DISC: Dynamic Interleaved Segment Caching for Interactive Streaming

Lei Guo[1], Songqing Chen[2], Zhen Xiao[3], and Xiaodong Zhang[1]

[1]Department of Computer Science
College of William and Mary
Williamsburg, VA 23187, USA
{lguo, zhang}@cs.wm.edu

[2]Department of Computer Science
George Mason University
Fairfax, VA 22030, USA
sqchen@cs.gmu.edu

[3]AT&T Labs-Research
180 Park Ave.
Florham Park, NJ 07932, USA
xiao@research.att.com

Abstract

Streaming media objects have become widely used on the Internet, and the demand of interactive requests to these objects has increased dramatically. Typical interactive requests include fast forward and direct jumps. Unfortunately, most of existing streaming proxies are designed for sequential accesses, and only a few solutions have been proposed to maintain additional data structures in the proxy to support some interactive operations (such as fast forward) other than jumps, which are among the most common interactive requests from the clients.

Focusing on interactive accesses, in this paper we present an analysis of streaming media workload collected from thousands of broadband users hosted by a major ISP. Our analysis shows that jump accesses (48%) and pauses (51%) are the dominant client interactive requests and that jump accesses often suffer serious delays due to slow buffering through the network. To support jump accesses effectively, we further propose a novel caching algorithm – DISC (Dynamic Interleaved Segment Caching), which trades cache performance for response time to client interactive requests. In this algorithm, segments of a media object are cached dynamically according to client access patterns. DISC can support direct jumps efficiently while ensuring timely prefetching of uncached segments for sequential accesses. Trace-driven simulations demonstrate that DISC outperforms other caching schemes significantly for interactive requests with only a small degradation in cache performance.

1 Introduction

Streaming media content delivery has become increasingly important with the proliferation of multimedia content in a broad spectrum of application areas, such as remote education, digital libraries, military communications, and entertainment. In addition, media files on the Internet today are much larger and their playback durations are much longer than they were just a few years ago [5, 11]. Consequently, VCR-like interactive operations (e.g. jump, pause, fast forward) are more common nowadays during the playback as clients search for the parts they are interested in. This trend has been observed previously in both the educational environment and in the entertainment environment (see e.g. [1, 6, 13]).

Efficient support for interactive operations is a challenging problem in Internet streaming. Although a number of proxy caching solutions have been proposed for efficient delivery of streaming media (see e.g. [4, 17], most of them are designed to optimize client perceived quality during sequential accesses. They typically divide a media object into segments and cache a few beginning segments only, expecting uncached segments to be fetched later. While this design may work well when a client watches the media continuously, it provides inadequate support for interactive requests from the client. For example, when the client jumps from the current position to a later position in the media, the proxy may not have the corresponding segment in its cache. If it has to request the segment from the origin server, the client is likely to suffer long delays or jitters during playback.

So far, only a small number of solutions have been proposed to support client interactive requests in streaming. They usually require the creation of additional data files in the proxy or in the media server and can support only a limited set of interactive operations such as fast forward, rewind, and preview [9, 16]. None of them provides effective support for jumps, which according to a recent study are among the dominant interactive operations from clients [6].

In this paper, we first present an analysis of a streaming media workload from thousands of broadband home users collected at a major ISP. Our major findings include:

1. The locality of references for multimedia objects has

improved significantly during the past three years, indicating that caching streaming media objects might be more effective than before.

2. The majority of the client interactive requests are jumps (48%) and pauses (51%) while most of jumps are on popular objects.

3. Client jump operations often suffer long delays due to slow buffering rate.

Our study shows that caching segments in sequential order does not benefit interactive requests, such as jump accesses. Thus, a dynamic caching scheme is desirable, which not only helps the interactive requests, but should also retain the advantage of sequential caching. We propose **D**ynamic **I**nterleaved **S**egment **C**aching (DISC) for this purpose. In this algorithm, segments of a media object are cached dynamically according to the access patterns of clients. It supports jump accesses by caching appropriate segments discretely according to the clients' jump patterns. In the meantime, it ensures continuous streaming delivery for sequential accesses with prefetching support whenever necessary. By caching objects in interleaved segments, DISC sacrifices proxy cache performance in order to reduce the response time to client jump requests. Simulation based experiments demonstrate that our algorithm outperforms the continuous segment caching strategy for interactive requests with only a small degradation in cache performance.

