposed method achieves a 30% reduction in network traffic. The utility functions of the replacement policy are also evaluated with different variations to show its accuracy.

Categories and Subject Descriptors

H.4.m [**Information System**]: Miscellaneous

General Terms

Algorithms, Experimentation

Keywords

Streaming media delivery, Lazy Segmentation, Proxy Caching

NOSSDAV'03, June 1–3, 2003, Monterey, California, USA.

1. INTRODUCTION

Proxy caching has been shown to reduce network traffic and improve client-perceived startup latency. However, the proliferation of multimedia content makes caching difficult. Due to the large sizes of typical multimedia objects, a full-object caching strategy quickly exhausts the cache space. Two techniques are typically used to overcome this problem, namely prefix caching and segment-based caching. Prefix caching [17] works well when most clients access the initial portions of media objects as noted in [4, 5]. It also reduces startup latency by immediately serving the cached prefix from the proxy to the client while retrieving subsequent segments from the origin server. In prefix caching, the determination of the prefix size plays a vital role in the system's performance.

Segment-based caching methods have been developed for increased flexibility. These methods also cache segments of media objects rather than entire media objects. Typically two types of segmentation strategies are used. The first type uses uniformly sized segments. For example, authors in [14] consider caching uniformly sized segments of layer-encoded video objects. The second type uses exponentially sized segments. In this strategy, media objects are segmented with increasing lengths; for example, the segment length may double [19]. This strategy is based on the assumption that later segments of media objects are less likely to be accessed. A combination of these methods can be found in [2], in which constant lengths and exponentially increased lengths are both considered. This type of method also favors the beginning segments of media objects.

The prefix and segmentation-based caching methods discussed above have greatly improved media caching performance. However, they do not address the following considerations: 1) Client accesses to media objects typically represent a skewed pattern: most accesses are for a few popular objects, and these objects are likely to be watched in their entirety or near entirety. This is often true for movie content in a VoD environment and training videos in a corporation environment. A heuristic segment-based caching strategy with a predefined segment size, exponential or uniform, always favorably caches the beginning segments of media objects and does not account for the fact that most accesses are targeted to a few popular objects. 2) The access characteristics of media objects are dynamically changing. The media object's popularity and most-watched portions may vary with time. For example, some objects

may be popular for an initial time period where most users access entire objects. Then, as the time goes on, there may be fewer requests for these objects and there may be fewer user accesses to the later portions of the objects. In this scenario, using a fixed strategy of caching several early segments may not work, since during the initial time period this may overload the network as later segments need to be retrieved frequently; then during the later time, caching all the initial segments may become wasteful of resources. The lack or poorness of adaptiveness in the existing proxy caching schemes may render proxy caching to be ineffective. 3) 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 them 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 cause increased management overhead.

In this paper, we propose an adaptive and lazy segmentation based caching strategy, which responsively adapts to the real time accesses and lazily segments objects as late as possible. Specifically, we design an aggressive admission policy, a lazy segmentation strategy, and a two-phase iterative replacement policy. The proxy system supported by the proposed caching strategy has the following advantages: 1) It achieves maximal network traffic reduction by favorably caching the popular segments of media objects, regardless of their positions within the media object. If most of the clients tend to watch the initial portions of these objects, the initial segments are cached. 2) It dynamically adapts to changes in object access patterns over time. Specifically, it performs well in common scenarios in which the popularity characteristics of media objects vary over time. The system automatically takes care of this situation without assuming *a priori* access pattern. 3) It adapts to different types of media objects. Media objects from different categories are treated fairly with the goal of maximizing caching efficiency.

Specifically, the adaptiveness of our proposed method falls into two areas. 1) The segment size of each object is decided adaptively based on the access history of this object recorded in real time. The segment size determined in this way more accurately reflects the client access behaviors. The access history is collected by delaying the segmentation process. 2) Segment admission and eviction policies are adapted in real time based on the access records. A utility function is derived to maximize the utilization of the cache space. Effectively, the cache space is favorably allocated to popular segments regardless of whether they are initial segments or not.

Both synthetic and real proxy traces are used to evaluate the performance of our proposed method. We show that (1) the uniform segmentation method achieves a similar performance result as the exponential segmentation method on average; (2) our proposed adaptive and lazy segmentation strategy outperforms the exponential and the uniform segmentation methods by about 30% in byte hit ratio on average, which represents a server workload and network traffic reduction of 30%.

The rest of the paper is organized as follows. The design of the adaptive and lazy segmentation based caching system is presented in Section 2. Performance evaluation is presented in Section 3 and further evaluation is presented in Section 4. We evaluate the utility function of the replacement policy in Section 5 and make concluding remarks in Section 6.

1.1 Related Work

Proxy caching of streaming media has been explored in [17, 6, 20, 10, 13, 14, 15, 19, 8, 18, 12, 2]. Prefix caching and its related protocol considerations as well as partial sequence caching are studied in [17, 7, 6]. It had been shown that prefix/suffix caching is worse than exponential segmentation in terms of caching efficiency in [19]. Studies have also shown that it is appropriate to cache popular media objects in their entirety.

Video staging [20] reduces the peak or average bandwidth requirements between the server and proxy channel by considering the fact that coded video frames have different sizes depending on the scene complexity and coding method. Specifically, if a coded video frame exceeds a predetermined threshold, then the frame is cut such that a portion is cached on the proxy while the other portion remains on the server, thus reducing or smoothing the bandwidth required between the two. In [13, 14, 15], a similar idea is proposed for caching scalable video, and this is done in a manner that co-operates with the congestion control mechanism. The cache replacement mechanism and cache resource allocation problems are studied according to the popularity of video objects.

In [10], the algorithm attempts to partition a video into different chunks of frames with alternating chunks stored in the proxy, while in [11], the algorithm may select groups of non-consecutive frames for caching in the proxy. The caching problem for layer-encoded video is studied in [8]. The cache replacement of streaming media is studied in the [18, 12].

2. ADAPTIVE AND LAZY SEGMENTATION BASED CACHING SYSTEM

This section describes our proposed segmentation-based caching algorithm. In our algorithm, each object is fully cached according to the proposed aggressive admission policy when it is accessed for the first time. The fully cached object is kept in the cache until it is chosen as an eviction victim by the replacement policy. At that time, the object is segmented using the lazy segmentation strategy and some segments are evicted by the first phase of the two-phase iterative replacement policy. From then on, the segments of the object are adaptively admitted by the aggressive admission policy or adaptively replaced as described in the second phase of the two-phase iterative replacement policy.

For any media object accessed through the proxy, a data structure containing the following items is created and maintained. This data structure is called the access log of the object.

- T_1: the time instance the object is accessed for the first time;
- T_r: the last reference time of the object. It is equal to T_1 when the object is accessed for the first time;
- L_{sum}: the sum of the duration of each access to the object;
- n: the number of accesses to the object;
- L_b: the length of the base segment;
- n_s: the number of the cached segments of the object.

Quantities T_r, n and n_s are dynamically updated upon each access arrival. Quantity L_{sum} is updated upon each session termination. Quantity L_b is decided when the object is segmented.

