Fast Proxy Delivery of Multiple Streaming Sessions in Shared Running Buffers

Songqing Chen, *Member, IEEE*, Bo Shen, *Senior Member, IEEE*, Yong Yan, Sujoy Basu, and Xiaodong Zhang, *Senior Member, IEEE*

***Abstract*—With the falling price of memory, an increasing number of multimedia servers and proxies are now equipped with a large memory space. Caching media objects in the memory of a proxy helps to reduce the network traffic, the disk I/O bandwidth requirement, and the data delivery latency. The *running buffer* approach and its alternatives are representative techniques to caching streaming data in the memory. There are two limits in the existing techniques. First, although multiple running buffers for the same media object co-exist in a given processing period, data sharing among multiple buffers is not considered. Second, user access patterns are not insightfully considered in the buffer management. In this paper, we propose two techniques based on *shared running buffers* in the proxy to address these limits. Considering user access patterns and characteristics of the requested media objects, our techniques adaptively allocate memory buffers to fully utilize the currently buffered data of streaming sessions, with the aim to reduce both the server load and the network traffic. Experimentally comparing with several existing techniques, we show that the proposed techniques achieve significant performance improvement by effectively using the shared running buffers.**

***Index Terms*—Patching, proxy caching, shared running buffer, streaming media, video-on-demand (VOD).**

I. Introduction

A BASIC structure of a content delivery network is a server-proxy-client system. In this system, the server delivers the content to the client through a proxy. The proxy can choose to cache the object so that subsequent requests to the same object can be served directly from the proxy without contacting the server. Proxy caching strategies have therefore been the focus of many studies. Much work has been done in caching static web content to reduce network load and end-to-end latency. Typical examples of such work include CERN httpd [1], Harvest [2], and Squid [3].

The caching of streaming media content presents a different set of challenges: 1) the size of a streaming media object is usually orders of magnitudes larger than traditional web contents—for example, a 2-h long MPEG video requires about 1.4 GB of disk space, while traditional web objects are of the magnitude of 10 kB and 2) the demand of continuous and timely delivery of a streaming media object is more rigorous than that of text-based Web pages. Therefore, a lot of resources have to be reserved for delivering the streaming media data to a client. In practice, even a relatively small number of clients can overload a media server, creating bottlenecks by demanding high disk bandwidth on the server and hight network bandwidth to the clients.

To address these challenges, researchers have proposed different methods to cache streaming media objects via partial caching, patching or proxy buffering. In the partial caching approach, either a prefix [4] or segments [5] of a media object instead of the whole object is/are cached. Therefore, less storage space is required. For on-going streaming sessions, patching can be used so that later sessions for the same object can be served simultaneously. For proxy buffering, either a fixed-size running buffer [6] or an interval [7] can be used to allocate buffers to buffer a running window of an on-going streaming session in the memory of the proxy. Among these techniques, partial caching uses disk resource on the proxy; patching uses storage resource on the client side, and theoretically no memory resource is required at the proxy; proxy buffering uses the memory resource on the proxy. However, neither running buffer nor interval caching uses the memory resource to the full extent. More detailed analysis of these techniques can be found in Section II.

In this paper, we first propose a new memory-based caching algorithm for streaming media objects using shared running buffers (SRB). In this algorithm, the memory space on the proxy is allocated adaptively based on the user access pattern and the requested media objects themselves. Streaming sessions are cached in running buffers. This algorithm dynamically caches media objects in the memory while delivering the data to the client so that the bottleneck of the disk and/or network I/O is relieved. More importantly, similar sessions can share different runs of the on-going sessions. This approach is especially useful when requests to streaming objects are highly temporal localized. The SRB algorithm: 1) adaptively allocates memory space according to the user access pattern; 2) enables maximal sharing of the cached data in memory; 3) optimally reclaims memory space when requests terminate; and 4) applies a near-optimal replacement policy in the real time.

Based on the SRB media caching algorithm, we further propose an efficient media delivering algorithm: Patching SRB (PSRB), which further improves the performance of the media data delivery without the necessity of caching.

Manuscript received November 24, 2003; revised January 7, 2005. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Ton A.C.M. Kalker.

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. Basu are with the Mobile and Media System Lab, Hewlett-Packard Laboratories, Palo Alto, CA 94304 USA (e-mail: boshen@hpl.hp.com; basus@hpl.hp.com).

Y. Yan is an independent consultant, Fremont, CA 34077 USA.

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.2005.858417

Simulations are conducted based on synthetic workloads of web media objects with mixed lengths as well as workloads with accesses to lengthy media objects in the video-on-demand (VOD) environment. In addition, we use an access log of a media server in a real enterprise intranet to further conduct simulations. The simulation results indicate that the performance of our algorithms is superior to previous solutions.

The rest of the paper is organized as follows. In Section II, the related work is surveyed. Section III describes the optimal memory-based caching algorithms we propose. To test the performance of the proposed algorithms, we use synthetic workloads as well as real workload from an enterprise media server. Some statistical analysis of these workloads is provided in Section IV. Performance evaluations are conducted in Section V. We then make concluding remarks in Section VI.

II. Related Work

In this section, we survey previous work related to the caching of streaming media content. Three types of methods are investigated as follows.

A. Partial Caching

Storing the entire media object in the proxy may be inefficient if mostly small portions of very large media objects are accessed. This is particularly true if the cached streaming media object is not popular. The first intuition is to cache portions of the media object. These partial caching systems always use the storage on the proxy. Some early work on the storage for media objects can be found in [8], [9]. Two typical types of partial caching have been investigated.

Prefix caching [4] stores only the first part (prefix) of the popular media object. When a client requests a media stream whose prefix is cached, the proxy delivers the prefix to the client while it requests the remainder of the object from the origin server. By caching a (large enough) prefix, the start-up latency perceived by the client is reduced. The challenge lies in the determination of the prefix size. Items such as roundtrip delay, server-to-proxy latency, video specific parameters (e.g., size, bit rate, etc.), and retransmission rate of lost packets can be considered in calculating the appropriate prefix length.

Alternatively, media objects can be cached in segments. This is particularly useful when clients only view portions of the objects. The segments in which clients are not interested will not be cached. Wu *et al.* [5] use segments with exponentially incremental size to model the fact that clients usually start viewing a streaming media object from the beginning and are more and more likely to terminate the session toward the end of the object. A combination of fixed length and exponential length segment-based caching method is considered in the RCache and Silo project [10]. Lee *et al.* [11] uses a context-aware segmentation so that segments of user interest are cached.

B. Session Sharing

The delivery of a streaming media object takes time to complete. We call this delivery process a streaming session. Sharing is possible among sessions that overlap. To this end, various kinds of patching schemes are proposed for sharing along the time line. This type of work is typically seen in research related to VOD systems.

Patching algorithms [12] use storage on the client device so that a client can listen to multiple channels. Greedy patching always patches to the existing full streaming session while grace patching restarts a new full streaming session at some appropriate point in time. Optimal patching [13] further assumes sufficient storage resource at the client side to receive as much data as possible while listening to as many channels as possible.

In the greedy patching, a full server-to-proxy session is established at the first request arrival for a certain media object. All subsequent requests for the same media object are served as patches to the this full session before it terminates, at which point another full session is established for the next request arrival. It is obvious that the patching session can be excessively long. It is more efficient to start a new session even before the previous full session terminates. In the grace patching, upon the arrival of the fifth request, a new full server-to-proxy session is started. Therefore, the patching session of the last request is significantly shorter than in the greedy patching example. The optimal point to start a new full session can be derived mathematically given certain request arrival pattern [14].

The optimal patching scheme is introduced in [13]. The basic idea is that later sessions patch to as many on-going sessions as possible. The ongoing sessions can be a full session or a patching session. The last request patches to the patching session started for the second last request as well as the full session started for the first request. This approach achieves the maximum server-to-proxy traffic reduction. On the other hand, it requires extra resource in client storage and proxy-to-client bandwidth as well as complex scheduling at the proxy.

Note that no storage resource is required on the proxy for these session sharing algorithms. On the other hand, since clients have to listen to multiple channels and store content before its presentation time, client side storage is necessary.

One special type of the session-sharing approaches is the batching approach [15], [16]. In this approach, requests are grouped and served simultaneously via multicast. Therefore, requests arriving earlier have to wait. Hence, certain delay is introduced to the early arrived requests.

