Shared Running Buffer Based Proxy Caching of Streaming Sessions

Songqing Chen[1], Bo Shen, Yong Yan, Sujoy Basu
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2003-47
March 10th , 2003*

E-mail: sqchen@cs.wm.edu, {boshen, yongyan, basus}@hpl.hp.com

shared running buffer, proxy caching, patching, streaming media delivery, VOD

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 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 (SRB)* 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.

* Internal Accession Date Only Approved for External Publication
[1] Department of Computer Science, College of William and Mary, Williamsburg, VA 23188

Shared Running Buffer Based Proxy Caching of Streaming Sessions

Songqing Chen *
Department of Computer Science
College of William and Mary
Williamsburg, VA 23188
sqchen@cs.wm.edu

Bo Shen, Yong Yan and Sujoy Basu
Mobile and Media System Lab
1501 Page Mill Road
Palo Alto, CA 94304
{boshen,yongyan,basus}@hpl.hp.com

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 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 (SRB)* 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.

Key Words: Shared Running Buffer, Proxy Caching, Patching, Streaming Media, VOD.

1 Introduction

The building block 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 [16], Harvest [4] and Squid [14].

The caching of streaming media content presents a different set of challenges: (i) The size of a streaming media object is usually orders of magnitudes larger than traditional web contents. For

*Work performed as an intern at HP Labs, summer 2002.

example, a two-hour long MPEG video requires about 1.4 GB of disk space, while traditional web objects are of the magnitude of 10 KB; (ii) 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 [18] or segments [20] 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 [3] or an interval [9] 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 2.

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 (i) adaptively allocates memory space according to the user access pattern; (ii) enables maximal sharing of the cached data in memory; (iii) optimally reclaims memory space when requests terminate; (iv) 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.

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 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 2, the related work is surveyed. Section 3 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 4. Performance evaluations are conducted in Section 5. We then make concluding remarks in Section 6.

2 Related Work

In this section, we survey previous work related to the caching of streaming media content. Three types of methods are investigated. First, we briefly discuss streaming media caching methods that

use the storage resource on the proxy. These methods cache media objects for streaming sessions that are not overlapped in time. The cached data persists in cache even after session termination. Usually, a long-term storage resource such as disk storage is used. Second, we investigate caching methods that use the storage resource on the client device. These methods share streaming sessions that are overlapped in time. However, the proxy does not buffer data for any session. Lastly, we study caching methods that use the storage resource on the proxy to buffer portions of media objects for streaming sessions that are overlapped in time. Note that the buffered data persists in cache only when the sessions are alive. Therefore, short-term storage resource such as RAM is used.

2.1 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 [2, 19]. Two typical types of partial caching have been investigated.

Prefix caching [18] 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. [20] 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 [5]. Lee et al. [15] uses a context-aware segmentation so that segments of user interest are cached.

2.2 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 video on demand (VOD) systems.

Patching algorithms [13] 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 [17] further assumes sufficient storage resource at the client side to receive as much data as possible while listening to as many channels as possible. Figure 1 shows the basic ideas of the patching techniques through examples.

In these figures, media position indicates a time position at which offset the media object is delivered

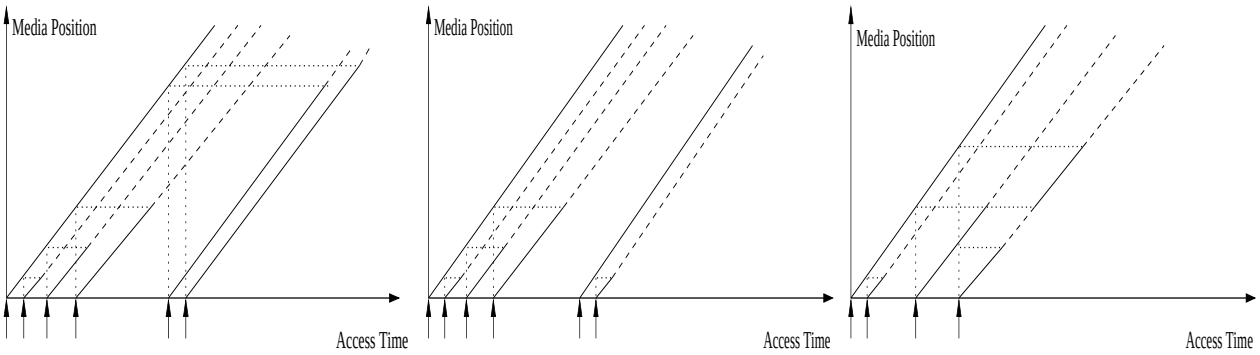

Figure 1: Left to right, (a) Greedy Patching, (b) Grace Patching, (c) Optimal Patching.

to the client. For illustration purpose, we assume instantaneous delivering of media content at a constant rate equal to the presentation rate of the media object. That is, if a request is served immediately, the client receives n time units of data after n time units since the request is initiated. The solid lines represent a streaming session between the server and the proxy. The goal of these algorithms is to reduce the server-to-proxy traffic.

A greedy patching example is shown in Figure 1(a). 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 as shown in the last two requests in Figure 1(a). It is more efficient to start a new session even before the previous full session terminates. The grace patching example shown in Figure 1(b) indicates that 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 [12].

The optimal patching scheme is introduced in [17]. The basic idea is that later sessions patch to as many on-going sessions as possible. The on-going sessions can be a full session or a patching session. Figure 1(c) shows an example scenario. 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 the 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 [10, 11]. 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.

2.3 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.

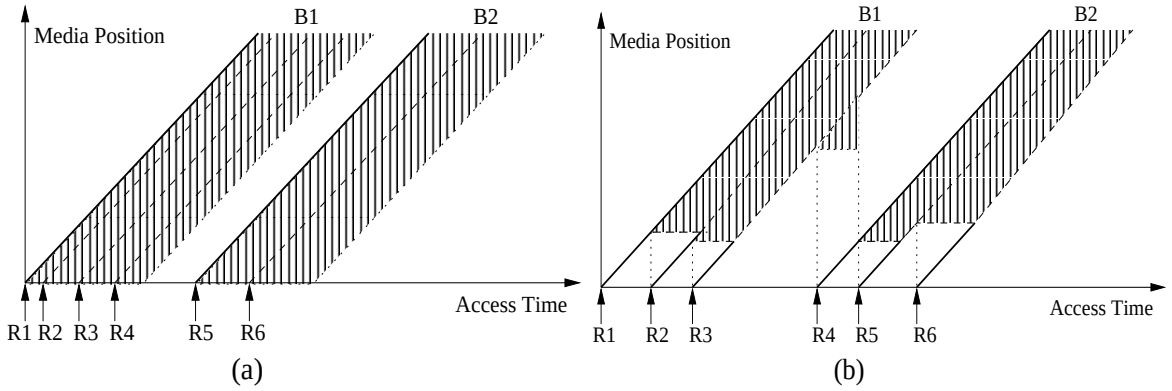

Figure 2: (a) Running Buffer, (b) Interval Caching.

Dynamic caching [3] 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). Figure 2(a) shows an example of running buffer caching. A fix-sized buffer is allocated upon the arrival of request R_1. The buffer is filled by the session started by R_1 and run to the end of stream (the paralleled vertical lines indicate the buffer fullness and the location of the buffered data relative to the beginning of the media object). Subsequent requests R_2, R_3, R_4 are served from the buffer since they arrived within the time period covered by the buffer. Request R_5 arrives beyond the range covered by the first buffer, hence a second buffer of the same predetermined size is allocated. Subsequently, R_6 is served from the second buffer.

On a different approach, interval caching [8, 9] 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. Figure 2(b) shows an example of the interval caching method. Upon the arrival of request R_2, an interval is formed, and a buffer of the size equivalent to the interval is allocated. The buffer is filled by the session started by R_1 from this point on. The session initiated by R_2 only needs to receive the first part of data from the server (the solid line starting from R_2). It can receive the rest of data from the buffer. The same applies when R_3 and R_4 arrive. The situation changes until the arrival of R_5. Since the interval between R_4 and R_5 is smaller than that between R_3 and R_4, the buffer initially allocated for the interval of R_3 and R_4 (not filled up yet) is released, and the space is re-allocated for the new interval of R_4 and R_5. Now the session for R_4 has to run to the end of the stream.

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.

3 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 [3] and interval caching [8, 9], do not make effective use of the limited memory resource. Motivated by the limits of the current memory buffering approaches, we design a Shared Running Buffer (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: Patching SRB (PSRB).

3.1 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 j-th 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 as 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 & 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 as the *resident requests* of this buffer and the buffer is called as 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:

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

   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 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:

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

   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.

3.2 Shared Running Buffer (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 3.2.3) 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 Eq. (3). During the process of the SRB algorithm, parts of a running buffer could be dynamically reclaimed as described in Section 3.2.2 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 3.2.1.