In addition, the following quantities can be derived from the above items and are used as measurements of access activities to each object. In our design, T_c is used to denote the current time instance. At time instance T_c, we denote the access frequency F as $\frac{n}{T_r - T_1}$, and denote the average access duration L_{avg} as $\frac{L_{\text{sum}}}{n}$. Both of these quantities are also updated upon each access arrival.

We now present the three major modules of the caching system. The aggressive admission policy is presented in section 2.1. Section 2.2 describes the lazy segmentation strategy. Details of the two-phase iterative replacement policy are presented in section 2.3.

2.1 Aggressive Admission Policy

For any media object, cache admission is evaluated each time it is accessed with the following aggressive admission policy.

- If there is no access log for the object, the object is accessed for the first time. Assuming the full length of the object is known to the proxy, sufficient cache space is allocated through an adaptive replacement algorithm as described in section 2.3. The accessed object is subsequently cached entirely regardless of the request's accessing duration. An access log is also created for the object and the recording of the access history begins.

- If an access log exists for the object (not the first time), but the log indicates that the object is fully cached, the access log is updated. No cache admission is necessary.

- If an access log exists for the object (not the first time), but the log indicates that the object is not fully cached, the system aggressively considers caching the $(n_s + 1)th$ segment if $L_{\text{avg}} \geq \frac{1}{a} * (n_s + 1) * L_b$, where a is a constant determined by the replacement policy (see section 2.3). The inequality indicates that the average access duration is increasing to the extent that the cached n_s segments can not cover most of the requests while a total of $n_s + 1$ segments can. Therefore, the system should consider the admission of the next uncached segment. The final determination of whether this uncached segment is finally cached or not is determined by the replacement policy (see section 2.3). (In our system, $a = 2$, that is, when $\frac{L_{\text{sum}}}{n} \geq \frac{n_s+1}{2} * L_b$ is true, the next uncached segment of this object is considered to be cached.)

In summary, using aggressive 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.

2.2 Lazy Segmentation Strategy

The key of the lazy segmentation strategy is as follows. Once there is no cache space available and thus cache replacement is needed, the replacement policy calculates the caching utility of each cached object (see section 2.3). Subsequently, the object with the smallest utility value is chosen as the victim if it is not active (no request is currently accessing it). If the victim object is 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 of this object, that is, $L_b = L_{\text{avg}}$. Note that the value of L_b is fixed once it is determined. The object is then segmented uniformly based on L_b. After that, the first a segments are kept in the cache, while the remaining segments are evicted (see section 2.3). The number of cached segments, n_s, is updated in the access log of the object accordingly. If a later request demands more than the cached number of segments of this object, data of length L_b (except for the last segment) is prefetched from the server.

In contrast with existing segmentation strategies, in which segmentation is performed when the object is accessed for the first time, the lazy segmentation strategy delays the segmentation process as late as possible, thus allowing the proxy to collect a sufficient amount of accessing statistics to improve the accuracy of the segmentation for each media object. By using the lazy segmentation strategy, the system adaptively sets different base segment lengths for different objects according to real time user access behaviors.

2.3 Two-Phase Iterative Replacement Policy

The replacement policy is used to select cache eviction victims. We design a two-phase iterative replacement policy as follows. First of all, a utility function is derived to help the victim selection process. Several factors are considered to predict future accesses.

- The average number of accesses;

- the average duration of accesses;

- the length of the cached data (could be the whole object, or could be some segments), which is the cost of the storage; and

- the probability of the future access. In addition to the above factors used to predict the users' future access behaviors, the two-phase iterative replacement policy considers the possibility of future accesses as follows: the system compares the $T_c - T_r$, the time interval between now and the most recent access, and the $\frac{T_r - T_1}{n}$, the average time interval for an access happening in the past. If $T_c - T_r > \frac{T_r - T_1}{n}$, the possibility that a new request arrives soon for this object is small. Otherwise, it is more likely that a request may be coming soon.

Intuitively, the caching utility of an object is proportional to the average number of accesses, the average duration of accesses and the probability of accesses. In addition, it is inversely proportional to the size of the occupied cache space. Therefore, the caching utility function of each object is defined as follows:

$$\frac{f_1(\frac{L_{\text{sum}}}{n})^{p_1} * f_2(F)^{p_2} * \text{MIN}\{1, \frac{\frac{T_r - T_1}{n}}{T_c - T_r}\}}{f_3(n_s * L_b)^{p_3}}, \tag{1}$$

where
$f_1(\frac{L_{\text{sum}}}{n})$ represents the average duration of future access;
$f_2(F)$ represents the average number of future accesses;
$\text{MIN}\{1, \frac{\frac{T_r - T_1}{n}}{T_c - T_r}\}$ denotes the possibility of future accesses; and
$f_3(n_s * L_b)$ is the cost of disk storage.

Equation 1 can be simplified as

$$\frac{\frac{L_{\text{sum}}}{T_r - T_1} * MIN\{1, \frac{\frac{T_r - T_1}{n}}{T_c - T_r}\}}{n_s * L_b} \tag{2}$$

when $p_1 = 1$, $p_2 = 1$ and $p_3 = 1$.

Compared with the distance-sensitive utility function $\frac{1}{(T_c - T_r) \times i}$ (i represents the i^{th} segment, $\frac{1}{T_c - T_r}$ is the estimated frequency) used in the exponential segmentation method [19] which favorably caches segments closer to the beginning of media objects, the proposed utility function provides a more accurate estimation based on the popularity of segments regardless of their relative positions in the media object. This helps to ensure that less popular segments get evicted from the cache.

Given the definition of the utility function, we design a two-phase iterative replacement policy to maximize the aggregated utility value of the cached objects. Upon object admission, if there is not enough cache space, the system calculates the caching utility of each object currently in the cache. The object with the smallest utility value is chosen as the victim and partial cached data of this object is evicted in one of the two phases as follows.

- First Phase: If the access log of the object indicates that the object is fully cached, the object is segmented as described in section 2.2. The first a ($a = 2$) segments are kept and the rest segments are evicted right after the segmentation is completed. Therefore, the portion of the object left in cache is of length $2 * L_b$. Given that $L_b = L_{avg}$ at this time instance, the cached 2 segments cover a normal distribution in the access duration.

- Second Phase: If the access log of the object indicates that the object is partially cached, the last cached segment of this object is evicted.

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 two-phase iterative replacement policy reduces the chances of making wrong decisions 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. In addition, the iterative nature of the replacement procedure ensures that the aggregated utility value of the cached objects is maximized.

Note that even after an object is fully replaced, the system still keeps its access log. If not, when the object is accessed again, it should be fully cached again. Since media objects tend to have diminishing popularity as the time goes on, if the system caches the object in full again, it results in an inefficient use of the cache space. Our design enhances the resource utilization by avoiding this kind of situation.

3. PERFORMANCE EVALUATION

An event-driven simulator is implemented to evaluate the performance of the exponential segmentation, the uniform segmentation, and our proposed adaptive and lazy segmentation techniques by using synthetic and real traces. For the adaptive and lazy segmentation strategy, Equation 2 is used as the utility function. The exponential segmentation strategy always reserves a portion of cache space (10%) for beginning segments, and leaves the rest for later segments. The utility function used is as described in section 2.3 for the replacement of later segments, while the LRU policy is used for the beginning segments. The only difference between the uniform segmentation and the exponential segmentation method is as follows. Instead of segmenting the object exponentially, the uniform segmentation strategy segments the object with a constant length. Since the exponential segmentation strategy always caches the first 6 segments as in [19], for a fair comparison, the uniform segmentation strategy always caches the same length of first several segments of media objects. Thus whether the exponentially increasing segment length plays an important role can also be evaluated.