C. Proxy Buffering

To further improve the caching performance for streaming media, memory resources on the proxy are used. This is more and more practical given that the price for memory keeps falling. For memory-based caching, running buffer and interval caching methods have been studied.

Dynamic caching [6] uses a fixed-size running buffer to cache a running window of a streaming session. It works as follows. When a request arrives, a buffer of a predetermined length is allocated to cache the media data the proxy fetches from the server, in the expectation that closely followed requests will use the data in the memory instead of fetching from other sources (e.g., disk or origin server or other cooperative caches). A fix-sized buffer is allocated upon the arrival of the first request. The buffer is filled by the session started by the first request and run to the end of stream. Subsequent requests are served

from the buffer if they arrive within the time period covered by the buffer. If a request arrives beyond the range covered by the first buffer, a second buffer of the same predetermined size is allocated, which servers the requests arrived in the time period covered by this new buffer.

On a different approach, interval caching [7], [17] further considers request arrival patterns so that memory resource is more efficiently used. Interval caching considers closely arrived requests for the same media object as a pair and orders their arrival intervals globally. The subsequent allocation of memory always favors smaller intervals. Effectively, more requests are served given a fixed amount of memory. Upon the arrival of the second request an interval is formed with the first request, and a buffer of the size equivalent to the interval is allocated. The buffer is filled by the session started by the first request from this point on. The session initiated by the second request only needs to receive the first part of data from the server. It can receive the rest of data from the buffer. When there is no space for the newly formed intervals, the largest buffer will be released, and the space will be allocated to the newly formed one.

Note that these two approaches use the memory resource on the proxy so that the client is relieved from the buffer requirement as in the patching schemes discussed in the previous section. In addition, one proxy-to-client channel suffices for these buffering schemes.

III. Shared Running Buffer (SRB) Media Caching Algorithm

Buffering streaming content in memory has been shown to have great potential to alleviate contention on the streaming server so that a larger number of sessions can be served. It has been shown that two existing memory caching approaches: running buffer caching [6] and interval caching [7], [17], do not make effective use of the limited memory resource. Motivated by the limits of the current memory buffering approaches, we design a SRB-based caching algorithm for streaming media to better utilize memory. In this section, with the introduction of several new concepts, we first describe the basic SRB media caching algorithm in detail. Then, we present an extension to the SRB: PSRB.

A. Related Definitions

The algorithm first considers buffer allocation in a time span T starting from the first request. We denote R_i^j as the jth request to media object i, and T_i^j as the arrival time of this request. Assume that there are n request arrivals within the time span T and R_i^n is the last request arrived in T. For the convenience of representation without losing precision, T_i^1 is normalized to 0 and T_i^j (where $1 < j \leq n$) is a time relative to T_i^1. Based on the above, the following concepts are defined to capture the characteristics of the user request pattern.

1) *Interval Series*: An interval is defined as the difference in time between two consecutive request arrivals. We denote I_i^k as the k-th interval for object i. An *Interval Series* consists a group of intervals. Within the time T, if $n = 1$, the interval I_i^1 is defined as ∞; otherwise, it is defined as

$$I_i^k = \begin{cases} T_i^{k+1} - T_i^k, & \text{if } 1 < k < n \\ T - T_i^n, & \text{if } k = n. \end{cases} \quad (1)$$

Since I_i^n represents the time interval between the last request arrival and the end of the investigating time span, it is also called the *Waiting Time*.

2) *Average Request Arrival Interval (ARAI)*: The *ARAI* is defined as $\sum_{k=1}^{n-1} I_i^k/(n-1)$ when $n > 1$. *ARAI* does not exist when $n = 1$ since it indicates only one request arrival within time span T and thus we set it as ∞.

For the buffer management, three buffer states and three timing concepts are defined, respectively, as follows.

1) *Construction State and Start-Time*: When an initial buffer is allocated upon the arrival of a request, the buffer is filled while the request is being served, expecting that the data cached in the buffer could serve closely followed requests for the same object. The size of the buffer may be adjusted to cache less or more data before it is frozen. Before the buffer size is frozen, the buffer is in the *Construction State*. Thus, the *Start-Time* S_i^j of a buffer B_i^j, the j-th buffer allocated for object i, is defined as the arrival time of the last request before the buffer size is frozen. The requests arriving in a buffer's *Construction State* are called the *resident requests* of this buffer and the buffer is called the *resident buffer* of these requests.

   Note that if no buffer exists for a requested object, a first buffer with superscript $j = 1$ is allocated. Subsequent buffer allocations use monotonically increasing js even if the immediate preceding buffer has been released. Therefore, only after all buffers of the object run to the end, is it possible to reset j and start from 1 again.

2) *Running State & Running-Distance*: After the buffer freezes its size it serves as a running window of a streaming session and moves along with the streaming session. Therefore, the state of the buffer is called the *Running State*.

   The *Running-Distance* of a buffer is defined as the distance in time between the start-time of a running buffer and the start-time of its preceding running buffer. We use D_i^j to denote the *Running-Distance* of B_i^j. Note that for the first buffer allocated to an object i, D_i^1 is equal to the length of object i: L_i, assuming a complete viewing scenario. Since we are encouraging sharing among buffers, clients served from B_i^j are also served from any preceding buffers that are still in running state. This requires that the running-distance of B_i^j equals to the time difference with the closest preceding buffer in running state. Mathematically, we have

$$D_i^j = \begin{cases} L_i, & \text{if } j = 1 \\ S_i^j - S_i^m, & \text{if } j > 1, \end{cases} \quad (2)$$

   where $m < j$ and S_i^m is the start time of the closest preceding buffer in running state.

3) *Idle State & End-Time*: When the running window reaches the end of the streaming session, the buffer enters the *Idle*

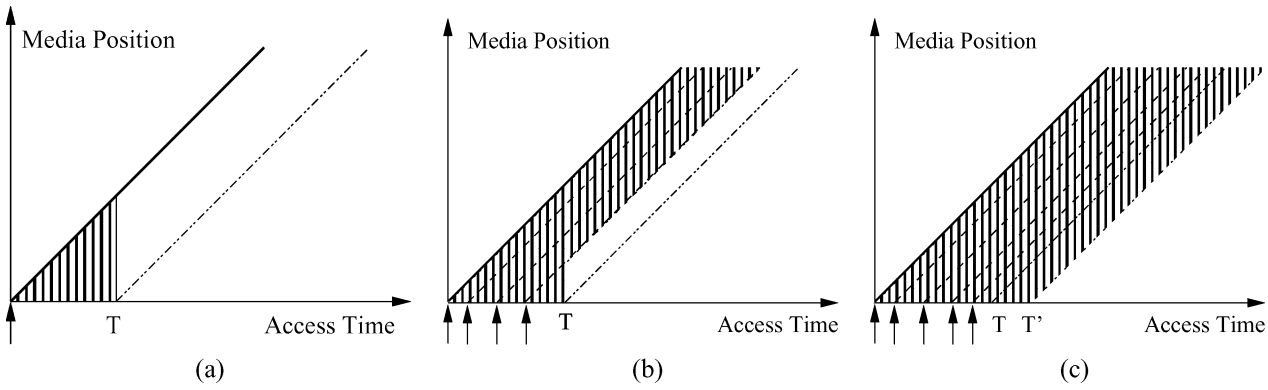


Fig. 1. SRB memory allocation: the initial buffer freezes its size.

State, which is a transient state that allows the buffer to be reclaimed.

The *End-Time* of a buffer is defined as the time when a buffer enters *idle state* and is ready to be reclaimed. The *End-Time* of the buffer B_i^j, denoted as E_i^j, is defined as

$$E_i^j = \begin{cases} S_i^j + L_i, & \text{if } j = 1 \\ \min\left(S_i^{\text{latest}} + D_i^j, S_i^j + L_i\right), & \text{if } j > 1. \end{cases} \quad (3)$$

S_i^{latest} denotes the start time of the latest running buffer for object i. Here, E_i^j is dynamically updated upon the forming of new running buffers. The detailed updating procedure of is described in the following section.