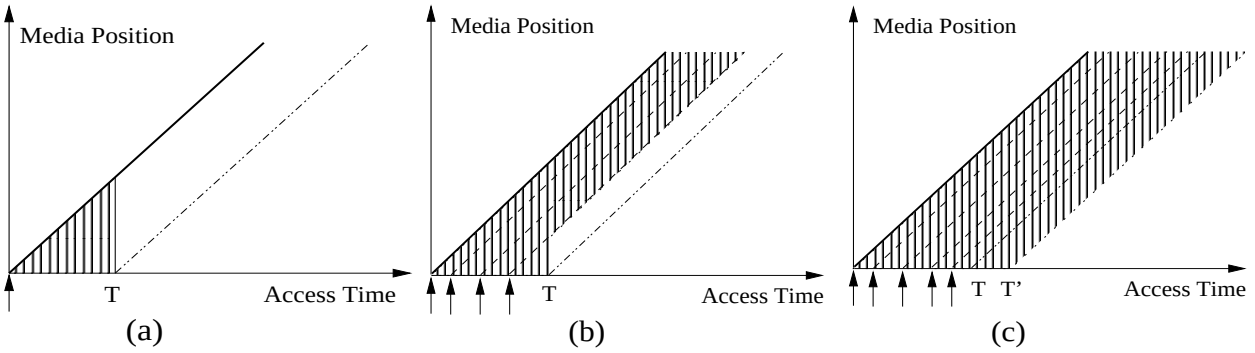

Figure 3: SRB Memory Allocation: the Initial Buffer Freezes its Size (a) (b) (c)

3.2.1 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. Figure 3(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. Figure 3(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. Figure 3(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.

  Figure 4 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.**

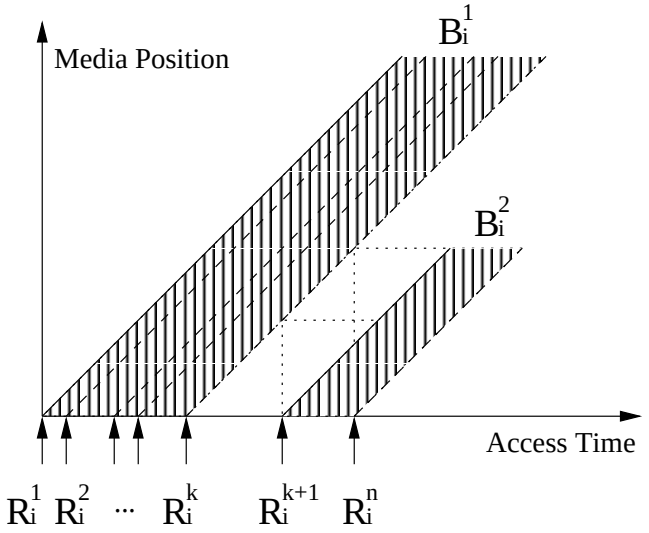

Figure 4: Multiple Running Buffers

  The *Running-Distance* and *End-Time* are determined based on Eq. (2) and Eq. (3) respectively for any buffer entering the *Running State*. In addition, the *End-Time*s of its preceding running buffers need to be modified according to the arrival time of the latest request, as shown in Eq. (3). Figure 5 shows an example scenario for the updating of *End-Time*s. Note that E_i^2 is updated when B_i^3 is started, at which time the running distance of B_i^2 is extended (dashed line). When a buffer runs to its *End-Time*, it enters the *Idle State* (case 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 minute to 10 minutes) 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.

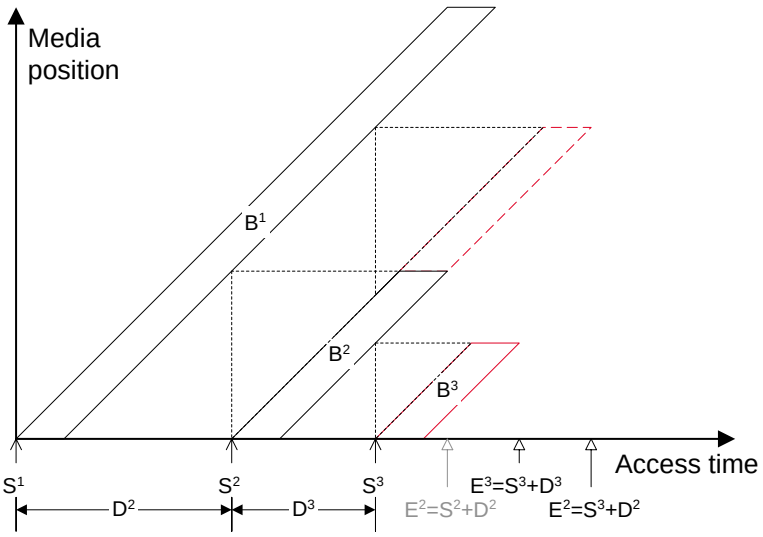

Figure 5: Example on Updating of End-Time

3.2.2 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. Figure 6(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 Figure 6(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 Figure 6(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 Figure 6(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 Figure 6(c').

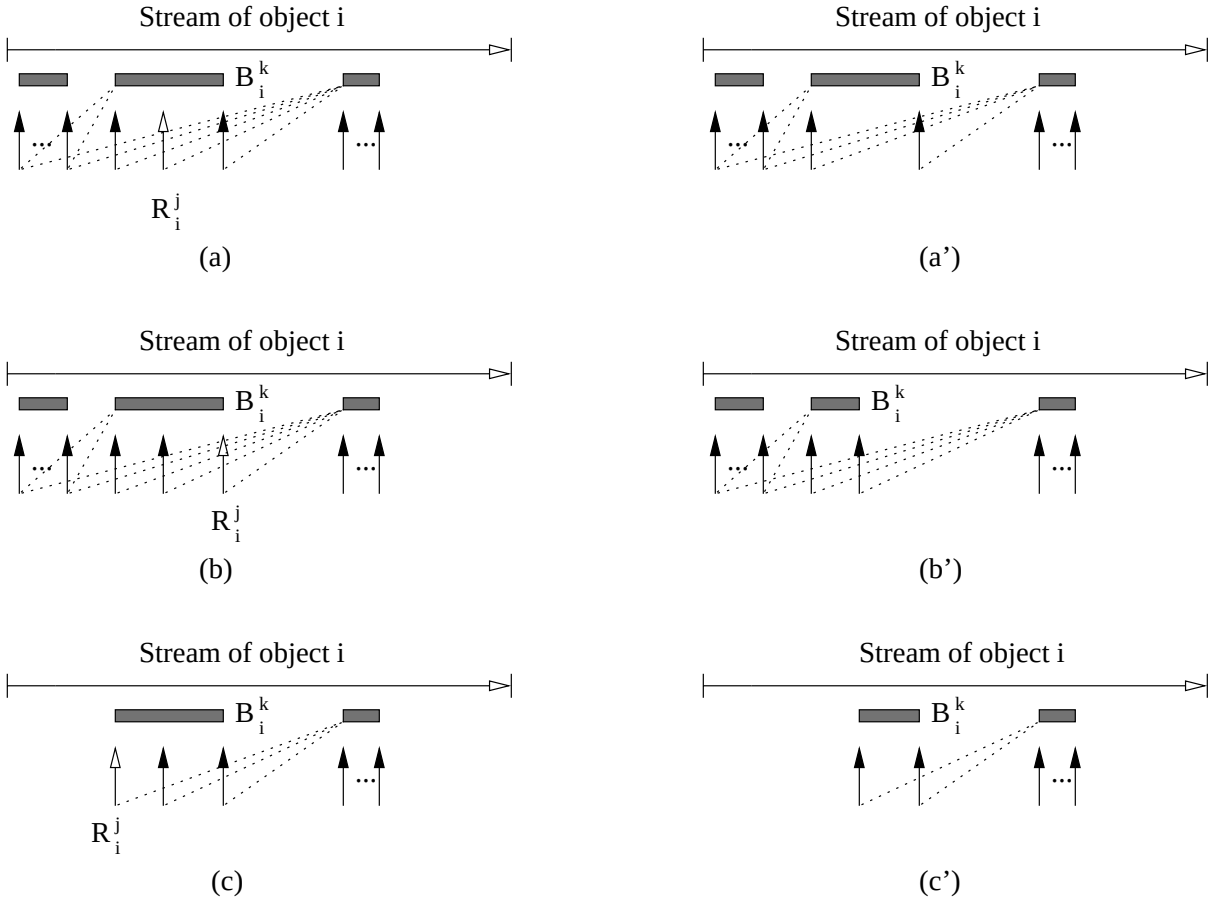

Figure 6: SRB Memory Reclamation: Different Situations of Session Termination

3.2.3 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 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 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 non-replacement policy. We have evaluated the performances of both two polices by simulations in the later section. It is shown that a straightforward non-replacement police may achieve similar performance given long enough system running time.

3.3 Patching SRB (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 patching SRB (PSRB) algorithm which enables the sharing of such bypass sessions. This is related to the session sharing algorithms as discussed in Section 2.2. It is important to note that PSRB scheme makes the memory-based SRB algorithm work with the memoryless patching algorithm.