The byte hit ratio is defined as how many bytes are delivered to the client from the proxy directly, normalized by the total bytes the client requested. It is used as the major metric to evaluate the reduction of the network traffic to the server and the disk bandwidth utilization on the server. The delayed start request ratio is defined as how many requests among the total do not have a startup latency since the initial portion of the requested object is cached on the proxy. It is used to indicate the efficiency of these techniques in reducing the user perceived startup latency. The average number of cached objects per time unit denotes that the average number of objects whose segments are partially or fully cached. It is used to indicate whether the method favorably caches the beginning segments of a large number of different objects or favorably caches the popular segments of a small number of different objects.

3.1 Workload Summary

Table 1 lists some known properties of synthetic traces and an actual enterprise trace.

Trace Name	Num of Request	Num of Object	Size (GB)	λ	α	Range (minute)	Duration (day)
WEB	15188	400	51	4	0.47	2-120	1
VOD	10731	100	149	60	0.73	60-120	7
PART	15188	400	51	4	0.47	2-120	1
REAL	9000	403	20	-	-	6 - 131	10

Table 1: Workload Summary

WEB and VOD denote the traces for the Web and VoD environment with complete viewing, while PARTIAL denotes the trace for the Web environment with partial viewing. These synthetic traces assume a Zipf-like distribution ($p_i = f_i / \sum_{i=1}^{N} f_i, f_i = 1/i^{\alpha}$) for the popularity of media objects. They also assume the request arrival follows the Poisson distribution ($p(x, \lambda) = e^{-\lambda} * (\lambda)^x/(x!), x = 0, 1, 2...$).

REAL denotes the trace extracted from server logs of HP Corporate Media Solutions, covering the period from April 1 through April 10, 2001.

3.2 Evaluation on Complete Viewing Traces

Figure 1 shows the performance results from simulations using the WEB trace. Lazy segmentation refers to our proposed adaptive and lazy segmentation method. Exponential segmentation refers to the exponential segmentation method. Uniform segmentation (1K) refers to the uniform segmentation method with 1KB sized segments, while uniform segmentation (1M) refers to the uniform segmentation method with 1MB sized segments[1]. Evident from Figure 1(a), lazy segmentation achieves the highest byte hit ratio. When the cache size is 10%, 20% and 30% of the total object size, the byte hit ratios of lazy segmentation and exponential segmentation are more than 50% and 13%, 67% and 39%, 75% and 29%, respectively. The absolute performance gap is more than 30% on average and gradually decreases with the increase of available cache space. On average, uniform segmentation achieves a similar result as exponential segmentation, which indicates that the exponentially increased length does not have an obvious effect on the byte hit ratio.

Figure 1(b) shows that in terms of the delayed start request ratio, uniform segmentation (1K) achieves the best result, while exponential segmentation is ranked second. This is expected since both of them always favorably cache the beginning segments of media objects. Lazy segmentation achieves the worst percentage among the three. The results indicate that the achieved high byte hit ratio is paid at the expense of a high delayed start request ratio.

Figure 1(c) shows the average number of cached objects per time unit. Lazy segmentation always has the least number of objects cached on average, while it always achieves the best byte hit ratio. The results implicitly indicate that favorably caching the beginning segments of media objects is not efficient in reducing network traffic to the server and disk bandwidth utilization on the server.

The results of the VOD trace shown in Figure 2 show similar trends as those of WEB. The byte hit ratio of lazy segmentation is improved by 28, 24 and 10 percentage points over exponential segmentation when the cache size is 10%, 20% and 30% of the total object size correspondingly.

Since WEB and VOD are the complete viewing scenarios, results from simulations using these two traces demonstrate that in

[1]In the following context, we also use them to represent their corresponding strategies for brevity.

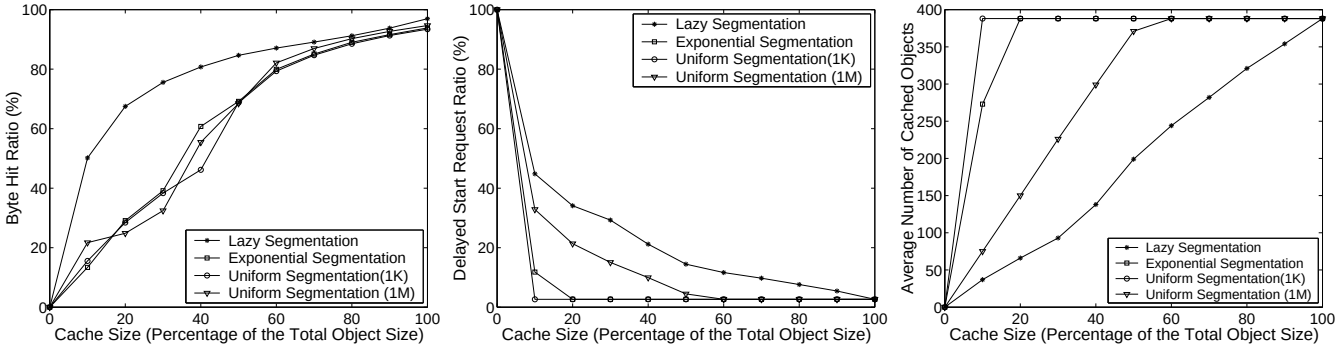

Figure 1: WEB: (a) Byte Hit Ratio, (b) Delayed Start Request ratio, and (c) Average Number of Cached Objects

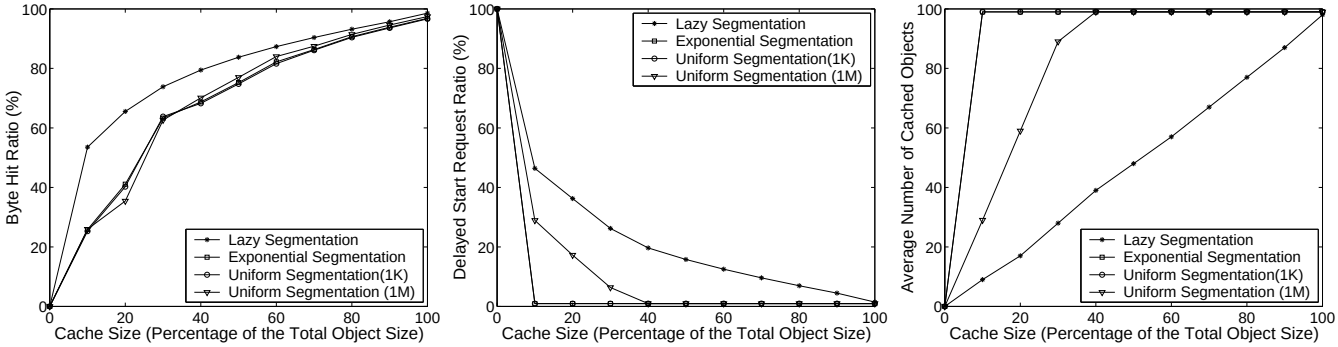

Figure 2: VOD: (a) Byte Hit Ratio, (b) Delayed Start Request Ratio, and (c) Average Number of Cached Objects

terms of the byte hit ratio, it is more appropriate to cache the popular segments of objects instead of favorably caching the beginning segments of a media object. It also implicitly shows that the two-phase iterative replacement policy of our proposed method can successfully identify the popular objects when compared with the distance-sensitive replacement policy used in the the exponential segmentation technique.