B. SRB Algorithm

For an incoming request to the object i, the SRB algorithm works as follows: 1) If the latest running buffer of the object i is caching the prefix of the object i, the request is served directly from all the existing running buffers of the object. 2) Otherwise, (a) If there is enough memory, a new running buffer of a predetermined size T is allocated. The request is served from the new running buffer and all existing running buffers of the object i. (b) If there is not enough memory, the SRB buffer replacement algorithm (see Section III-B-3) is invoked to either re-allocate an existing running buffer to the request or serve this request without caching. 3) Update the *End-Time*s of all existing buffers of the object i based on (3). During the process of the SRB algorithm, parts of a running buffer could be dynamically reclaimed as described in Section III-B.2 due to the request termination and the buffer is dynamically managed based on the user access pattern through a lifecycle of three states as described in Section III-B1.

1) SRB Buffer Lifecycle Management: Initially, a running buffer is allocated with a predetermined size of T. Starting from the *Construction State*, the buffer then adjusts its size by going through a three-state lifecycle management process as described in the following.

- Case 1: the buffer is in the *Construction State*. The proxy makes a decision at the end of T as follows.
  - If $ARAI = \infty$, which indicates that there is only one request arrival so far, the initial buffer enters the *Idle State* (case 3) immediately. For this request, the proxy acts as a bypass server, i.e., content is passed to the client without caching. This scheme gives preference to more frequently requested objects in the memory allocation. Fig. 1(a) illustrates this situation. The shadow indicates the allocated initial buffer, which is reclaimed at T.
  - If $I^n > ARAI$ (I^n is the waiting time), the initial buffer is shrunk to the extent that the most recent request can be served from the buffer. Subsequently, the buffer enters the *Running State* (case 2). This running buffer then serves as a shifting window and run to the end. Fig. 1(b) illustrates an example. Part of the initial buffer is reclaimed at the end of T. This scheme performs well for periodically arrived request groups.
  - If $I^n \leq ARAI$, the initial buffer maintains the construction state and continues to grow to the length of T', where $T' = T - I^n + ARAI$, expecting that a new request arrives very soon. At T', the $ARAI'$ and $I^{n\prime}$ are recalculated and the above procedure repeats. Eventually, when the request to the object becomes less frequent, the buffer will freeze its size and enter the *Running State* (case 2). In the extreme case, the full length of the media object is cached in the buffer. In this case, the buffer also freezes and enters the running state (a static running). For most frequently accessed objects, this scheme ensures that the requests to these objects are served from the proxy directly. Fig. 1(c) illustrates this situation. The initial buffer has been extended beyond the size of T for the first time.

    The buffer expansion is bounded by the available memory in the proxy. When the available memory is exhausted, the buffer freezes its size and enters the running state regardless of future request arrivals.
- Case 2: the buffer is in the *Running State*. After a buffer enters the running state, it starts running away from the beginning of the media object and subsequent requests can not be served completely from the running buffer. In this case, a new buffer of an initial size T is allocated and goes through its own lifecycle starting from case 1. Subsequent requests are served from the new buffer as well as its preceding running buffers.

Fig. 2 illustrates the maximal data sharing in the SRB algorithm. The requests R_i^{k+1} to R_i^n are served simultaneously from B_i^1 and B_i^2. Late requests are served from all existing running buffers. *Note that except for the first buffer, the other buffers do not run to the end of the object.*

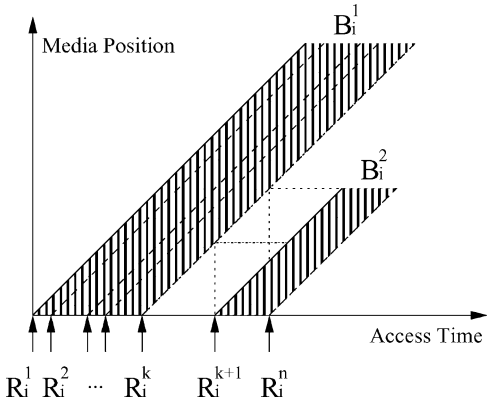


Fig. 2. Multiple running buffers.

The *Running-Distance* and *End-Time* are determined based on (2) and (3), respectively, for any buffer entering the *Running State*. In addition, the *End-Times* of its preceding running buffers need to be modified according to the arrival time of the latest request, as shown in (3).

- Case 3: the buffer is in the *Idle State*. When a buffer enters the *Idle State*, it is ready for reclamation.

In the above algorithm, the time span T (which is the initial buffer size) is determined based on the object length. Typically, a *Scale* factor (e.g., 1/2 to 1/32) of the origin length is used. To prevent a extremely large or small buffer size, the buffer size is bounded by a upper bound *High-Bound* and a lower bound *Low-Bound*. These bounds are dependent on the streaming rate to allow the initial buffer to cache a reasonable portion (e.g., 1 to 10 min) of media objects. The algorithm requires the client be able to listen to multiple channels at the same time: once a request is posted, it should be able to receive data from all the ongoing running buffers of that object simultaneously.

2) SRB Dynamic Reclamation: Memory reclamation of a running buffer is triggered by two different types of session terminations: complete session termination and premature session termination. In complete session termination, a session terminates only when the delivery of the whole media object is completed. Assuming that R_i^1 is the first request being served by a running buffer, when R_i^1 reaches the end of the media object, the *resident buffer* of R_i^1 is reclaimed as follows.

- If the resident buffer is the only buffer running for the media object, the resident buffer enters the *Idle State*. In this state, the buffer maintains its content until all the resident requests reach the end of the session, at which time the buffer is released.
- If the resident buffer is not the only buffer running, that is, there are succeeding running buffers, the buffer enters the *Idle State* and maintains its content until its *End-Time*. Note that the *End-Time* may have be updated by succeeding running buffers.

Premature session termination is much more complicated. In this case, a request that arrives later may terminate earlier. Consider a group of consecutive requests R_i^1 to R_i^n, the session for one of the requests, say R_i^j, where $1 < j < n$, terminates before everyone else. The situation is handled as follows.

- If R_i^j is served from the middle of its resident buffer, that is, there are preceding and succeeding requests served from the same running buffer, the resident buffer maintain its current state and R_i^j is deleted from all its associated running buffers. Fig. 3(a) and (a') show the buffer situation before and after R_i^j is terminated, respectively.
- If R_i^j is served from the head of its resident buffer as shown in Fig. 3(b), the request is deleted from all of its associated running buffers. The resident buffer enters the *Idle State* for a time period of I. During this time period, the content within the buffer is moved from R_i^{j+1} to the head. At the end of the time period I, the buffer space from the tail to the last served request is released and the buffer enters the *Running State* again as shown in Fig. 3(b').
- If R_i^j is served at the tail of a running buffer, two scenarios are further considered.

— After deleting the R_i^j from the request list of its resident buffer, if the request list is not empty, then do nothing. Alternatively, the algorithm can choose to shrink the buffer to the extent that R_i^{j-1} is served from the buffer assuming R_i^{j-1} is a resident request of the same buffer. In this case, the *End-Time* of the succeeding running buffers needs to be adjusted.

— If R_i^j is at the tail of the last running buffer as shown in Fig. 3(c), the buffer is shrunk to the extent that R_i^{j-1} is the last request served from the buffer. R_i^j is deleted from the request list. Subsequently, the buffers run as shown in Fig. 3(c').

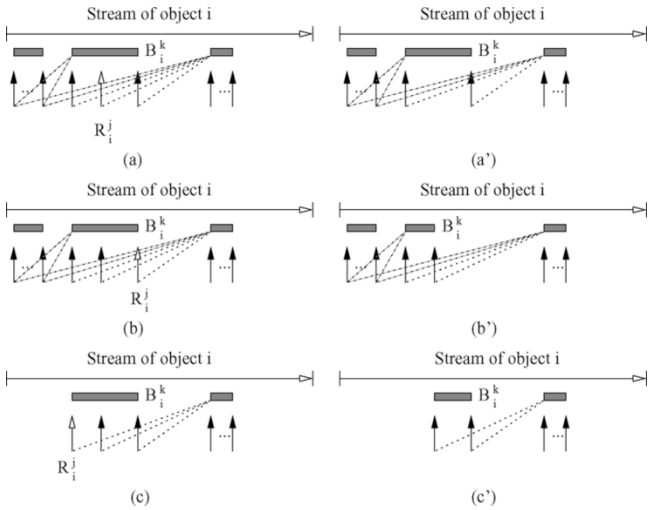


Fig. 3. SRB memory reclamation: different situations of session termination.