The reminder of this paper is organized as follows. We overview RTSP, the standard Internet streaming protocol, in Section 2. We present our trace study in Section 3. Section 4 describes the details of our interleaved segment caching algorithms. Simulation results are presented in Section 5. Section 6 describes related work and Section 7 contains our concluding remarks.

2 RTSP Operation Overview

RTSP (Real Time Streaming Protocol) is the standard Internet streaming protocol for the communication between a media player and a media server. Figure 1 illustrates client and server interactions for client play, pause and jump access in RTSP. A media player uses the `SETUP` method to establish a connection for a video/audio stream in a multimedia session. The `PLAY` method is used to start the transfer of the stream. The player sends the `PAUSE` method to the media server when it needs to halt the transmission temporarily while keeping the session state at the server. When the client initializes a jump access, such as clicking or dragging the playback cursor in the media player, a set of messages exchange between the client and the server: the player first sends a `PAUSE`, waiting for the server to respond. After getting the server's response, the player

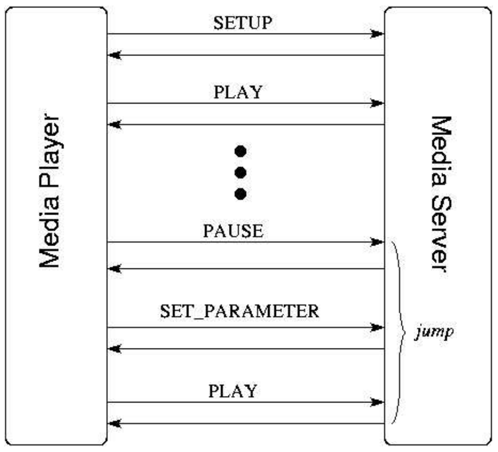

Figure 1. Overview of RTSP Operations

may exchange some parameters with the media server using the `SET_PARAMETER` method. Finally, the player sends a `PLAY` command to the media server, specifying the position it wants to jump to. In contrast, for a normal startup play request, the media player simply sends a `PLAY` command to the server.

3 Analysis of Client Interactivity

We collected streaming media workload from a large group of broadband users connected to the Internet via a major ISP. The workload covers a three-day period from 2004-09-21 17:14:20 to 2004-09-24 17:21:30. We captured the first IP packets of all RTSP messages and the TCP/UDP packet headers of the streaming traffic with the packet arrival time using the Gigascope appliance [7]. From the TCP/UDP headers we can derive the real transferred traffic. Then we grouped the RTSP messages by TCP connections, with each connection representing a RTSP streaming session. We parsed the RTSP commands in the RTSP packets, and extracted media URLs, playback lengths, and encoding rates. Then we matched the streaming traffic with the identified RTSP sessions based on their IP addresses, port numbers, and data capturing time. Finally we measured the interactive response time and data transmission rate.

There are 2,748 distinct clients in our workload. They accessed a total of 10,266 distinct media objects from 1,110 streaming servers during 23,001 streaming sessions, which together transferred 80 GB streaming media traffic. The media encoding rate in this workload ranges from 8 Kbps to 1.769 Mbps, and the user playback duration ranges from several seconds to five hours. Compared with the workload study from the University of Washington three years ago [5], both the quality and the playback time of streaming media have improved significantly.

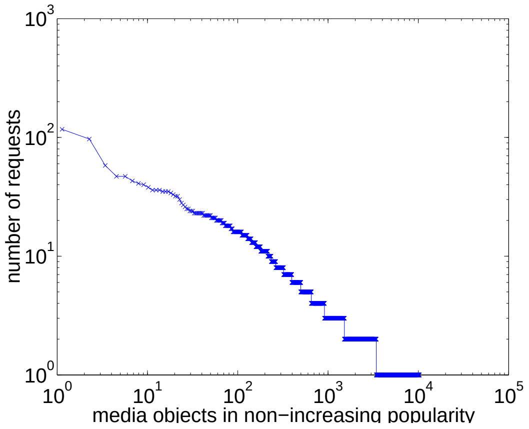

Figure 2. The locality of media object references. Both axes are in log scale.