3.3 Evaluation on Partial Viewing Traces

Figure 3 shows the performance results from simulations using PARTIAL. 80% of requests in PARTIAL only access 20% of the object. Shown in Figure 3(a), lazy segmentation achieves the best byte hit ratio: when the cache size is 10%, 20% and 30% of the total object size, the byte hit ratio increases by 28, 42, and 7 percentage points, respectively, over the exponential segmentation method. Compared with results of WEB, the improvement of our proposed method over the exponential segmentation method is reduced due to the 80% partial viewing sessions. Again, uniform segmentation (1K) achieves a similar byte hit ratio as that of exponential segmentation on average.

Figure 3(b) shows the delayed start request ratio. As expected, the lowest is achieved by uniform segmentation (1K) while exponential segmentation still gets the second lowest percentage. Lazy segmentation achieves the highest. This confirms that the higher byte hit ratio is paid by a higher delayed start request ratio.

Figure 3(c) shows the average number of cached objects of PARTIAL. It shows that lazy segmentation always has the least number of objects cached, while it always achieves the highest byte hit ratio. The results further indicate that favorably caching the beginning segments of media objects is not effective for alleviating the bottlenecks of delivering streaming media objects.

For the real trace REAL, Figure 4(a) shows the byte hit ratio as a function of increased cache size. When the cache size is 20%, 30%, 40% and 50% of the total object size, the byte hit ratio increases of lazy segmentation are 31, 28, 29, 30 percentage points, respectively, over exponential segmentation. The average performance improvement is about 30 percentage points. The trends shown in Figure 4(a) are consistent with the previous ones.

Figure 4(b) shows the delayed start request ratio for the REAL trace. It shows that lazy segmentation has similar results as exponential segmentation. Its performance even exceeds that of exponential segmentation when the cache size is 10% and 20% of the total cache size. This is due to the nature of partial viewing in REAL. In our proposed method, much cache space is available for the beginning segments of objects. The result reflects the adaptiveness of our proposed method.

Figure 4(c) shows that consistent with previous evaluations, our proposed method still has the least number of objects cached on average while achieving the highest byte hit ratio.

The results of REAL show that lazy segmentation achieves the highest byte hit ratio and nearly the lowest delayed start request ratio. The adaptiveness of our proposed method shown in this evaluation confirms our analysis in Section 1.

All these performance results show that: (1) in terms of byte hit ratio (reductions of server workload and network traffic), our proposed adaptive and lazy segmentation method always performs best with the least number of objects cached; (2) uniform segmentation (1K) achieves a similar result on both the byte hit ratio and the delayed start request ratio as the exponential segmentation method on average.

The performance results also indicate that favorably caching the beginning segments of media objects is not effective in alleviating

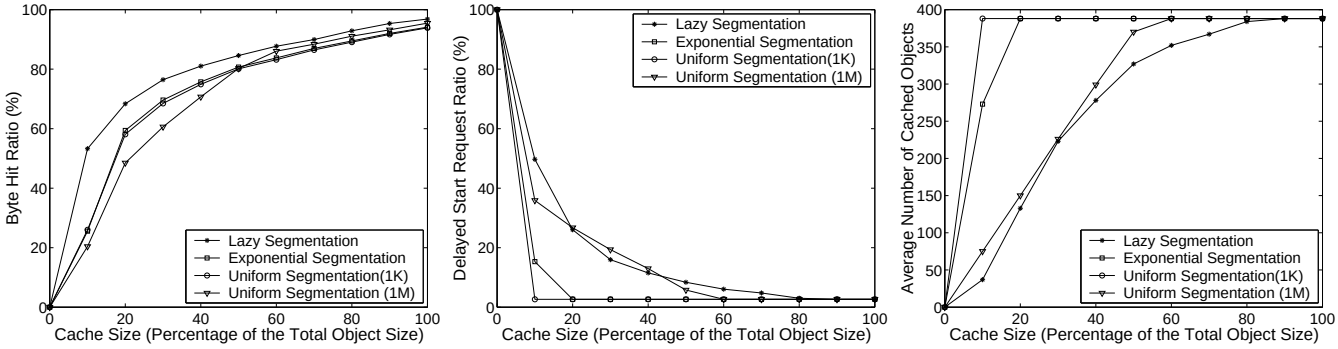

Figure 3: PART: (a) Byte Hit Ratio, (b) Delayed Start Request Ratio, and (c) Average Number of Cached Objects

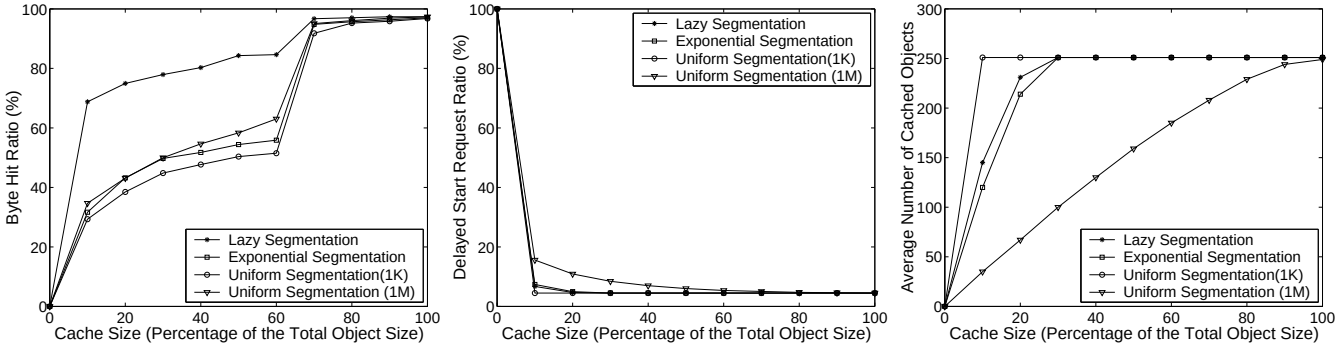

Figure 4: REAL: (a) Byte Hit Ratio, (b) Delayed Start Request Ratio, and (c) Average Number of Cached Objects

the bottlenecks for delivering streaming media and exponentially increasing segment length does not have an obvious advantage over constant segment length in terms of byte hit ratio.

Uniform segmentation with other base segment lengths are also tested. They achieve similar performance results as those of uniform segmentation (1M) in byte hit ratio and worse results in delayed start request ratio.