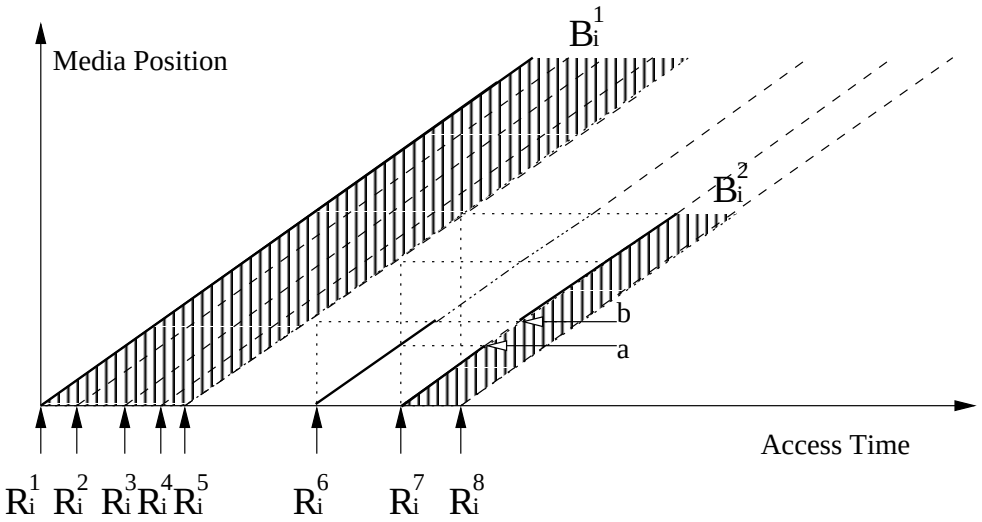

Figure 7: PSRB Caching Example

Figure 7 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 Figure 7, 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 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.

4 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 length of the video varies from short ones to longer ones. The second simulates the video access in a video-on-demand (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 ($p(x, \lambda) = e^{-\lambda} * (\lambda)^x/(x!), x = 0, 1, 2...$).

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^{TM}, 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 [6].

4.1 Workload Characteristics

Table 1 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 minutes to 2 hours. The request inter-arrival follows a Poisson distribution with $\lambda = 4$ seconds and $\alpha = 0.47$ (according to[7]) of Zipf-like distribution for object popularities. The media objects are coded and streamed at 256 Kbps. The total number of requests is 15188. The simulation lasts 87114 seconds or roughly 24 hours. The *Low-Bound* and *High-Bound* for the initial buffer size are set as 2 MB 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

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

Table 1: Workload Statistics

of WEB as shown in Table 1.

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 hour to 2 hours. The request inter-arrival follows a Poisson distribution with $\lambda = 60$ seconds. The Zipf-like distribution for object popularity is set as $\alpha = 0.73$ (according to [1]). The media objects are coded and streamed at 2 Mbps. The total number of requests is 10731. The simulation runs for 654539 seconds or roughly a week. The *Low-Bound* and *High-Bound* for initial buffer allocation are set as 16 MB and 128 MB respectively.

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 916427 seconds or roughly 10 days. Figure 8(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 minutes. Figure 8(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.

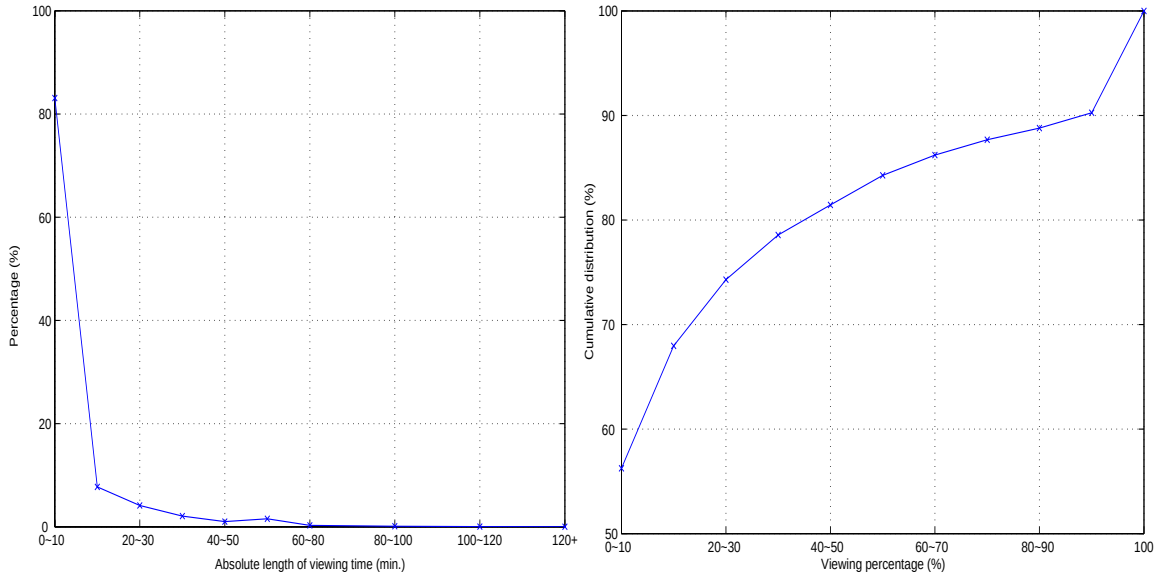

Figure 8: Real workload (left to right), (a) Distribution of absolute viewing time (b) distribution of viewing percentage.

4.2 Potentially Cachable Content

To find out the amount of potentially cachable content, we evaluate the number of overlapped sessions at each time instance during the simulation. Specifically, at one point in time, if a session is streaming a media object similar to at least another session, it is called a *shared* session. Note that *shared* sessions may be streaming different portions of the media object. If a session is streaming a media object that no other session is streaming, it is called a *not-shared* session. Figures 9 and 10 show the accumulated number of sessions in two categories during full or part of the simulations for the four workloads.

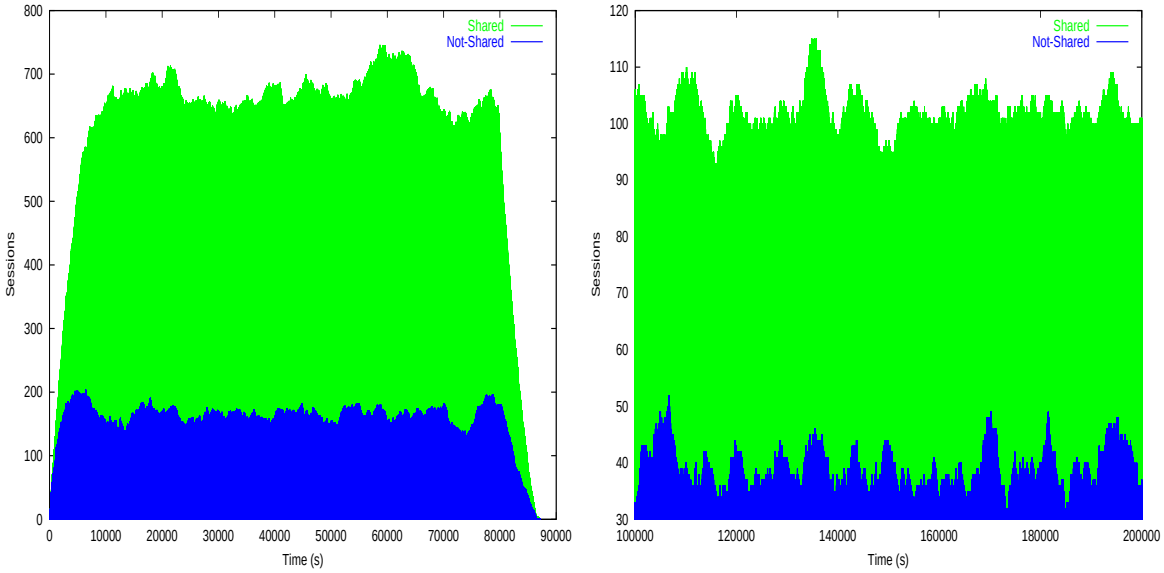

Figure 9: Potentially Cachable Content: WEB (left) and VOD (right) workloads

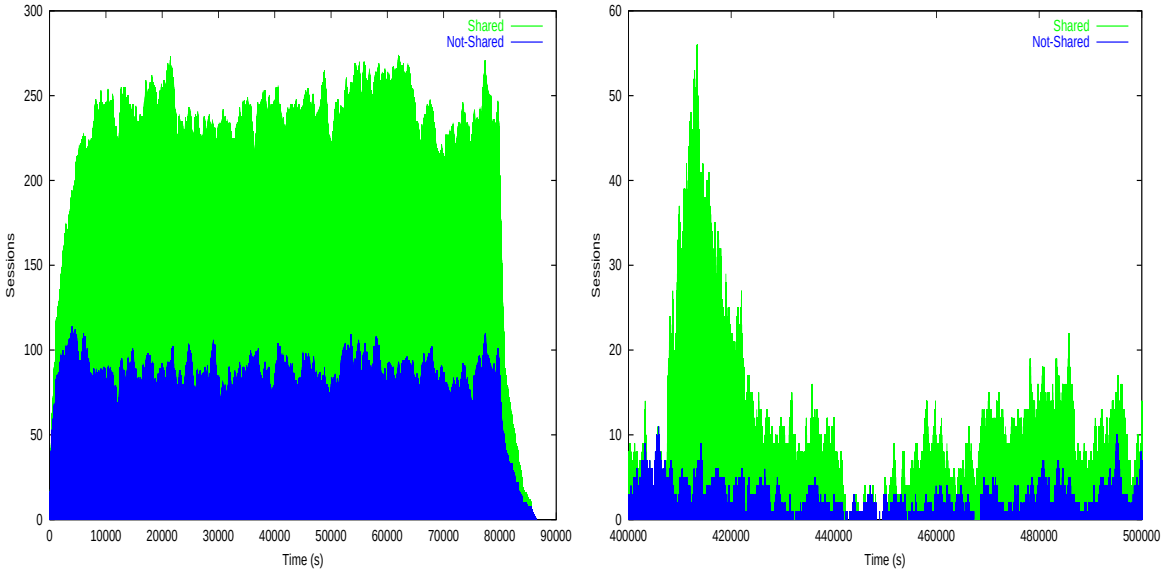

Figure 10: Potentially Cachable Content: PARTIAL (left) and REAL (right) workloads