3) SRB Buffer Replacement Policies: The replacement policy is important in the sense that the available memory is still scarce compared to the size of video objects. So to efficiently use the limited resources, it is critical to achieve the best performance gain. In this section, we propose popularity-based replacement policies for the SRB media caching algorithm. The basic idea is described as follows.

- When a request arrives while there is no available memory, all the objects that have on-going streams in memory are ordered according to their popularities calculated in a certain past time period. If the object being demanded has a higher popularity than the least popular object in memory, then the latest running buffer of the least popular object is released, and the space is re-allocated to the new request. Those requests without

1. SRB lifecycle management
- **1.1** upon the arrival of the new request
- **1.2** **if** there is a running buffer in construction state
  server this request from the buffer
- **1.3** **else**
  if no sufficient memory space
  call 3
  allocate a buffer, its *End-Time* is T
- **1.4** update all related buffer's *End-Time*
- **1.5** **switch**(buffer state):
- **1.6** *Construction:*
  if $ARAI = \infty$
  goto 1.8
  if $I^n > ARAI$
  shrink buffer
  goto 1.7
  if $I^n \leq ARAI$
  expand buffer
  $T = T - I^n + ARAI$
  goto 1.5
- **1.7** *Running:*
  buffer runs away
  if new request arrival
  update *End-Time*
  update *Running-Distance*
- **1.8** *Idle:*
  call 2

2. SRB buffer dynamic reclamation
- **2.1** **if** complete session termination
  release resident buffer at its *End-Time*
- **2.2** **if** premature session termination
  if middle request
  delete it from all related buffers
  if head request
  shift the buffered data
  release partial buffer from tail
  goto 1.8
  if tail request
  if last buffer
  release partial buffer
  else
  delete the request from the buffer

3. SRB buffer replacement
- **3.1** order the objects according to their popularity
- **3.2** release the latest buffer of least popular object

Fig. 4. SRB algorithm.

running buffers do not buffer their data at all. In this case, theoretically, they are assumed to have no memory consumption.

Alternatively, the system can choose to start a memoryless session in which the proxy bypasses the content to the client without caching. This is called a nonreplacement policy. We have evaluated the performances of both two polices by simulations in the later section. It is shown that a straightforward nonreplacement police may achieve similar performance given long enough system running time.

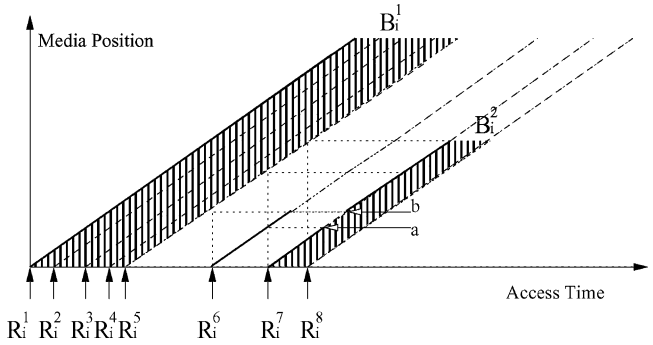


Fig. 5. PSRB caching example.

Fig. 4 shows the SRB algorithm containing these three main components in the pseudocode.

C. PSRB Media Delivering Algorithm

Since the proxy has finite amount of memory space, it is possible that the proxy serves as a bypass server without caching concurrent sessions. The SRB algorithm prohibits the sharing of such sessions, which may lead to excessive server access if there are intensive request arrivals to many different objects. To solve this problem, the SRB algorithm can be extended to a PSRB algorithm which enables the sharing of such bypass sessions. This is related to the session sharing algorithms as discussed in Section II-B. It is important to note that PSRB scheme makes the memory-based SRB algorithm work with the memoryless patching algorithm.

Fig. 5 illustrates a PSRB scenario. The first running buffer B_i^1 has been formed for requests R_i^1 to R_i^5. No buffer is running for R_i^6 since it does not have a close neighboring request. However, a patching session has been started to retrieve the absent prefix in B_i^1 from the content server. At this time, request R_i^6 is served from both the patching session and B_i^1 until the missing prefix is patched. Then, R_i^6 is served from B_i^1 only (the solid line for R_i^6 stops in the figure).

When R_i^7 and R_i^8 arrive and form the second running buffer B_i^2, they are served from B_i^1 and B_i^2 as described in the SRB algorithm. In addition, they are also served from the patching session initiated for R_i^6. Note that the patching session for R_i^6 is transient, or we can think of it as a running buffer session with zero buffer size. As evident from Fig. 5, the filling of B_i^2 does not cause server traffic between position a and b (no solid line between a and b) since B_i^2 is filled from the patching session for R_i^6. Thus, sharing the patching session further reduces the number of server accesses for R_i^7 and R_i^8. By using more client-side storage, PSRB tries to maximize the data sharing among concurrent sessions in order to minimize the server-to-proxy traffic.

IV. Workload Analysis

To evaluate the performance of the proposed algorithms and to compare them with prior solutions, we conduct simulations based on several workloads. Both synthetic workloads and a real workload extracted from enterprise media server logs are used. We design three synthetic workloads. The first simulates accesses to media objects in the Web environment in which the

TABLE I
WORKLOAD STATISTICS

Workload name	Number of requests	Number of objects	Total size (GB)	Object length (min)	Poisson λ	Zipf α	Simulation duration (day)
WEB	15188	400	51	2∼120	4	0.47	1
PARTIAL	15188	400	51	2∼120	4	0.47	1
VOD	10731	100	149	60∼120	60	0.73	7
REAL	9000	403	20	6∼131	-	-	10

length of the video varies from short ones to longer ones. The second simulates the video access in a VOD environment in which only longer streams are served. Both workloads assume complete viewing client sessions. We use WEB and VOD as the name of these workloads. These workloads assume a Zipf-like distribution ($p_i = f_i / \sum_{i=1}^{N} f_i$, $f_i = 1/i^{\alpha}$) for the popularity of the media objects. They also assume request inter arrival to follow the Poisson distribution ($p(x, \lambda) = e^{-\lambda} * (\lambda)^x / (x!), x = 0, 1, 2 \ldots$).

In the case of video accessing in the Web environment, clients accesses to videos may be incomplete, that is, a session may terminate before the whole media object is delivered. We simulate this scenario by designing a partial viewing workload based on the WEB workload. In this workload, called PARTIAL, 80% of the sessions terminate before 20% of the accessed objects is delivered.

For the real workload, we obtain logs from HP Corporate Media Solutions, covering the period from April 1 through April 10, 2001. During these ten days, there were two servers running Windows Media Server, serving contents to clients around the world within HP intranet. The contents include videos, with the coverage of keynote speeches at various corporate and industry events, messages from the company's management, product announcements, training videos and product development courses for employees, etc. This workload is called REAL. A detailed analysis of the overall characteristics of the logs from the same servers covering different time periods can be found in [18].

A. Workload Characteristics

Table I lists some statistics of the four workloads. For the WEB workload, there is a total of 400 unique media objects, with a total size of 51 GB, stored on the server. The length of the media objects ranges from 2 min to 2 h. The request inter-arrival follows a Poisson distribution with $\lambda = 4$ s and $\alpha = 0.47$ (according to [19]) of Zipf-like distribution for object popularities. The media objects are coded and streamed at 256 kbps. The total number of requests is 15 188. The simulation lasts 87 114 s or roughly 24 h. The *Low-Bound* and *High-Bound* for the initial buffer size are set as 2 and 16 MB, respectively.

For the PARTIAL workload, 80% of the requests view only 20% of the requested streaming media objects and then prematurely terminate. Both the partial-viewing requests and the partial-viewed objects are randomly distributed. Other parameters of the synthesized trace are identical to those of WEB as shown in Table I.

For the VOD workload, the number of unique objects stored on the server is 100, which accounts to total size of 149 GB. The length of the objects ranges from 1 to 2 h. The request inter-arrival follows a Poisson distribution with $\lambda = 60$ s. The Zipf-like distribution for object popularity is set as $\alpha = 0.73$ (according to [20]). The media objects are coded and streamed at 2 Mbps. The total number of requests is 10 731. The simulation runs for 654 539 s or roughly a week. The *Low-Bound* and *High-Bound* for initial buffer allocation are set as 16 and 128 MB, respectively.