4. ADDITIONAL RESULTS

In Section 3, the adaptive and lazy segmentation strategy is evaluated comparatively with the exponential segmentation and the uniform segmentation strategies. We have learned that generally the adaptive and lazy segmentation strategy achieves a higher byte hit ratio. We also know that the adaptive and lazy segmentation strategy does not reserve space for beginning segments of media objects, while the exponential segmentation and uniform segmentation strategies do reserve space for beginning segments (10% of the total cache size in the experiments). Thus one may argue that the higher byte hit ratio achieved by lazy segmentation comes from freeing the reserved space. To examine whether this is true or not, two groups of experiments are designed and performed based on the changes of either the lazy segmentation strategy, either the exponential and uniform segmentation strategies. These experiments are used to evaluate whether the freeing of reserved space has a significant impact on the byte hit ratio improvement achieved by lazy segmentation.

4.1 Small Cache Size for Lazy Segmentation

Firstly, we use a smaller cache space for lazy segmentation. It means that the available cache space for the adaptive and lazy segmentation strategy is the same as the cache space for the exponential segmentation strategy other than the reserved part. In these experiments, the reserved portion for the exponential segmentation strategy is set to 10%, thus the totally available cache space for lazy segmentation is only 90% of the cache space for exponential segmentation. The remaining 10% is reserved for no use.

Figure 5 shows the corresponding results using the WEB trace. Compared with Figure 1, the achieved byte hit ratio of lazy segmentation does decrease a little. However, it still achieves the highest byte hit ratio among all the strategies as shown in Figure 5(a). In Figure 5(b), the delayed start request ratio achieved by lazy segmentation is even worse compared to Figure 1(b), due to the decrease of the total available cache size. Figure 5(c) shows that due to the case when only 90% space is available for lazy segmentation, the average number of cached objects is less than the total number of objects when the cache size is 100% of the total object size.

The results using the VOD trace is shown in Figure 6. All the trends indicated on Figure 6 are similar to those on Figure 5, with smaller changes correspondingly. The smaller changes are due to the longer durations of the VOD trace.

Compared with Figure 3, the variations of results using the trace PARTIAL in Figure 7 are more significant on byte hit ratio and delayed start request ratio. Due to the partial viewing nature of this trace, the performance results are more sensitive to the changes of the available cache size for the adaptive and lazy segmentation strategy. However, lazy segmentation still has a significant advantage over the others in achieving a high byte hit ratio. Note that in Figure 7, the average number of cached objects reaches the total number of objects when the cache size is 100%. This is different in Figure 5 and Figure 6.

Figure 8 shows the corresponding results using the REAL trace. The byte hit ratio, delayed start request ratio, and average number

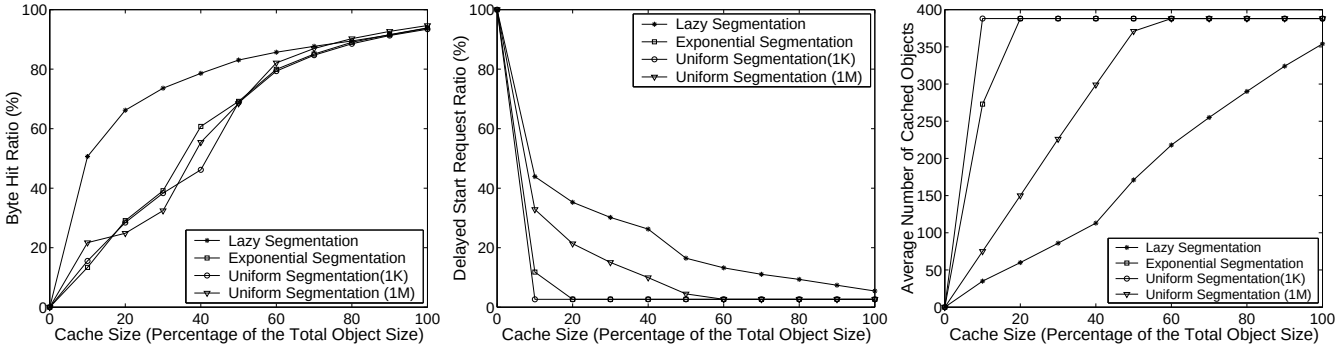

Figure 5: WEB: (a) Byte Hit Ratio, (b) Delayed Start Request Ratio, and (c) Average Number of Cached Objects

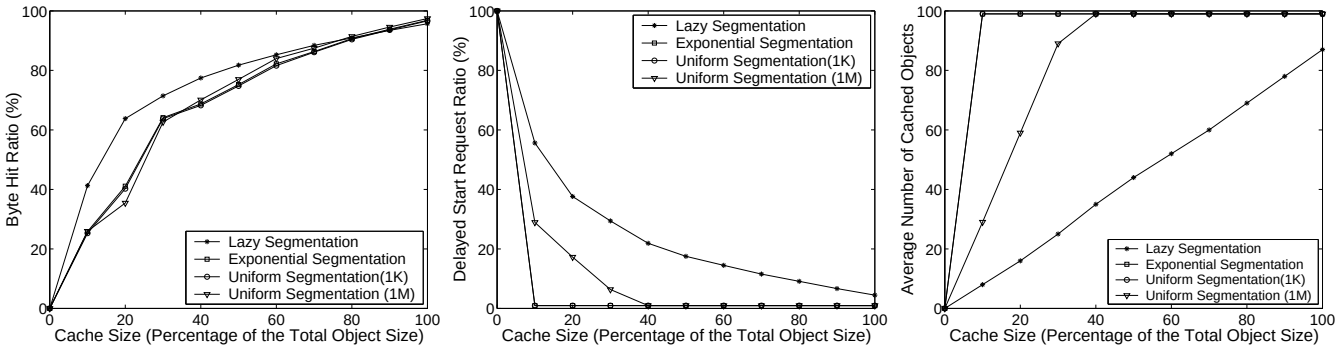

Figure 6: VOD: (a) Byte Hit Ratio, (b) Delayed Start Request Ratio, and (c) Average Number of Cached Objects

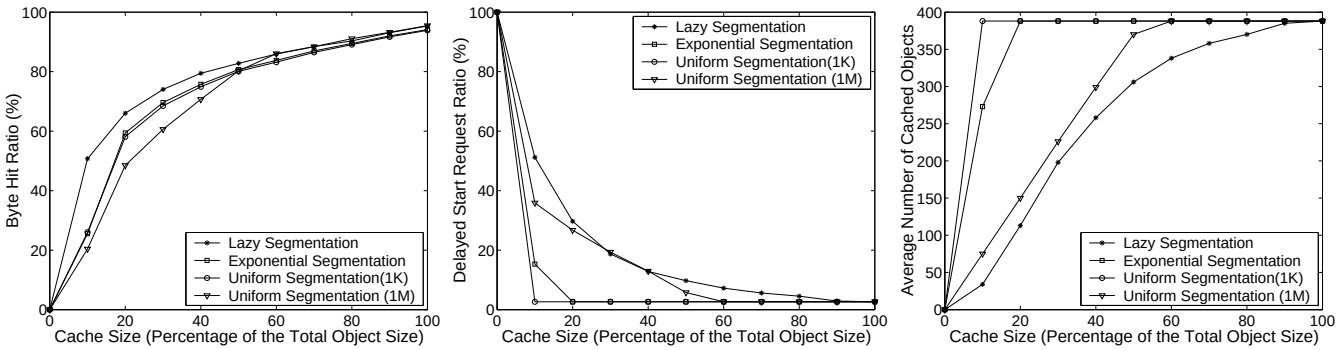

Figure 7: PART: (a) Byte Hit Ratio, (b) Delayed Start Request Ratio, and (c) Average Number of Cached Objects