At the first sight of these figures, it seems that the potential cachable contents of different workloads

in the decreasing order should be WEB, VOD, PARTIAL and REAL. But be aware that only WEB and PARTIAL show their whole durations in the figures, while VOD and REAL only show a part of their full durations since their durations are too large to be shown on these figures. Recall that the duration of PARTIAL is 7 days, while the duration of REAL is 10 days. Combining these factors, we expect that the WEB workload presents the greatest amount of potentially cachable contents, followed by VOD, REAL and PARTIAL workloads in that order.

4.3 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 on-going 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 is 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. Figure 11(a) shows the distribution in full range on the number of sharing requests, and (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.

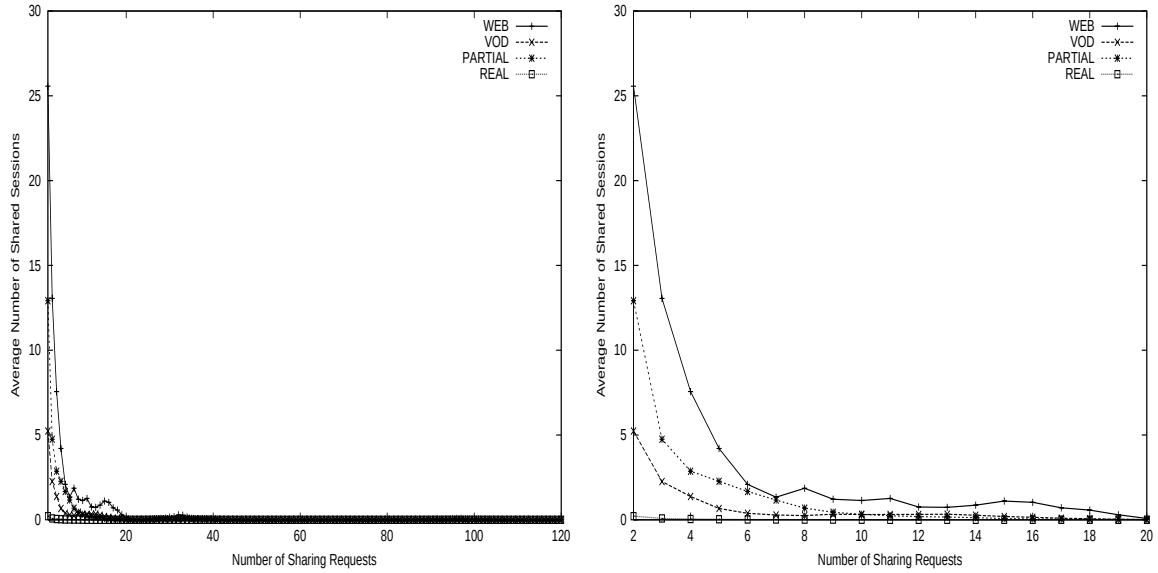

Figure 11: Shared request histogram. (a) full, (b) part.

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.

5 Performance Evaluation

5.1 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.

5.2 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.

5.2.1 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 1GB, 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 Figure 12.

Figure 12(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.

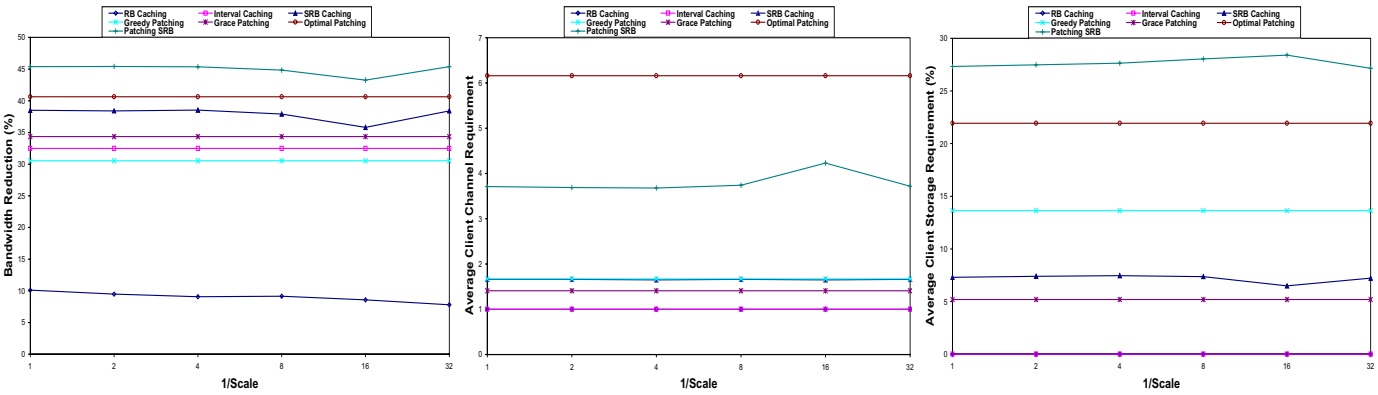

Figure 12: WEB workload (left to right): (a) Server Traffic Reduction, (b) Average Client Channel and (c) Storage Requirement(%) with 1GB Proxy Memory

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.

Figure 12 (b) and (c) show 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∼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 6 percentage point less traffic reduction than PSRB, but the requirement on client channel and storage is significantly lower.

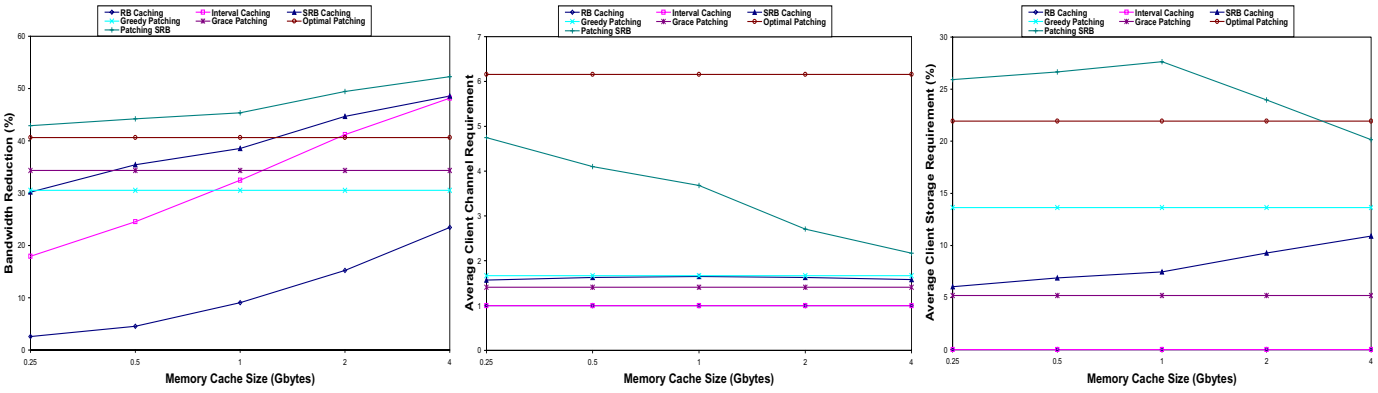

Figure 13: WEB workload (left to right): (a)Bandwidth Reduction, (b)Average Client Channel and (c)Storage Requirement(%) with the scale of 1/4

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. Figure 13 (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 Figure 13(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 channel needed for the optimal patching scheme. PSRB also requires less client storage at this cache capacity as indicated in Figure 13(c). And yet, PSRB achieves more than 10 percentage points of traffic reduction comparing 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.

5.2.2 Complete Viewing Workloads of VOD Objects

To further evaluate the performance of the proposed algorithm in a video delivery environment with relatively longer streaming sessions, we use the VOD workload for the simulation. Compared with the result we obtained in the WEB workload simulation, the simulation on VOD workload demonstrates similar trade-offs with respect to varying scale factors. This is evident as shown in Figure 14.