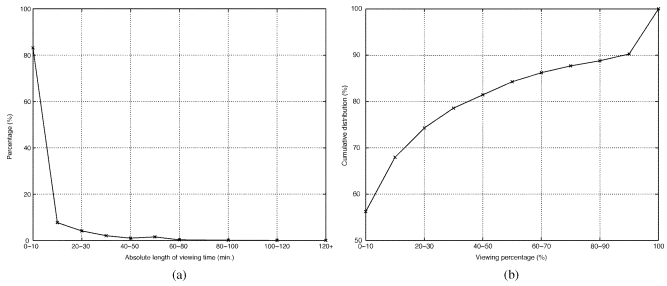


Fig. 6. REAL: (a) distribution of absolute viewing time (b) distribution of viewing percentage.

For the REAL workload, there is a total of 403 objects with a total size of 20 GB. There are 9000 request arrivals in a time span of 916 427 s or roughly ten days. Fig. 6(a) shows the distribution of the absolute length of viewing time (in minutes). It shows that 83.08% of the sessions last less than 10 min. Fig. 6(b) shows the cumulative distribution of the percentage of viewing time with respect to the full object length. It shows that 56.24% of the requests access less than 10% of the media object. Only 9.74% of the requests access the full objects.

B. Shared Session

During the course of a streaming session of an object, when there are other requests accessing the same object, this portion of the streaming session is shared. The actual caching benefit depends on the number of sharing requests. For example, the benefit of caching an ongoing sessions simultaneously shared by two requests is different from that by three requests. To further evaluate the benefit the proxy system can get by caching for shared sessions, we obtain the distribution of the number of requests on the shared sessions. This statistics are collected as follows. At each time instance (second), we recode the numbers of on-going sessions that are shared by different numbers of sharing requests. These numbers are accumulated in bins representing numbers of sharing requests. At the end of the simulation, the number in each bin is divided by the total simulation duration in second, thus obtaining a histogram of the averaged number of shared sessions at any time instance. Fig. 7(a) shows the distribution in full range on the number of sharing requests, and Fig. 7(b) shows the range from 2 to 20 in more detail. For a given point (x, y) on the curves, y indicates the average number of on-going sessions that are shared by x number of requests.

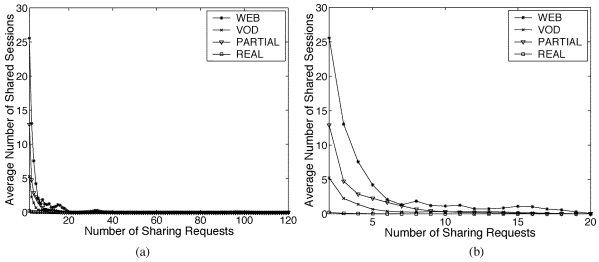


Fig. 7. Shared request histogram: (a) full and (b) part.

We find that the number of sharing sessions ranges predominately from 2 to 20. Since the sessions in the VOD workload last the longest, it is expected that its average number of shared sessions is the largest among the group of workloads. PARTIAL is the partial viewing case of WEB; thus, the number of shared sessions in PARTIAL should be less than that in WEB workload.

V. Performance Evaluation

A. Evaluation Metrics

We have implemented an event-driven simulator to model a proxy's memory caching behaviors. Since *object hit ratio* or *hit ratio* is not suitable for evaluating the caching performance of the streaming media, we use *server traffic reduction rate* (shown as "bandwidth reduction" in the figures) to evaluate the performance of the proposed caching algorithms. If the algorithms are employed on a server, this parameter indicates the disk I/O traffic reduction rate.

Using SRB or PSRB algorithms, a client needs to listen to multiple channels for maximal sharing of cached data in the proxy's memory. We measure the traffic between the proxy and the client in terms of the *average client channel requirement*. This is an averaged number of channels the clients are listening to during the sessions. Since the clients are listening to earlier on-going sessions, storage is necessary at the client side to buffer the data before its presentation. We use the *average client storage requirement* in percentage of the full size of the media object to indicate the storage requirement on the client side.

The effectiveness of the algorithms is studied by simulating different *scale* factors for the allocation of the initial buffer size and varying memory cache capacities. The streaming rate is assumed to be constant for simplicity. The simulations are conducted on HP workstation x4000, with 1-GHz CPU and 1-GB memory.

For each simulation, we compare a set of seven algorithms in three groups. The first group contains the buffering schemes which include dynamic buffering and interval caching. The second group contains the patching algorithms, namely greedy patching, grace patching and optimal patching algorithms. The third group contains the two shared running buffer algorithms proposed in this paper.

B. Performance on Synthetic Workloads

We consider complete viewing scenario for streaming media caching in both Web environments and VOD environments. We also simulate the partial viewing scenario for web environment. There are no partial viewing cases for media delivery in the VOD environment.

1) Complete Viewing Workload of Web Media Objects: First, we evaluate the caching performance with respect to initial buffer size. With a fixed memory capacity of 1 GB, the initial buffer size varies from 1 to 1/32 of the length of the media object. For each *scale* factor, an initial buffer of different size is allocated if the media object length is different. The *server traffic reduction*, the *average client channel requirement* and the *average client storage requirement* are recorded in the simulation. The results are plotted in Fig. 8.

Fig. 8(a) shows the server traffic reduction achieved by each algorithm. Note that PSRB achieves the best reduction and SRB achieves the next best reduction except optimal patching. RB caching achieves the least amount of reduction. As expected, the performance of the three patching algorithms does not depend on the *scale* factor for allocating of the initial buffer. Neither does that of interval caching since it allocates buffers based on access intervals.

For the running buffer schemes, we notice some variation in the performance with respect to the *scale* factor. In general, the variations are limited. To a certain extent, the performance gain of the *bandwidth reduction* is a trade-off between the number of running buffers and the size of running buffers. A larger buffer indicates that more requests can be served from the it. However, a larger buffer indicates less memory space left for other requests. This in turn leads to more server accesses since there is no available proxy memory. On the other hand, a smaller buffer may serve a smaller number of requests but it leaves more proxy memory space to allocate for other requests.

Fig. 8(b) and (c) shows the average channel and storage requirement on the client. Note that optimal patching achieves the better server traffic reduction by paying the penalty of imposing the biggest number of client channels required. Comparatively, PSRB and SRB requires 30% $\sim$ 60% less client channels while achieving similar or better server traffic reduction.

PSRB allows session sharing even when memory space is not available. It is therefore expected that PSRB achieves the maximal server traffic reduction. In the mean time, it also requires the maximum client side storage and client channels. On the other hand, SRB achieves six percentage point less traffic reduction than PSRB, but the requirement on client channel and storage is significantly lower.

We now investigate the performance of different algorithms with respect to various memory capacities on the proxy. In this simulation, we use a fixed *scale* factor of 1/4 for the initial buffer size. Fig. 9(a) indicates flat traffic reduction rate for the three patching algorithms. This is expected since no proxy memory resource is utilized in patching. On the other hand, all the other algorithms investigated achieve higher traffic reduction when memory capacity increases. It is important to note that the proposed two SRB algorithms achieve better traffic reduction than the interval caching and running buffer schemes.

In Fig. 9(b), the client channel requirement decreases for the PSRB algorithm when the cache capacity increases. This is again expected since more clients are severed from the proxy buffers instead of proxy patching sessions. When the cache capacity reaches 4 GB, PSRB requires only 30% of the client

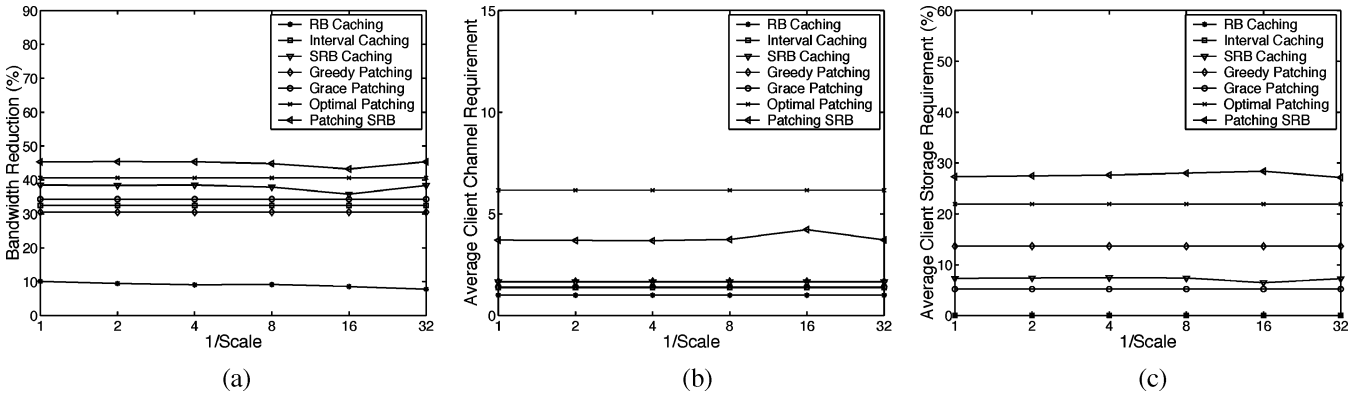


Fig. 8. WEB: (a) server traffic reduction, (b) average client channel, and (c) storage requirement(%) with 1-GB proxy memory.