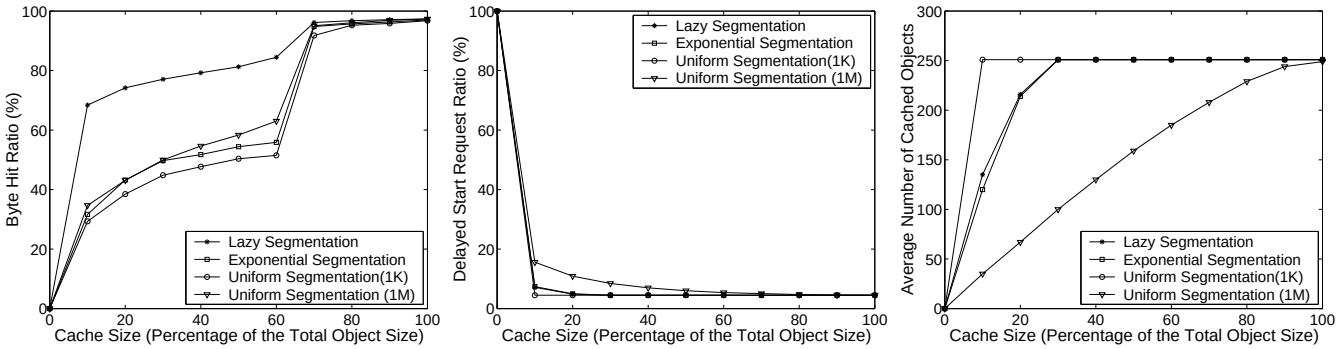

Figure 8: REAL: (a) Byte Hit Ratio, (b) Delayed Start Request Ratio, and (c) Average Number of Cached Objects

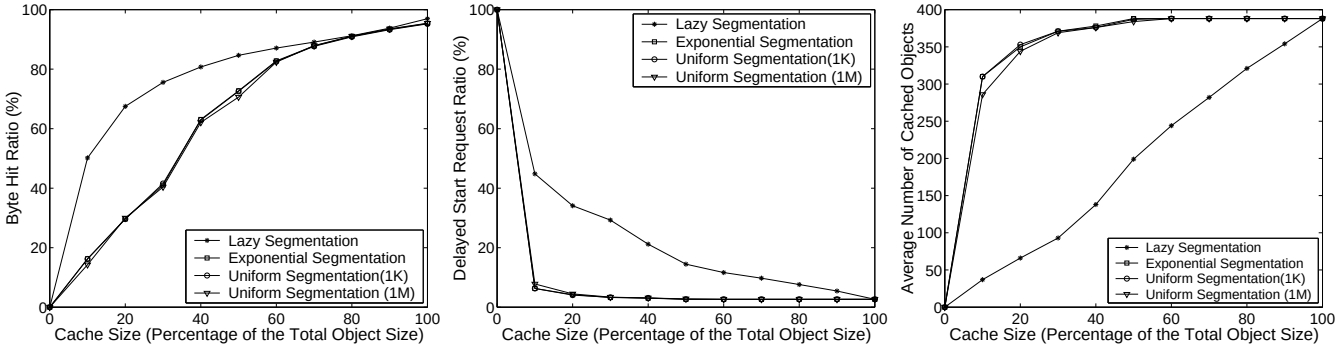

Figure 9: WEB: (a) Byte Hit Ratio, (b) Delayed Start Request Ratio, and (c) Average Number of Cached Objects

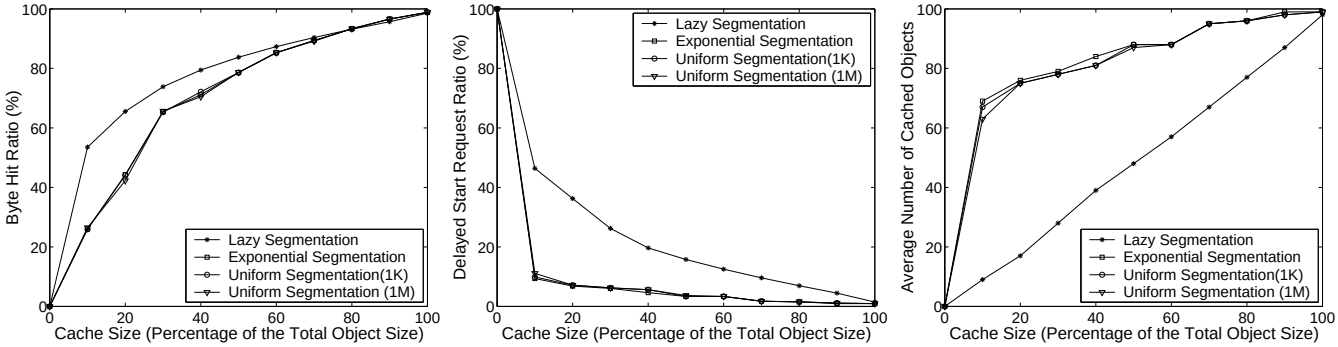

Figure 10: VOD: (a) Byte Hit Ratio, (b) Delayed Start Request Ratio, and (c) Average Number of Cached Objects(c)

of cached objects change little for lazy segmentation. We believe this is due to the fact that the REAL trace has many prematurely terminated sessions.

In summary, the results of these experiments show that the highest byte hit ratio achieved by lazy segmentation is not from freeing the reserved space.

4.2 Eliminating Reserved Space for Uniform and Exponential Segmentation

In previous section, we have altered the available cache space for lazy segmentation to show that the freeing of reserved space does not change the conclusion we made. In this section, we design another group of experiments for this purpose. In these experiments, the available cache space for the uniform and exponential segmentation strategies and the lazy segmentation strategy are the same. However, for uniform and exponential segmentation, no cache space is reserved for the beginning segments. Following the original strategy, for the uniform and exponential segmentation strategies, the first several segments will be cached when the object is initially accessed while the rest will not. Once the rest of an object is accessed again, it is considered to be cached according to its caching utility as defined in [19]. The beginning segments and the remaining segments compete together for the cache space when they need it. These comparisons can provide more insights into whether the byte hit ratio improvement of our proposed adaptive and lazy segmentation strategy comes from the reserved cache space.

Figure 9 shows the corresponding results using the WEB trace. Compared with Figure 1, the uniform and exponential segmentation strategies have similar performance results, and the lazy segmentation strategy still achieves the highest byte hit ratio among all the strategies as shown in Figure 9(a). In Figure 9(b), the delayed start request ratio achieved by the uniform and exponential segmentation strategies are almost same. Figure 9(c) shows that the average number of cached objects for uniform and exponential segmentation are close to each other.

The results using the VOD trace is shown in Figure 10. All the trends indicated on Figure 10 are similar to those shown on Figure 9.