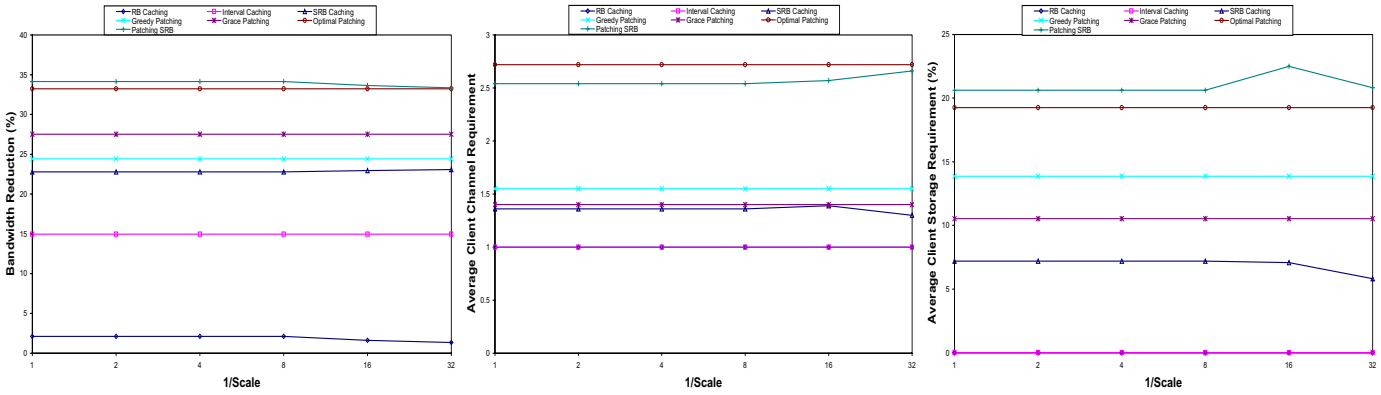

Figure 14: VOD workload (left to right): (a) Bandwidth Reduction, (b) Average Client Channel and (c) Storage Requirement(%) with 1GB Proxy Memory

With a fixed *scale* factor for the initial buffer size, Figure 15 (a) illustrates similar performance gain as achieved for the WEB workload. In addition, similar penalties are imposed on each algorithm as is evident from Figure 15 (b) and (c).

Note that the performance gain for the VOD workload is larger than that of the WEB workload. This is mainly due to the longer streaming session on average in the VOD workload. A longer streaming session provides more opportunity for multiple clients to share the session.

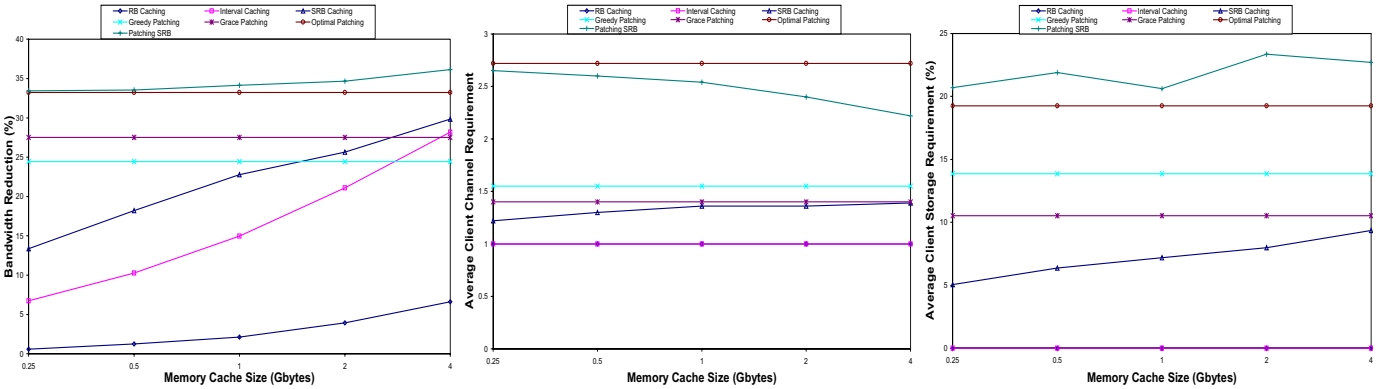

Figure 15: VOD workload (left to right): (a) Bandwidth Reduction, (b) Average Client Channel and (c) Storage Requirement(%) with the Scale of 1/4

5.2.3 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 4, we perform the same simulations and evaluate the same set of parameters. Figure 16(a) shows similar characteristics as that in Figure 12. PSRB and SRB still achieves better traffic reduction. The conclusion holds that PSRB uses 60% of the client channel to achieve 5 percentage point better traffic reduction compared with the optimal patching.

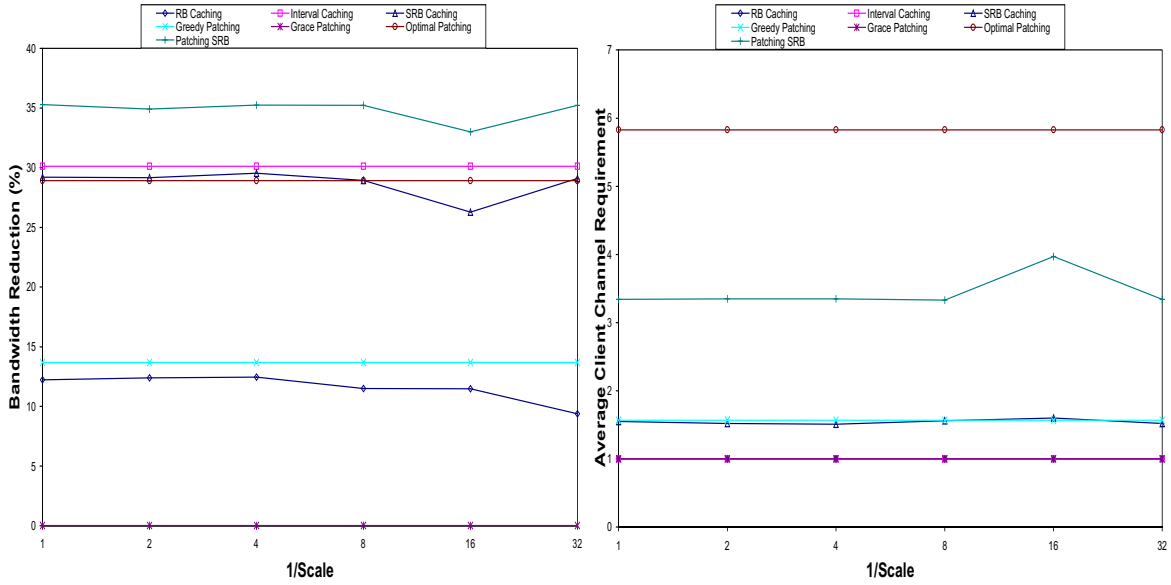

Figure 16: PARTIAL workload (left to right): (a) Bandwidth Reduction and (b) Average Client Channel Requirement with 1GB Proxy Memory

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. Figure 17 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*.

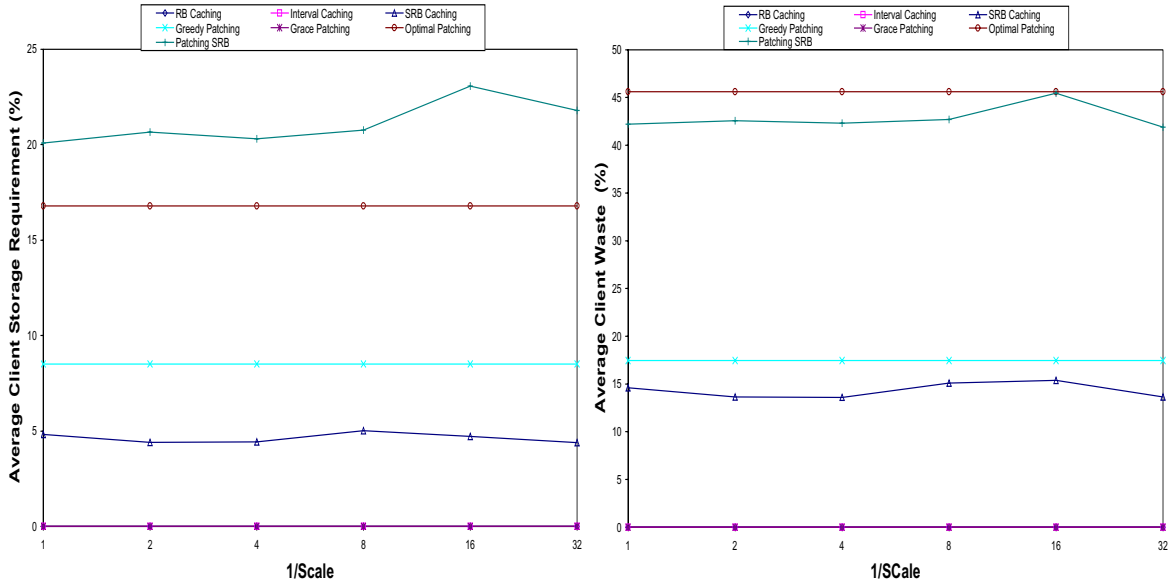

Figure 17: PARTIAL workload (left to right): (a) Average Client Storage Requirement(%) and (b) Client Waste(%) with 1GB Proxy Memory

As shown in Figure 17(b), PSRB and SRB have about 42% and 15% of prefetched data wasted comparing 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 comparing 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.