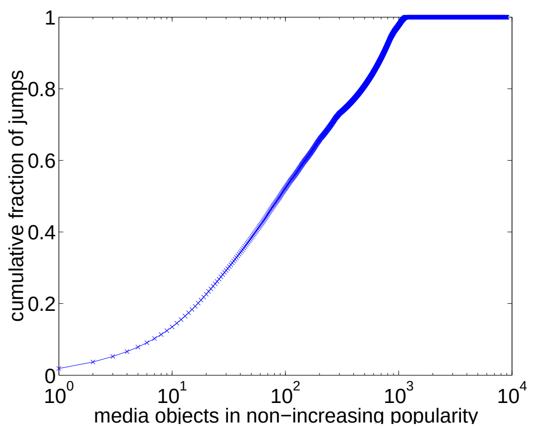

Figure 3. The locality of jump requests

3.1 Locality of References

We first analyze the distribution of media references in our workload, as shown in Figure 2. The x-axis is the sequence of media objects sorted by their popularities in non-increasing order, and the y-axis is the numbers of their corresponding references in the workload. Note that both axes are in log scale.

The figure indicates that the reference locality of streaming media objects follows a Zipf-like distribution $f_i \propto \frac{1}{i^\alpha}$ roughly, where i is the rank of a media object, f_i is its reference number, and $\alpha \approx 0.61$ is the skewness parameter. A measurement study [5] performed three years ago shows a similar distribution shape of the reference locality with $\alpha \approx 0.47$. Comparing these two studies, we find that the streaming media accesses have become more concentrated on popular objects, and that proxy caching can become more effective.

3.2 Locality of Interactive Requests

We extracted user interaction commands such as `PLAY`, `PAUSE`, and `TEARDOWN` from the RTSP packets, and then identified the jump operations in each streaming session based on the context of client accesses.

As reported in [6], where the fast forward and rewind occupy less than 1% in their workloads, similar results have been observed in our workload. Among all interactive requests, about half (48%) of them are jump requests, and the rest (51%) are pause requests.

Figure 3 shows the cumulative fraction of jump requests on media objects ordered in non-increasing popularities. As indicated in the figure, jump requests are highly concentrated on a small number of popular objects. For example, the top 100 popular objects account for about half of all jump accesses. The skewness of jump request distribution enables the feasibility of supporting jump operations through proxy caching.

3.3 Analysis of Jump Access Performance

A media player maintains a running buffer, which is preloaded with media data before playback starts, to smooth possible playback jitter due to the fluctuation of end-to-end bandwidth between the server and the client. The buffer size ranges from 5 seconds to 30 seconds of requested media data for most of commercial media players. Old media data is drained out and new media data is filled into the buffer continuously during streaming. When a user clicks the play button or drags the playback progress cursor on the media player to start a streaming session or to specify a seeking position, the user usually has to wait for a period of time before playback begins. This *user waiting time* mainly consists of three parts: the *round trip time* for communication between the client and the server, the *server response time* from the time when the server receives the command to the time when it sends the first demanded data packet, and the *buffering time* to fill the media player buffer.

Figure 4 shows the server response time to the media startup requests and the server response time to the jump requests, for streaming sessions with interactive operations.

As shown in the figure, the server response time of a jump request is much larger (about ten times longer) than that of a startup request. However, most of the server response times are less than 1 second.