We show the results using the PARTIAL trace in Figure 11. It is interesting to see that lazy segmentation has a larger number of cached objects on average when the cache size increases from 30% in Figure 11(c). The reason behind this is that a large portion of the sessions are terminated earlier. In Figure 11(a), we find that lazy segmentation still achieves the highest byte hit ratio.

The corresponding results using the trace REAL are shown in Figure 12. Again, the results do not show significant impact of the reserved space on the byte hit ratio improvement for the uniform and exponential segmentation strategies.

5. EVALUATION OF THE REPLACEMENT UTILITY FUNCTION

In the previous evaluation, we always use Equation 2 as the utility function for lazy segmentation. To examine the effects of the variant utility function on system performance, we vary p_1 and p_2 in Equation 1 to simulate the different weights of the frequency and the average access duration. To simulate different weights of storage space, we vary p_3 in Equation 1. The corresponding results are presented in this section.

Figure 13 shows the performance results using the WEB trace. Figure 13(a) shows that the byte hit ratio changes slightly when the caching utility is changing with the available cache size. Figure

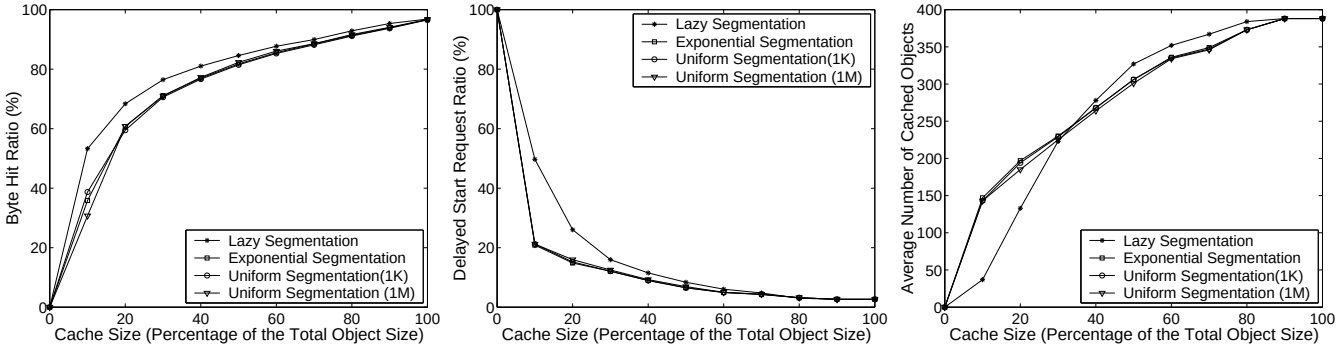

Figure 11: PART: (a) Byte Hit Ratio, (b) Delayed Start Request Ratio, and (c) Average Number of Cached Objects

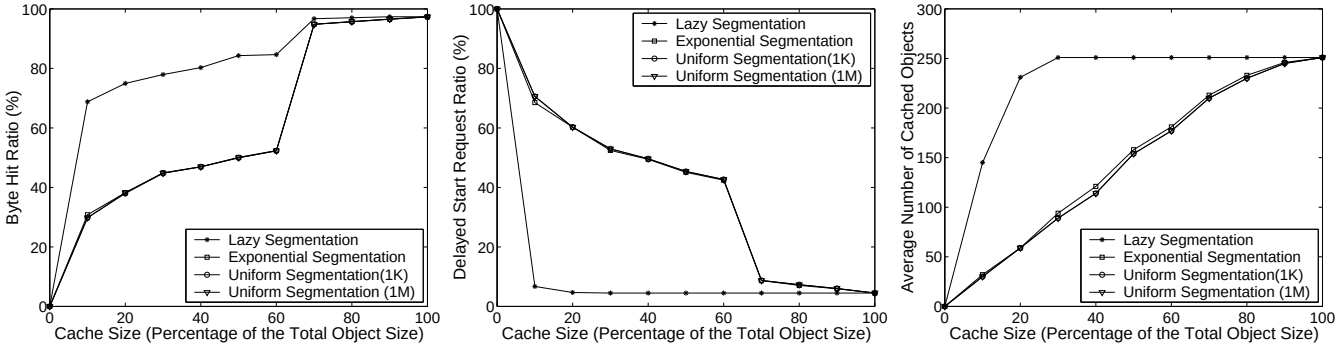

Figure 12: REAL: (a) Byte Hit Ratio, (b) Delayed Start Request Ratio, and (c) Average Number of Cached Objects

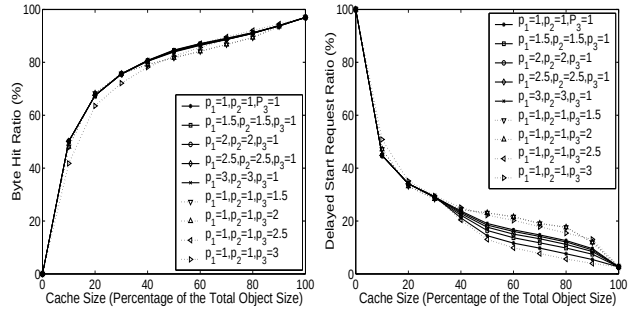

Figure 13: WEB: Variant Utility Functions of Replacement Policy

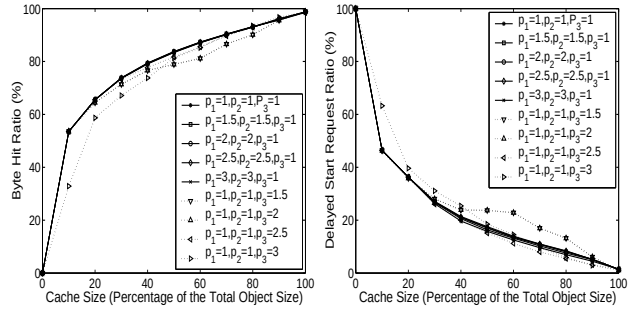

Figure 14: VOD: Variant Utility Functions of Replacement Policy

13(b) indicates that the delayed start request ratio has larger variations when the cache size increases, especially when the cache size increase from 40% to 90% of the total object size. It also shows that for lazy segmentation, a better delayed start request ratio can be achieved once the priority of the cache space consumption is increased.

The results from the VOD trace in Figure 14 indicate similar trends as before. Figure 14(a) shows that the byte hit ratio varies widely compared to those of the WEB trace while Figure 14(b) indicates the changes of the delayed start request ratio are not as significant as those of the WEB trace.

Figure 15 shows the results using the PARTIAL trace, which is a partial viewing case of the WEB trace. Generally, the trends shown in Figure 15 are similar to those of WEB. However, Figure 15(b) does show larger variations of the delayed start request ratio when the cache space is increasing. This indicates that the PARTIAL trace is more sensitive to the storage space consumption.

Figure 16 shows the results using the REAL trace. It indicates similar trends as shown in Figure 15. As shown on Figure 16, the increase of the priority on storage space consumption will worsen and fluctuate the delayed start request ratio for the lazy segmentation strategy. This is due to the large number of early terminated sessions.

Through all these experiments, we find the performance of the adaptive and lazy segmentation strategy can be adjusted depending on available cache space to a certain extent. However, in general, increasing the weights of average access duration and frequency has less impact on the performance results.