We now again start investigation of the caching performance with a fixed *scale* factor for the initial buffer size in Figure 18. Comparing with Figure 12(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 Figure 18(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 Figure 19, the client storage requirement and average client waste also decrease since a fewer number of patching is required.

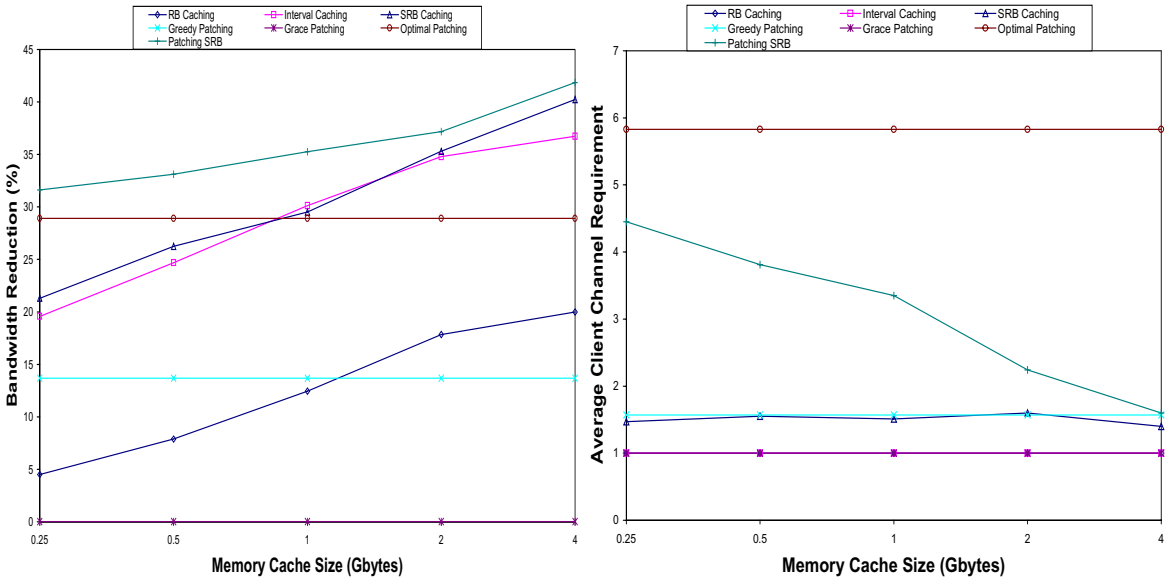

Figure 18: PARTIAL workload (left to right): (a) Bandwidth Reduction and (b) Average Client Channel Requirement with the Scale of 1/4

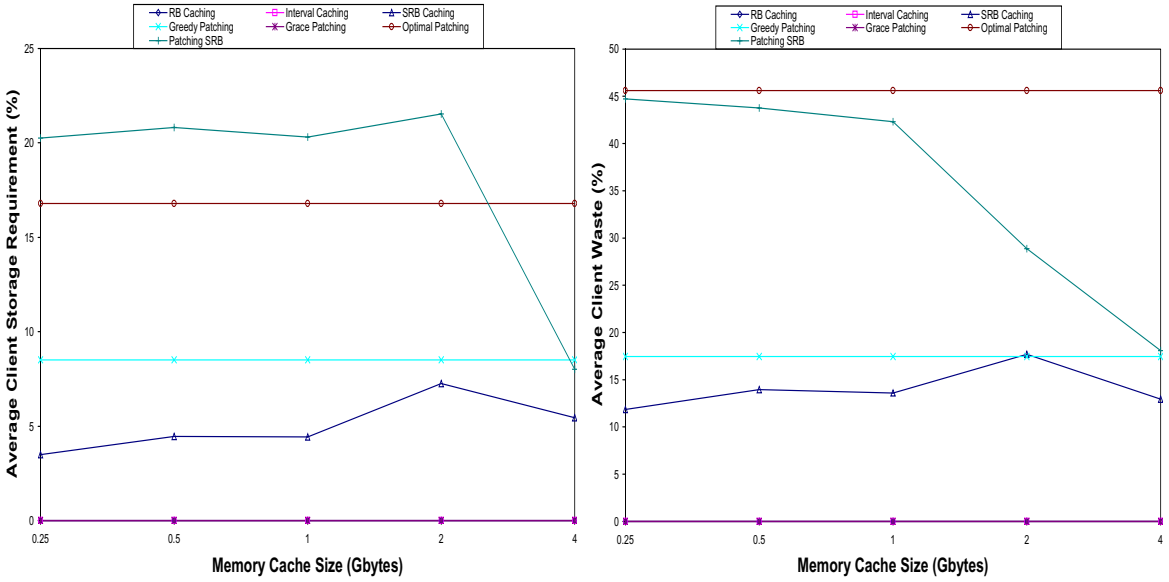

Figure 19: PARTIAL workload (left to right): (a) Average Client Storage Requirement(%) and (b) Client Waste(%) with the Scale of 1/4

5.3 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.

Comparing Figure 20(a) with Figure 12(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

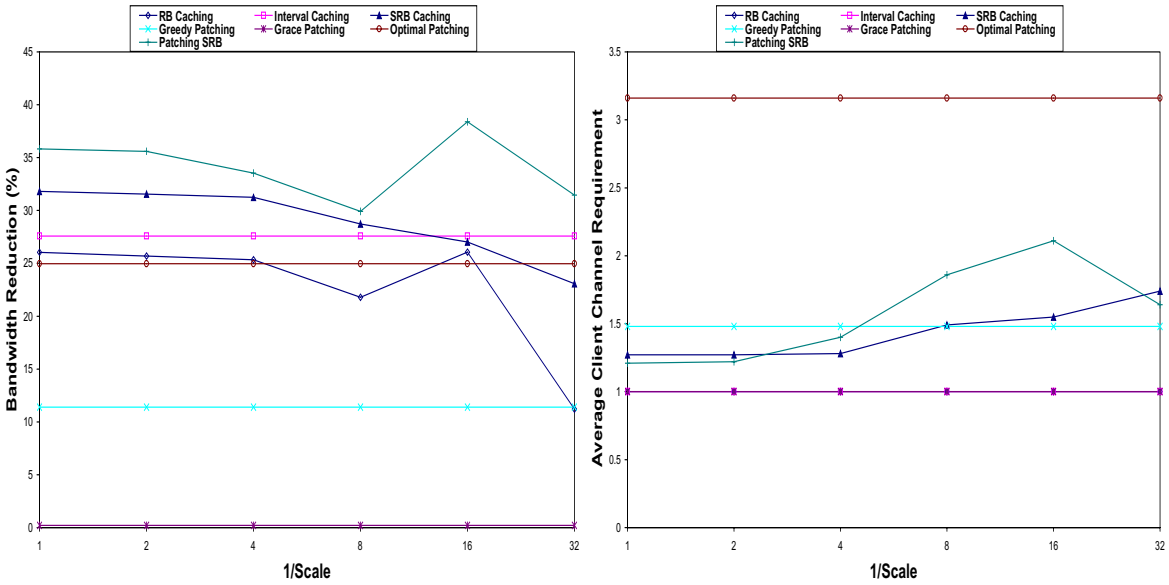

Figure 20: REAL workload (left to right): (a) Bandwidth Reduction and (b) Average Client Channel Requirement with 1GB Proxy Memory

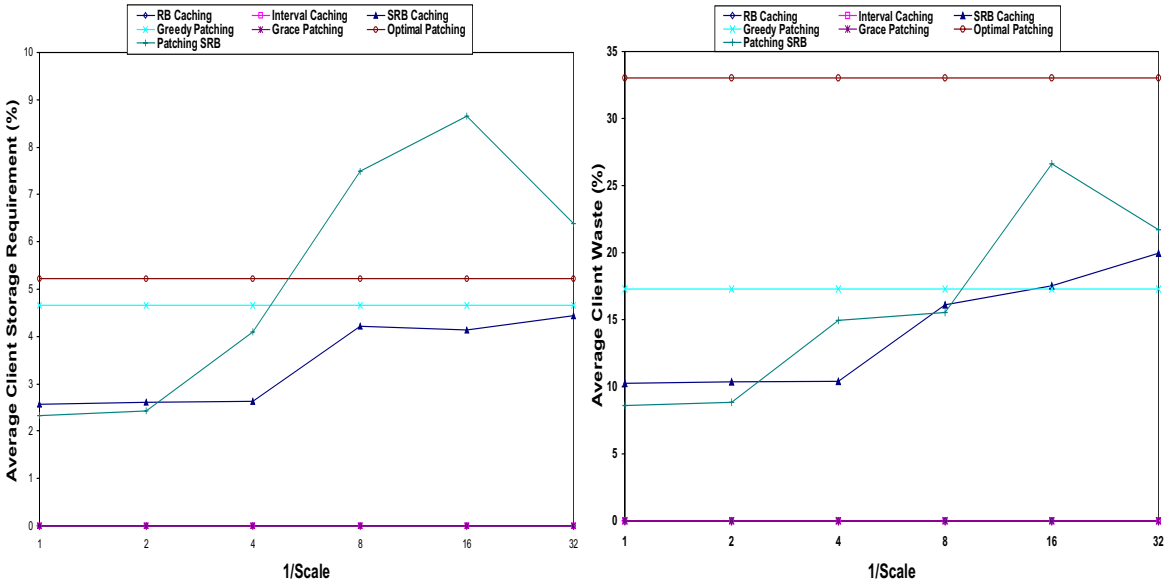

Figure 21: REAL workload (left to right): (a) Average Client Storage Requirement(%) and (b) Client Waste(%) with 1GB Proxy Memory

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. Figure 22 and 23 show the server traffic reduction and the client side statistics. Compared with the simulation results obtained with synthetic workloads, Figure 22(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.