The media player cannot start the playback until the buffer is full. When a media player sends a startup or jump request to the server, it specifies its desired buffering speed in the request message. Dissecting the messages exchanged between media players and media servers, we find a client

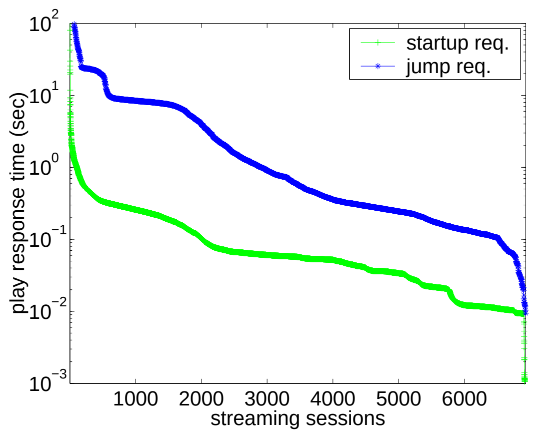

Figure 4. The server response time of jump requests and startup requests

player always requests a server to fill the buffer as fast as possible to minimize user waiting time. However, the server responds with different speeds to the jump requests and to the startup requests, with a maximum of about 5 times of normal playback speed.

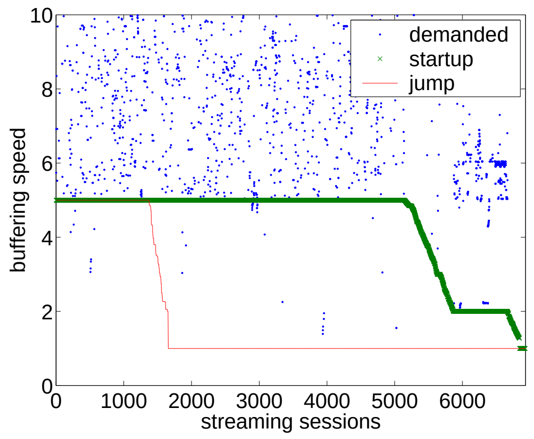

Figure 5. The buffering speeds

Figure 5 shows the demanded buffering speed and the actual buffering speed for jump requests, with a comparison to the buffering speed for startup requests. In this figure, the x-axis represents the sessions where jump accesses happened. The demanded buffering speed is always the speed at the maximal available bandwidth between the client and the server. The average buffering speed for startup play requests is about 4.4 times of normal playback speed, while the average buffering speed for jump requests is only about 1.8 times of normal playback speed. Compared with the startup delay where the buffering happens for the first time, the longer user waiting time for a jump request is mainly caused by slow buffering speed. A possible reason may be the handling priority for different requests on the streaming server: it is easy to understand that the streaming server should satisfy continuous streams first, and put interactive requests in second priority.

Our above findings through the measurement study motivate us to use the proxy caching approach to reduce user waiting time for client jump accesses, which has not been addressed in the proxy or server as far as we know.

4 Dynamic Interleaved Segment Caching Strategy

To support client jump accesses, we propose the interleaved segment caching strategies. In this section, we first present the basic interleaved segment caching (BISC) strategy to illustrate the principle of interleaved segment caching, followed by the proposition of our dynamic interleaved segment caching (DISC) algorithm. In these strategies, each object consists of a number of segments, each of which is composed of small blocks, where each block represents a sequence of actions in time or space.

4.1 Basic Interleaved Segment Caching

Our previous analysis shows that jump requests often suffer long buffering delays. One way to reduce such delays is through proxy caching. However, existing segment-based streaming proxies usually cache only the beginning portion of a media object [3]. We call them Continuous Segment Caching (CSC) strategies. This is illustrated in Figure 6 (a) where the first four segments of an object are cached sequentially. Such a design can reduce startup latency, provide free-of-jitter delivery, and improve cache performance based on object popularity. However, they lack the support to jump requests: once a client jumps to an uncached segment, the client will experience a long delay due to the buffering time.

Given only partial segment caching is allowed, the rationale of BISC is to disperse originally continuously cached segments of an object so that the chance of a cache hit during a jump access is higher. This is exemplified by Figure 6 (b). As long as the client jumps to a cached segment, the buffering time is reduced. Furthermore, when the client jumps to an uncached segment, it can be directed to the closest cached segment with some approximation. On the downside, BISC will cache fewer beginning segments of an object than CSC. Since the beginning part of an object is normally more popular than the later part, BISC trades byte hit ratio (the proxy performance) for response time (the client performance) to jump requests.