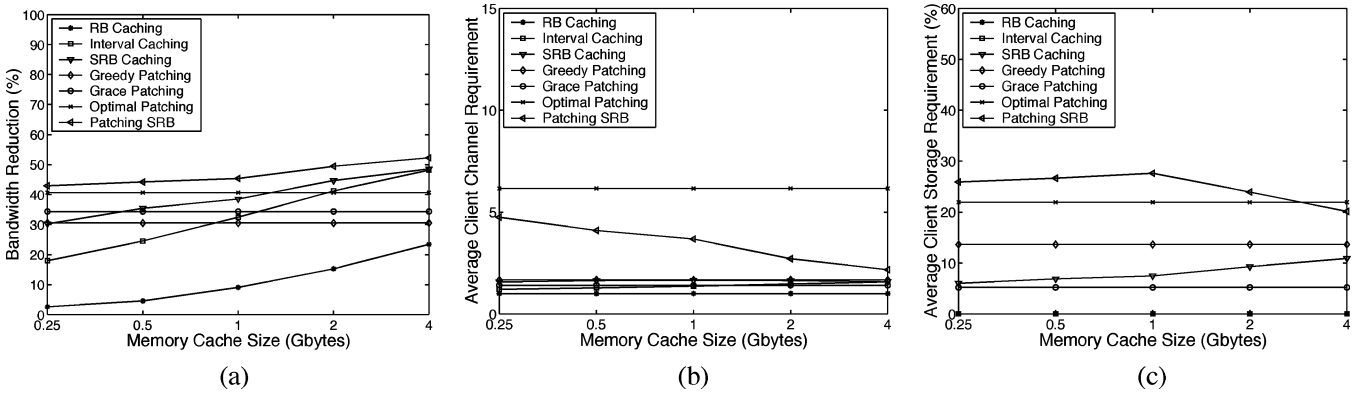


Fig. 9. WEB: (a) bandwidth reduction, (b) average client channel, and (c) storage requirement(%) with the scale of 1/4.

channel needed for the optimal patching scheme. PSRB also requires less client storage at this cache capacity as indicated in Fig. 9(c). And yet, PSRB achieves more than 10 percentage points of traffic reduction compared to the optimal patching scheme. For the SRB algorithm, it generally achieves the second best traffic reduction with even less penalty on client channel and storage requirements.

The performance gain for the VOD workload is larger than that of the WEB workload, correspondingly. This is mainly due to the longer streaming session on average in the VOD workload. We omit it for brevity; interested readers can refer to [21].

2) Partial Viewing of Web Workload: In streaming media delivery over the Web, it is possible that some clients terminate the session after watching for a while from the beginning of the media object. It is important to evaluate the performance of the proposed algorithm under this situation. Using the PARTIAL workload as defined in Section IV, we perform the same simulations and evaluate the same set of parameters. Fig. 10(a) shows similar characteristics as those in Fig. 8. PSRB and SRB still achieve better traffic reduction. The conclusion holds that PSRB uses 60% of the client channel to achieve five percentage points better traffic reduction compared with the optimal patching.

In the event that a session terminates before it reaches the end of the requested media object, it is possible that the client has already downloaded future part of the media stream which is no longer needed. To characterize this wasted delivery from the proxy to the client, we record *average client waste*. It is the percentage of wasted bytes versus the total prefetched data.

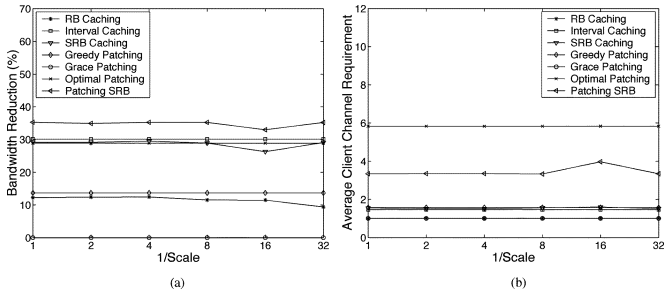


Fig. 10. PARTIAL: (a) bandwidth reduction and (b) average client channel requirement with 1-GB proxy memory.

Fig. 11 shows the client waste statistic. Note that for PARTIAL and REAL workloads, since both contain premature session terminations, the prefetched data which is not used in the presentation are not counted as bytes hit in the calculation of the *server traffic reduction*.

As shown in Fig. 11(b), PSRB and SRB have about 42% and 15% of prefetched data wasted compared with 0% for interval caching. Since the wasted bytes are not counted as hit, this leads to the lowered traffic reduction rate for PSRB and SRB compared to that of interval cache. From another perspective, interval caching does not promote sharing among buffers; hence, the client listens to one channel only and there is no buffering of future data. Thus, there is no waste in proxy-to-client delivery in the event of premature session termination.

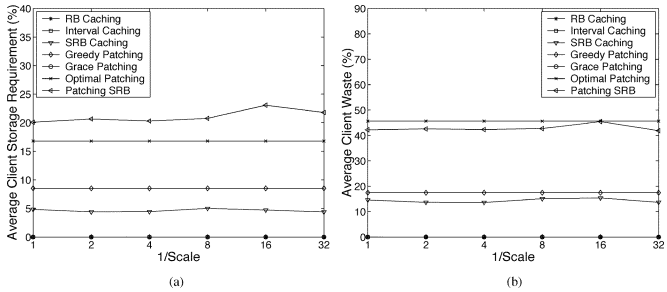


Fig. 11. PARTIAL: (a) average client storage requirement (%) and (b) client waste (%) with 1-GB proxy memory.

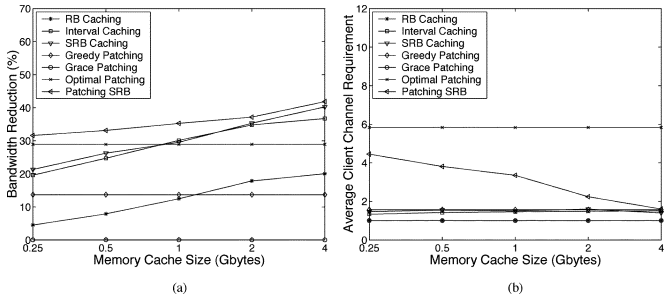


Fig. 12. PARTIAL: (a) bandwidth reduction and (b) average client channel requirement with the scale of 1/4.

We now again start investigation of the caching performance with a fixed *scale* factor for the initial buffer size in Fig. 12. Compared with Fig. 8(a), the distances between the traffic reduction curves between PSRB, SRB and interval caching become much smaller in general. This reinforces the observation above that PSRB and SRB may lead to more wasted bytes in the partial viewing cases. In addition, the grace patching achieving almost no traffic reduction shows its incapability in dealing with the partial viewing situation. The reason might be that the new session started by the grace patching, which is supposed to be a complete session, often terminates when 20% of the media object is delivered. Since the duration of the session is short, it is less likely that a new request to the same media object is received.

In Fig. 12(b), PSRB demonstrates monotonic decreasing of average client channel requirement when memory capacity increases. This is due to the fact that there is a fewer number of zero-sized running buffers with increasing proxy memory capacity. Similarly, as shown in Fig. 13, the client storage requirement and average client waste also decrease since a fewer number of patching is required.

C. Performance on REAL Workload

Based on a real video delivering workload captured from corporate intranet, the same simulations are conducted to evaluate the caching performance. We start first by evaluating the caching performance versus varying *scale* factor for the initial buffer size.