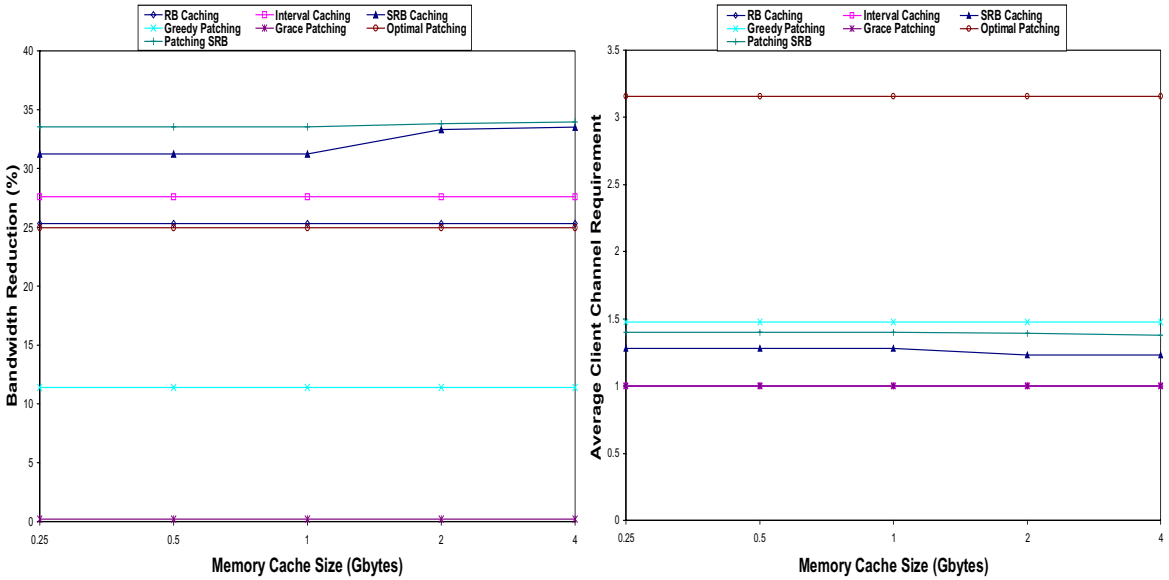

Figure 22: REAL workload (left to right): (a) Bandwidth Reduction and (b) Average Client Channel Requirement with the Scale of 1/4

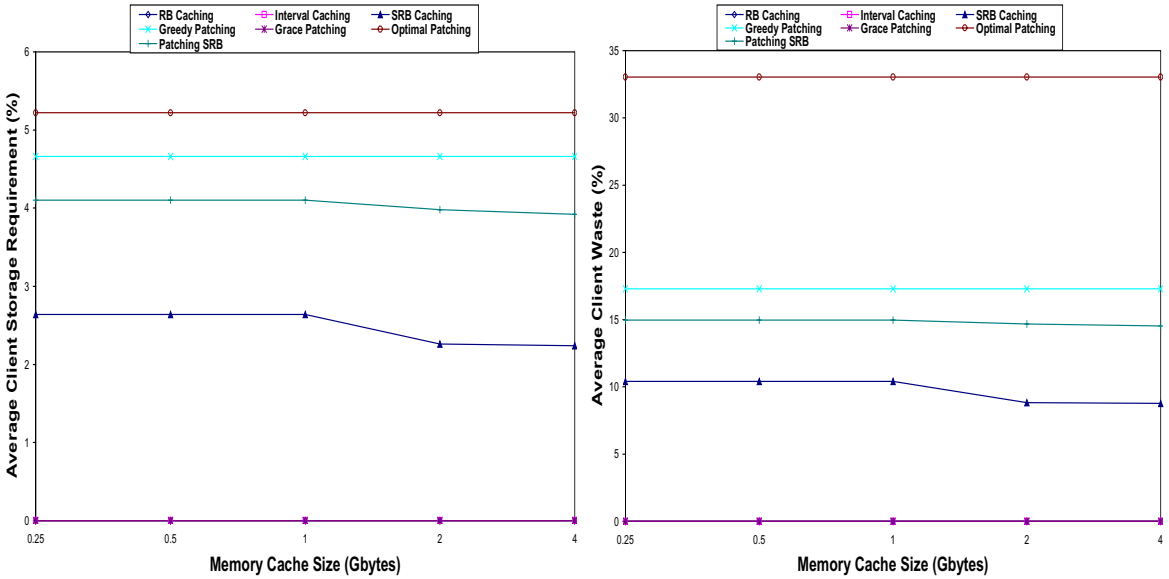

Figure 23: REAL workload (left to right): (a) Average Client Storage Requirement(%) and (b) Client Waste(%) with the Scale of 1/4

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 good caching performance with a limited memory resource.** This is also the

motivation of the proposed SRB and PSRB algorithms.

5.4 Further Analysis

5.4.1 Replacement Policies

The caching system faces two choices when the proxy memory is exhausted. First, the proxy executes no caching until memory space becomes available again either due to session termination or buffer reclamation. Second, the proxy reclaims buffers that serve on-going sessions. The detail description of the replacement can be found in Section 3.2.3 Here, we compare the performance of this two approaches. Figure 24 shows for various workloads, the server traffic reduction rate achieved by the SRB algorithm using the no-replacement and two flavors of replacement approaches.

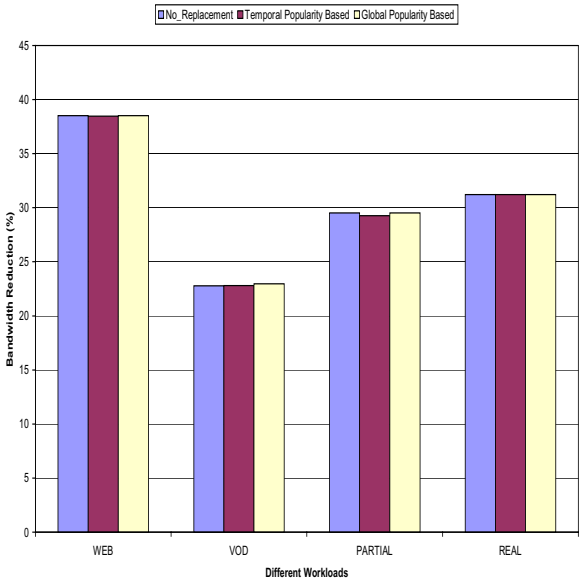

Figure 24: Different Replacement Policies for SRB Caching Algorithm

Evident from Figure 24, the three approaches achieve similar caching performance. This indicates that the proposed algorithm achieves similar caching performance without considering the replacement. This is because the proposed algorithms themselves adaptively allocate the buffer and always favor the popular and frequently requested objects. In other words, the sessions that delivers popular objects are more capable of obtaining and retaining the running buffers. In addition, as shown in Table 1, the time interval covered by a running buffer is often longer than the request inter-arrival time. The temporal-popularity-based replacement policy achieves similar performance as the global-popularity-based replacement policy since there is no variation in the popularity distribution when the synthetic workloads are created. In reality, if the popularity parameter varies with time, there may be a difference in performance.

5.4.2 Client Channel Requirement

The performance analysis in the previous section indicates that SRB and PSRB algorithm 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. Figure 25 and 26 show the CDF of the client channel requirement for simulations on four workloads. In these simulations, the proxy has 1GB memory capacity and the *scale* factor for the initial buffer size is fixed at 1/4.