6. CONCLUSION

We have proposed a streaming media caching proxy system based

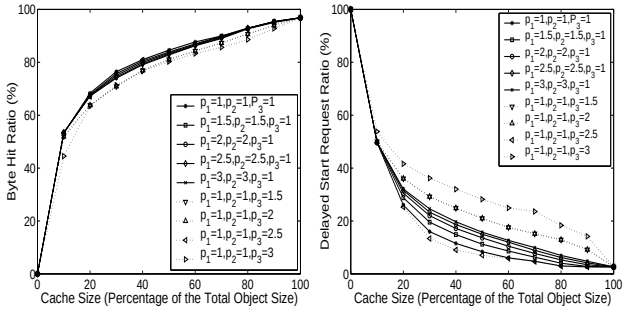

Figure 15: PART: Variant Utility Functions of Replacement Policy

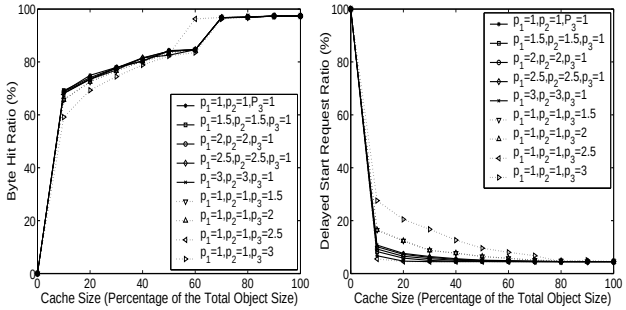

Figure 16: REAL: Variant Utility Functions of Replacement Policy

on an adaptive and lazy segmentation strategy with an aggressive admission policy and two-phase iterative replacement policy. The proposed system is evaluated by simulations using synthetic traces and an actual trace extracted from enterprise media server logs. Compared with a caching system using uniform and exponential segmentation methods, the byte hit ratio achieved by the proposed method is improved by 30% on average, which indicates a 30% reduction in the server workload and network traffic. Additional evaluations show that the improvement in byte hit ratio of the lazy segmentation is not from the freeing of reserved space. The results show that the adaptive and lazy segmentation strategy is a highly efficient segment-based caching method that alleviates bottlenecks for the delivery of streaming media objects.

We are currently investigating the trade-offs between network traffic reduction and client startup latency.

7. ACKNOWLEDGMENT

We would like to thank anonymous reviewers for their helpful comments on this paper. The work is supported by NSF grant CCR-0098055 and a grant from Hewlett-Packard Laboratories.

8. REFERENCES

[1] E. Bommaiah, K. Guo, M. Hofmann and S. Paul, " Design and Implementation of a Caching System for Streaming Media over the Internet", *IEEE Real Time Technology and Applications Symposium*, May 2000.

[2] Y. Chae, K. Guo, M. Buddhikot, S. Suri, and E. Zegura, "Silo, Rainbow, and Caching Token: Schemes for Scalable Fault Tolerant Stream Caching", *IEEE Journal on Selected Areas in Communications, Special Issue on Internet Proxy Services*, Vol. 20, pp 1328-1344, Sept. 2002.

[3] S. Chen, B. Shen, S. Wee and X. Zhang, "Adaptive and Lazy Segmentation Based Proxy Caching for Streaming Media Delivery", HPCS Lab Tech. Report TR-03-002, College of William and Mary, Jan., 2003.

[4] L. Cherkasova and M. Gupta, "Characterizing Locality, Evolution, and Life Span of Accesses in Enterprise Media Server Workloads", *NOSSDAV 2002*, Miami, FL, May 2002.

[5] M. Chesire, A. Wolman, G. Voelker and H. Levy, "Measurement and Analysis of a Streaming Media Workload", *Proc. of the 3rd USENIX Symposium on Internet Technologies and Systems*, San Francisco, CA, March 2001.

[6] M. Y.M. Chiu and K. H.A Yeung, "Partial Video Sequence Caching Scheme for VOD Systems with Heteroeneous Clients" *IEEE Transactions on Inducstrial Electronics*, 45(1):44-51, Feb. 1998.

[7] S. Gruber, J. Rexford and A. Basso, " Protocol considertations for a prefix-caching for multimedia streams", *Computer Network*, 33(1-6):657-668, June 2000.

[8] J. Kangasharju, F. Hartanto, M. Reisslein and K. W. Ross, "Distributing Layered Encoded Video Through Caches", *Proc. of IEEE INFOCOM'01*, Anchorage, AK, USA, 2001.

[9] S. Lee, W. Ma and B. Shen, "An Interactive Video Delivery and Caching System Using Video Summarization", *Computer Communications*, vol. 25, no. 4, pp. 424-435, Mar. 2002.

[10] W.H. Ma and H.C. Du, "Reducing Bandwidth Requirement for Delivering Video over Wide Area Networks with Proxy Server", *Proc. of International Conferences on Multimeida and Expo.*, 2000, vol. 2, pp. 991-994.

[11] Z. Miao and A. Ortega, "Scalable Proxy Caching of Video Under Storage Constraints", *IEEE Journal on Selected Areas in Communications*, vol. 20, pp 1315-1327, Sept. 2002.

[12] M. Reisslein, F. Hartanto and K. W. Ross, "Interactive Video Streaming with Proxy Servers", *Proc. of IMMCN*, Atlantic City, NJ, Feb. 2000.

[13] R. Rejaie, M. Handely and D. Estrin, "Quality Adaptation for Congestion Controlled Video Playback over the Internet", *Proc. of ACM SIGCOMM'99*, Cambridge, MA, Sept. 1999.

[14] R. Rejaie, M. Handley, H. Yu and D. Estrin, "Proxy Caching Mechanism for Multimedia Playback Streams in the Internet", *Proc. of WCW'99*, Apr. 1999.

[15] R. Rejaie, H. Yu, M. Handely and D. Estrin, "Multimedia Proxy Caching Mechanism for Quality Adaptive Streaming Applications in the Internet", *Proc. of IEEE INFOCOM'00*, Tel-Aviv, Israel, March 2000.

[16] S. Sen, L. Gao, J. Rexford and D. Towsley, "Optimal Patching Schemes for Efficient Multimedia Streaming", *NOSSDAV '99*, Basking Ridge, NJ, June 1999.

[17] S. Sen, K. Rexford and D. Towsley, "Proxy Prefix Caching for Multimedia Streams", *Proc. IEEE INFOCOM'99*, New York, USA, March 1999.

[18] R. Tewari, H. Vin, A. Dan and D. Sitaram, "Resource-based Caching for Web Servers", *Proc. SPIE/ACM Conference on Multimeida Computing and Networking*, Jan. 1998.

[19] K. Wu, P. S. Yu and J. L. Wolf, "Segment-based Proxy Caching of Multimedia Streams", *WWW'2001*, pp. 36-44.

[20] Z.L. Zhang, Y. Wang, D.H.C. Du and D. Su, "Video Staging: A Proxy-server Based Approach to End-to-end Video Delivery over Wide-area Networks", *IEEE Transactions on Networking*, Vol. 8, no. 4, pp. 429-442, Aug. 2000.