The server side traffic reduction is shown in Fig. 14(a) while the client side statistics are shown in Fig. 14(b) and Fig. 15. Comparing Fig. 14(a) with Fig. 8(a), it is clear that the difference in the *scale* factor has a much more significant impact on the performance of the proposed SRB and PSRB algorithms for

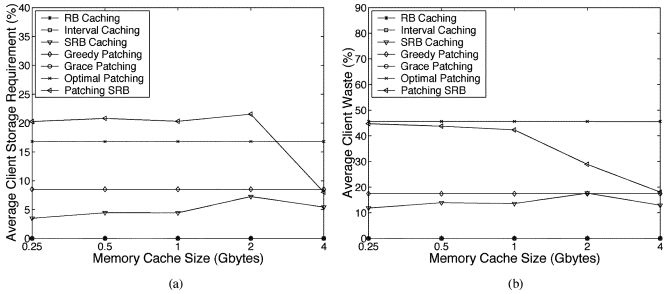


Fig. 13. PARTIAL: (a) average client storage requirement (%) and (b) client waste (%) with the scale of 1/4.

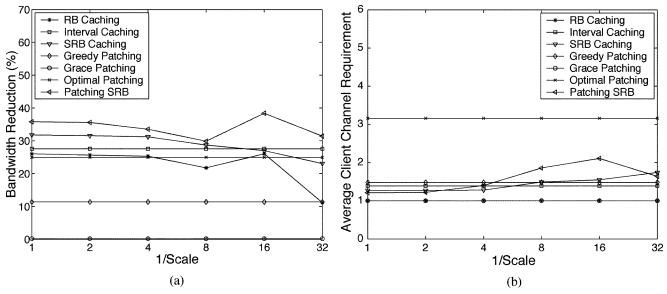


Fig. 14. REAL: (a) bandwidth reduction and (b) average client channel requirement with 1-GB proxy memory.

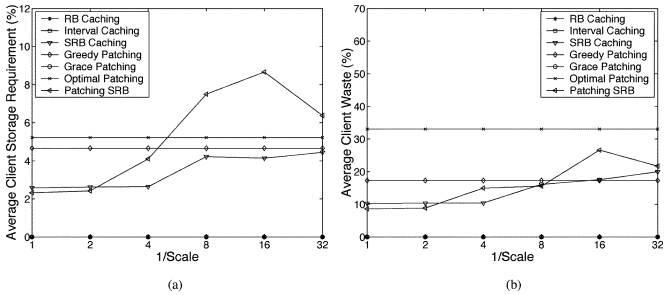


Fig. 15. REAL: (a) average client storage requirement (%) and (b) client waste (%) with 1-GB proxy memory.

REAL. This could be due to the bursty nature of the accesses logged in the workload. To a certain extent, this result indicates the effectiveness of the adaptive buffer allocation scheme we proposed in the algorithms.

Setting the initial buffer size as 1/4 of the requested media objects, we again evaluate the caching performance with increasing amount proxy memory available. Figs. 16 and 17 show the server traffic reduction and the client side statistics. Compared with the simulation results obtained with synthetic workloads, Fig. 16(a) shows a flat gain when memory capacity increases. It seems to indicate that memory capacity is less of a factor. Once again, the bursty nature of the request arrival may play a role here. In addition, the volume of the burst may also be low which leads to the fact that limited amount of memory space suffices the sharing of sessions.

The simulation results for the real workload provide the following understanding for the studying of caching of streaming media. *Contrary to the intuition that the caching of streaming media requires large memory space, our study using the synthetic and real workloads shows that the user-access pattern based buffer allocation and sharing policy is critical to achieve*

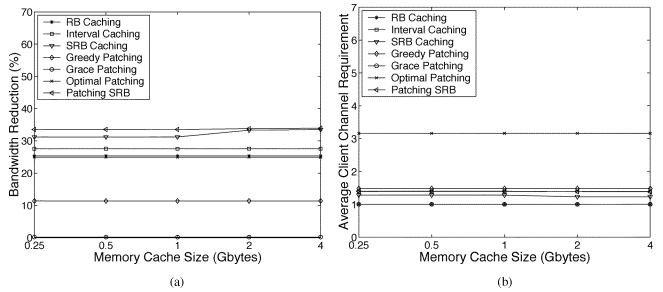


Fig. 16. REAL: (a) bandwidth reduction and (b) average client channel requirement with the scale of 1/4.

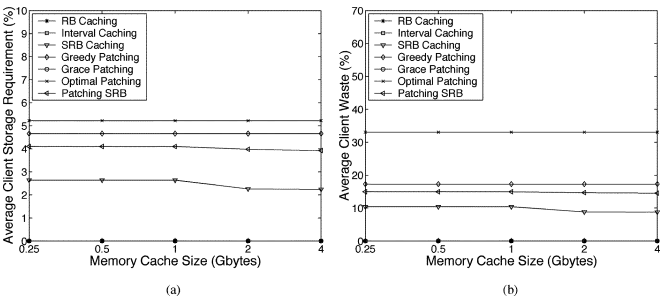


Fig. 17. REAL: (a) average client storage requirement (%) and (b) client waste (%) with the scale of 1/4.

good caching performance with a limited memory resource. This is also the motivation of the proposed SRB and PSRB algorithms.

D. Further Analysis on Client Channel Requirement

The performance analysis in the previous section indicates that the SRB and PSRB algorithms achieve superior server traffic reduction by utilizing the memory resource on the proxy and sufficient bandwidth resource between the proxy and the clients. In most cases, the proxy streams data from multiple buffers to the client through multiple channels. To have a better understanding on the client channel requirement, we collect additional statistics that illustrates the distribution of the number of client channels required. Figs. 18 and 19 show the CDF of the client channel requirement for simulations on four workloads. In these simulations, the proxy has 1-GB memory capacity and the *scale* factor for the initial buffer size is fixed at 1/4.

For simple running buffer caching, only one channel is required for a client since there is no session sharing. Greedy and grace patching algorithms need at most two client channels. For WEB and VOD workloads, approximately 60% of greedy patching sessions and 40% of grace patching sessions require only one client channel. Interval caching also requires at most two client channels with 78% of the sessions requiring only one channel.

Optimal patching needs the largest number of client channels. It is not surprising since requests arrive later always try to patch to as many earlier on-going sessions as possible. Among the simulation results of all four workloads, the number of client channel required could exceed nine for the optimal patching scheme.

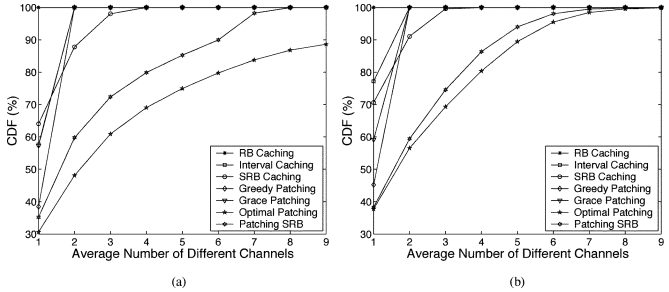


Fig. 18. Client channel requirement CDF: (a) WEB and (b) VOD.

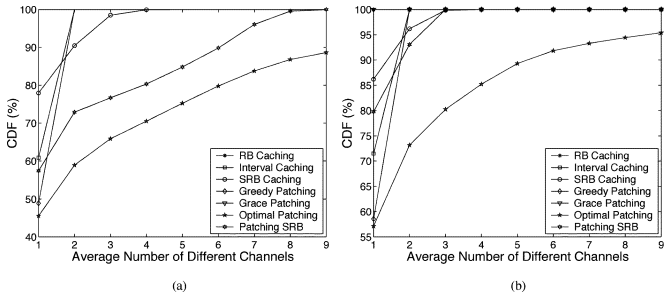


Fig. 19. Client channel requirement CDF: (a) PARTIAL and (b) REAL.