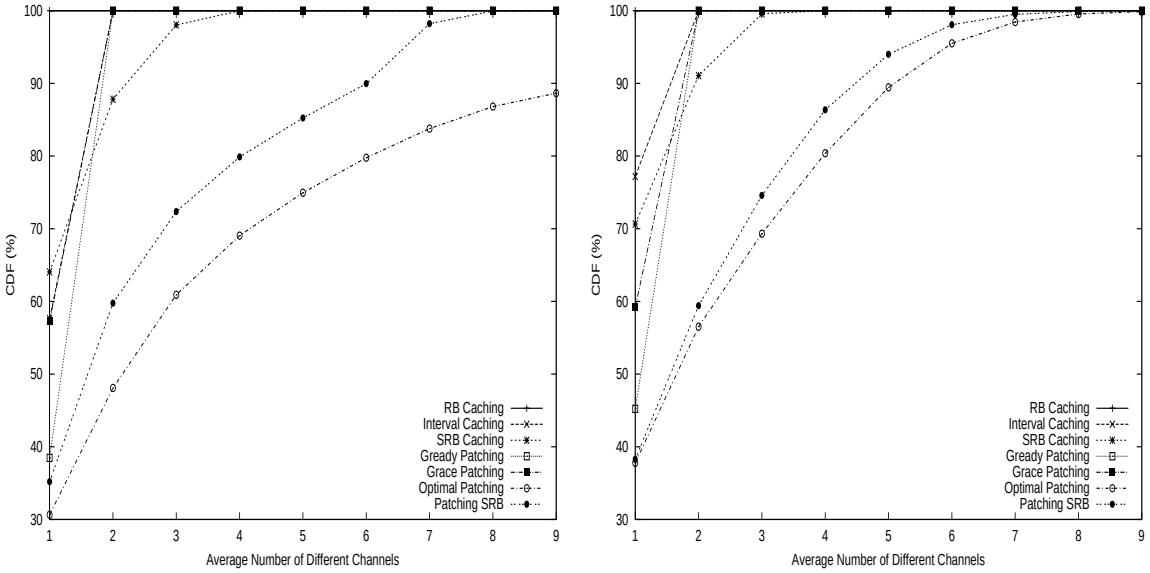

Figure 25: Client Channel Requirement CDF (left to right): (a) WEB and (b) VOD Workloads

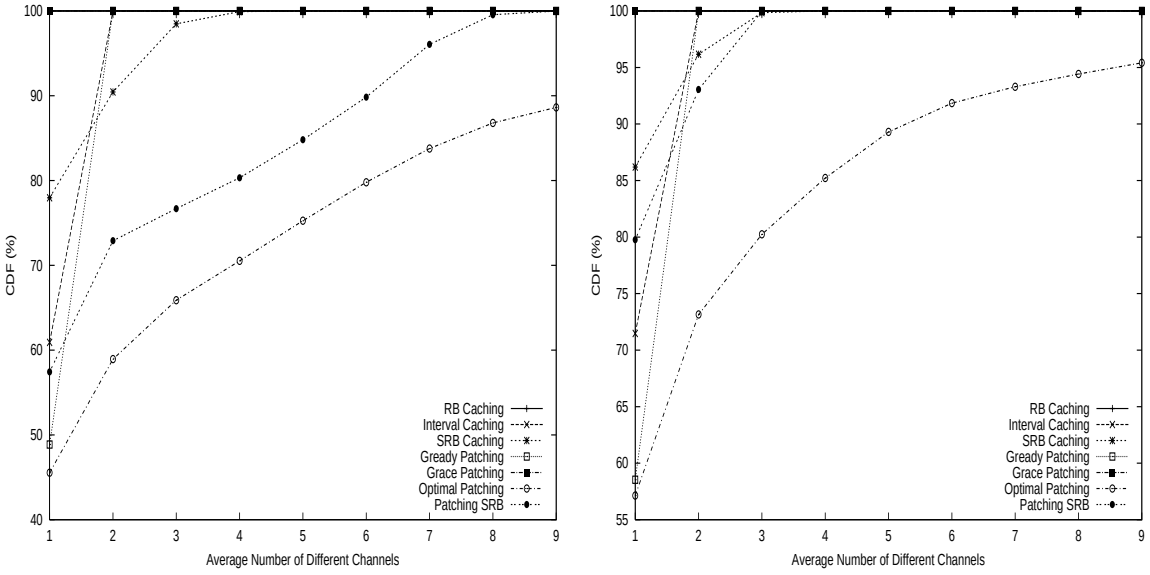

Figure 26: Client Channel Requirement CDF (left to right): (a) PARTIAL and (b) REAL Workloads

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.

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 Figure 26(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 Figure 22(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.

6 Conclusion

In this paper, we propose two new algorithms for caching of streaming media objects. Shared Running Buffers (SRB) caching algorithm is proposed to dynamically cache media objects in the proxy memory during delivery. Patching SRB(PSRB) algorithm is proposed to further enhance the memory utilization on the proxy. The adaptiveness of the two algorithms are analyzed and exploited. Extensive simulations using both synthetic and real workloads are conducted. The simulation results demonstrate the efficiency achieved by the proposed algorithms. Both algorithms require the client capable of listening to multiple channels at the same time. Compared with previous solutions which also require multiple client channels, the proposed algorithms achieve higher server traffic reduction rate with less or similar load on the link between the proxy and the client.

As observed from the simulation results based on the real workload, the proposed algorithms perform better when the access arrivals are bursty. We plan to improve the adaptiveness of the proposed algorithms taking into consideration of the bursty nature. Future work also includes the the investigation on the performance of the algorithms when the client side storage is limited and the streaming rate is not a constant.

Acknowledgment: We would like to thank Dr. Mitchell Trott for providing detailed corrections and valuable suggestions for future work. Thanks to Dr. Susie Wee for helpful discussions on the algorithm representations. Thanks to all other members in the streaming media group for helping in various parts of this project.

7 References

[1] C.C. Aggrawal, J.L. Wolf and P.S. Yu, "On Optimal Batching Policies for Video-on-Demand Storage Server", *Proc. of the IEEE Int'l Conf. on Multimedia Systems*, June 1996.

[2] David P. Anderson, Yoshitomo Osawa and Ramesh Govindan, "A File System for Continuous Media", *ACM Transactions on Computer Systems*, Vol. 10, No. 4, Nov. 1992, pp. 331-337.

[3] Ethendranath Bommaiah, Katherine Guo, Markus Hofmann and Sanjoy Paul, " Design and Implementation of a Caching System for Streaming Media over the Internet", *IEEE Real Time Technology and Applications Symposium* , May 200.

[4] C.M. Bowman, P.B Danzig, d.R. Hardy, U. Manber, M.F. Schwartz, and D.P. Wessels, "Harvest: A scalable, customizable discovery and access system", *Tech. Re. CU-CS-732-94*, University of Colorado, Boulder, USA, 1994.

[5] 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*, 2002.

[6] Ludmila Cherkasova and Minaxi Gupta, "Characterizing Locality, Evolution, and Life Span of Accesses in Enterprise Media Server Workloads", *Proceedings of NOSSDAV 2002*, Miami, FL, May 2002.

[7] M. Chesire, A. Wolman, G.M. Voelker and H.M. Levy, "Measurement and Analysis of a Streaming-Media Workload", *Proc. of 3rd USENIX Symposium on Internet Technologies and Systems*, Mar. 2000.

[8] Asit Dan, Dinkar Sitaram, "Buffer Management Policy for an On-Demand Video Server", *IBM Research Report 19347*, Yorktown Heights, NY 1993.

[9] Asit Dan, Dinkar Sitaram. "A Generalized Interval Caching Policy for Mixed Interactive and Long Video Workloads",*Proceedings of IS&T/SPIE Multimedia Computing and Networking 1996*, San Jose, California, January 29-31, 1996.

[10] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching", *Proc. of ACM Multimedia*, pp. 15-23, San Francisco, CA, Oct. 1994.

[11] N.L.F Fonseca and R.A eFaCanha, "The Look-Ahead-Maximize-Batch Batching Policy", *IEEE Transactions on MULTIMEDIA*, Vol 4, No. 1, March 2002.

[12] Lixin Gao and Don Towsley. "Supplying instantaneous Video-On-Demand services using controlled multicast", *ICMCS'99.*

[13] Kien A. Hua, Ying Cai and Simon Sheu, "Patching : A Multicast Technique for True Video-on-Demand Services",*ACM Multimedia 98*, Bristol, England, 1998.

[14] http://www.squid-cache.org/

[15] Sung-Ju Lee, Wei-Ying Ma, and Bo Shen, "An Interactive Video Delivery and Caching System Using Video Summarization", *Computer Communications*, vol. 25, no. 4, pp. 424-435, Mar. 2002.

[16] A. Lutonen, H.F. Nielsen and T. Berners-Lee, "Cern httpd", http://www.w3.org/Daemon/Status.html

[17] Subhabrata Sen, Lixin Gao, Jennifer Rexford and Don Towsley, "Optimal Patching Schemes for Efficient Multimedia Streaming", *Proc. NOSSDAV '99*, Basking Ridge, NJ, June 1999.

[18] Subhabrata Sen, Jennifer Rexford, and Don Towsley, "Proxy Prefix Caching for Multimedia Streams". *INFOCOM'99*.

[19] F. Tobagi, J. Pang, R. Baird, and M. Gang, "Streaming RAID A disk array management system for video files", *ACM Multimedia*, pp. 393-400, 1993.

[20] Kun-Lung Wu, Philip S. Yu and Joel L. Wolf, "Segment-based proxy caching of multimedia streams", *WWW 2001*, pp. 36-44.