For the proposed SRB and PSRB algorithms, the number of the required client channel often falls in between that of the optimal patching and the group of algorithms containing greedy, grace patching and interval caching. Note further that for SRB algorithm, very few sessions require more than three client channels with around 98% of session requires no more than two. The statistics shown for in the REAL workload as in Fig. 19(b) verifies further that 94% of the PSRB sessions needs no more than two client channels. On the other hand, more than 10% of the optimal patching sessions needs three or more client channels. Referring back to Fig. 16(a), it is clear that SRB and PSRB algorithms achieve higher server traffic reduction rate than the optimal caching but pay less penalty in proxy-to-client channel requirement. This analysis enhances the advantages of the proposed algorithms. In addition, these observations are useful when limited bandwidth resource is available between the proxy and the client. In this case, the proxy system can choose to execute a session sharing algorithm which achieves better caching performance without exceeding the proxy-to-client link capacity.

Based on the evaluation results, these SRB based algorithms can be used not only for the media delivery in the VOD environment, but also for processing the increasing amount of streaming media objects on the Web today to improve the performance on the client side (such as the playback jitter, the startup delay) eventually. Different from the SRB work in this paper, we have also thoroughly evaluated the client playback jitter and startup delay in [22] and [23], which are important for streaming delivery systems from the client point of view.

VI. Conclusion

In this paper, we propose two new algorithms, SRB and PSRB, for caching of streaming media objects, by maximizing the available memory resource utilization in the existing proxies

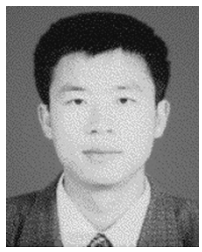
to exploit the temporal locality of client accesses, thus to relieve the bottlenecks of disk bandwidth and network bandwidth for streaming media delivery. The SRB caching algorithm dynamically caches media objects in the proxy memory during delivery while the PSRB algorithm further enhances the memory utilization in the proxy. Extensive simulations using both synthetic and real workloads are conducted. The simulation results demonstrate they can greatly reduce the network traffic or the disk bandwidth. Although both algorithms require the client capable of listening to multiple channels at the same time, the proposed algorithms achieve higher server traffic reduction rate with less or similar load on the link between the proxy and the client when compared with previous solutions which also require multiple client channels.

ACKNOWLEDGMENT

The authors appreciate the critical and constructive comments from the anonymous referees. They thank Dr. M. Trott for providing detailed corrections on the text and valuable suggestions for future work. The preliminary results were presented in [24].

REFERENCES

[1] A. Luotonen, H. F. Nielsen, and T. Berners-Lee. Cern httpd. [Online] Available: http://www.w3.org/Daemon/Status.html
[2] C. Bowman, P. Danzig, D. R. Hardy, U. Manber, M. Schwartz, and D. Wessels, "Harvest: A Scalable, Customizable Discovery and Access System,", Tech. Rep. CU-CS-732-94, 1994.
[3] Squid Web Proxy Cache.. [Online] Available: http://www.squid-cache.org/
[4] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proc. IEEE INFOCOM*, New York, Mar. 1999.
[5] K. Wu, P. S. Yu, and J. Wolf, "Segment-based proxy caching of multimedia streams," in *Proc. WWW*, **AUTHOR: PLS. SUPPY MORE INFORMATION** Sep. 2001.
[6] 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.
[7] A. Dan and D. Sitaram, "A generalized interval caching policy for mixed interactive and long video workloads," in *Proc. Multimedia Computing and Networking*, San Jose, CA, Jan. 1996.
[8] D. P. Anderson, Y. Osawa, and R. Govindan, "A file system for continuous media," *ACM Trans. Computer Systems (TOCS)*, vol. 10, no. 4, Nov. 1992.
[9] F. Tobagi, J. Pang, R. Baird, and M. Gang, "Streaming raid: a disk array management system for video files," in *Proc. ACM Multimedia*, Anaheim, CA, Aug. 1993.
[10] Y. Chae, K. Guo, M. Buddhikot, S. Suri, and E. Zegura, "Silo, rainbow, and caching token: Schemes for scalable fault tolerant stream caching," *IEEE J. Select. Areas Commun.*, vol. 20, no. 7, pp. 1328–1344, Sep. 2002.
[11] S. Lee, W. Ma, and B. Shen, "An interactive video delivery and caching system using video summarization," *Comput. Commun.*, vol. 25, pp. 424–435, Mar. 2002.
[12] K. A. Hua, Y. Cai, and S. Sheu, "Patching: a multicast technique for true video-on-demand services," in *Proc. ACM Multimedia*, Bristol, U.K., Sep. 1998.
[13] S. Sen, L. Gao, J. Rexford, and D. Towsley, "Optimal patching schemes for efficient multimedia streaming," in *Proc. ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Basking Ridge, NJ, Jun. 1999.
[14] L. Gao and D. Towsley, "Supplying instantaneous video-on-demand services using controlled multicast," in *Proc. IEEE Int. Conf. Multimedia Computing and Systems*, Florence, Italy, Jun. 1999.
[15] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proc. ACM Multimedia*, San Francisco, CA, Oct. 1994.
[16] N. Fonseca and R. eFaCanha, "The look-ahead-maximize-batch batching policy," *IEEE Trans. Multimedia*, vol. 4, no. 1, pp. 114–120, Mar. 2002.
[17] A. Dan and D. Sitaram, "Buffer management policy for an on-demand video server," IBM Res. Rep. 19 347, 1993.
[18] 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.
[19] 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.
[20] C. C. Aggarwal, J. Wolf, and P. Yu, "A permutation-based pyramid broadcasting scheme for video-on-demand systems," *Proc. Int. Conf. Multimedia Computing and Systems*, Jun. 1996.
[21] S. Chen, B. Shen, Y. Yan, and S. Basu, "Srb:the shared running buffer based proxy caching of streaming sessions," Hewlett-Packard Laboratories, Tech. Rep. HPL-2003-47, 2003.
[22] S. Chen, B. Shen, S. Wee, and X. Zhang, "Adaptive and lazy segmentation based proxy caching for streaming media delivery," in *Proc. ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Monterey, CA, Jun. 2003.
[23] ——, "Designs of high quality streaming proxy systems," in *Proc. IEEE INFOCOM*, Hong Kong, China, Mar. 2004.
[24] S. Chen, B. Shen, Y. Yan, S. Basu, and X. Zhang, "Srb: shared running buffers in proxy to exploit memory locality of multiple streaming sessions," in *Proc. 24th Int. Conf. Distributed Computing Systems (ICDCS)*, Tokyo, Japan, Mar. 2004.



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 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, Nanjing, China, and the Ph.D. degree in computer science from Wayne State University, Detroit, MI.

He is currently with Hewlett-Packard Laboratories, Palo Alto, CA, where he is a Senior Research Scientist in the Streaming Media System Group. His research interests include image/video processing, multimedia networking, and content distribution systems. He has published over 30 papers in refereed journals and conferences. He holds three U.S. patents, with many pending.

Dr. Shen has served as a program committee member in a number of technical conferences in multimedia and distributed system areas.

Yong Yan received the B.S. and M.S. degrees in computer science from Huazhong University of Science and Technology, Huazhong, China, in 1984 and 1987, respectively, and the Ph.D. degree in computer science from the College of William and Mary, Williamsburg, VA, in 1998.

He was a System Architect with HAL Computer Systems Inc. from 1998 to 1999. He then joined the Mobil and Media Systems Laboratory, Hewlett–Packard Laboratories, Palo Alto, CA, as a Research Scientist from 1999 to 2003. Currently an independent consultant, his interests are in the areas of parallel and distributed computing, performance evaluation, operating systems, and algorithms analysis.

Sujoy Basu received the B.Tech. degree in computer science from Georgia Institute of Technology, Atlanta, the M.S. degree computer science from IIT, Kanpur, India, and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign.

He has been a member of the Mobile and Media Systems Laboratory, Hewlett-Packard Laboratories, Palo Alto, CA, since 2000. His current research spans networking, web services, and grid computing. He has also worked on processor and system architecture and proxy caching.

Xiaodong Zhang (SM'94) received the Ph.D. degree in computer science from the University of Colorado at Boulder.

He is the Robert M. Chritchfiled Professor of Engineering at The Ohio State University, Columbus, and the Chairman of Computer Science and Engineering Department. He was the Lettie Pate Evans Professor of Computer Science and the Department Chair at the College of William and Mary, Williamsburg, VA. His research interests include high performance computing and systems.

Dr. Zhang was the Program Director of Advanced Computational Research at the National Science Foundataion, 2001-2004. He is an associate editor of the IEEE TRANSACTIONS ON COMPUTERS and *IEEE Micro*.