Despite of the reduced cache performance, BISC can provide the same continuous delivery guarantee as CSC for

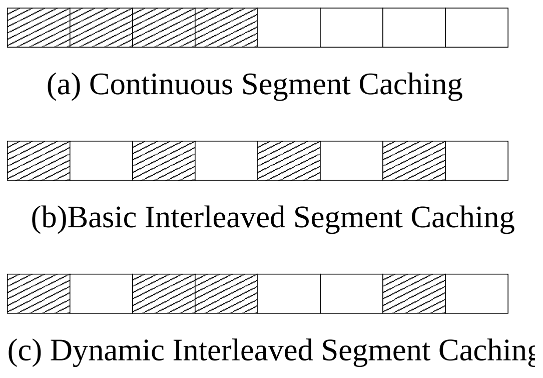
(a) Continuous Segment Caching
(b)Basic Interleaved Segment Caching
(c) Dynamic Interleaved Segment Caching

Figure 6. Rationale of Basic Interleave Segment Caching

the same amount of cache space. In summary,

1. To client sequential access, BISC can provide the same continuous streaming delivery as CSC does, with prefetching support whenever necessary;

2. Additionally, when prefetching is needed, the potential wastage of prefetched data caused by client early terminations in BISC is less than that in CSC.

Prefetching is necessary when the end-to-end bandwidth between the media server and the client is less than the media object playback rate. By assuming that

- the object length is L;
- the object encoding rate (also the client playback rate) is E on average;
- the end-to-end bandwidth between the proxy and the server is B on average;

we will show how to design BISC and show its aforementioned features.

With the support of partial data caching for media objects, a certain fraction of an object can be cached in the proxy before client access to guarantee a timely prefetching [4]. Statically, it is easy to find that in the worst scenario when the client is going to access the entire object, the minimum length X that needs to be cached must satisfy the following:

$$\frac{L}{E} > \frac{L-X}{B}, \tag{1}$$

which leads to

$$X_{min} = L \times (1 - \frac{B}{E}). \tag{2}$$

We assume X_{min} consists of S segments, which are cached continuously according to the CSC strategy. Once a client requests the object, the proxy starts to prefetch the remainder segments $S+1, S+2, ...$, until the termination of the media object or the end of client access. The caching of the initial S segments guarantees a timely prefetching of $\frac{B}{E}L$ data.

In BISC, we disperse these S segments discontinuously with the segment length $L_s = \frac{L(1-\frac{B}{E})}{S}$ and the distance to the next cached segment being $\frac{\frac{B}{E}L}{S}$. Thus, caching of one segment needs to guarantee the timely prefetching of data of length $\frac{\frac{B}{E}L}{S}$. Based on Equation 1, this is guaranteed.

To show the second feature, suppose at any point, the possibility of client termination is P_x. In the CSC strategy, the average prefetched data wastage on average is thus

$$\int_0^{L(1-\frac{B}{E})} P_x \times \frac{x}{E} \times B dx + \int_{L(1-\frac{B}{E})}^{L} P_x \times (\frac{x}{E} \times B - (x - L(1-\frac{B}{E})))dx. \tag{3}$$

While in BISC, the prefetched data wastage on average is the sum of from each $\frac{L}{S}$ unit. So we get

$$\sum_{i=0}^{i=S-1} \int_{i\frac{L}{S}}^{i\frac{L}{S}+\frac{L(1-\frac{B}{E})}{S}} P_x \times \frac{x}{E} \times B dx +$$
$$\int_{i\frac{L}{S}+\frac{L(1-\frac{B}{E})}{S}}^{(i+1)\frac{L}{S}} P_x \times (\frac{x}{E} \times B - (x - \frac{L(1-\frac{B}{E})}{S}))dx. \tag{4}$$

If P_x is independent of the position in the object, then for BISC, the average wasted data can be represented as

$$S \times P_x \times (\int_0^{\frac{L(1-\frac{B}{E})}{S}} \frac{x}{E} \times B dx +$$
$$\int_{\frac{L(1-\frac{B}{E})}{S}}^{\frac{L}{S}} (\frac{x}{E} \times B - (x - \frac{L(1-\frac{B}{E})}{S}))dx). \tag{5}$$

If we assume that the client terminates her request with a 50% probability at any access point, for the CSC strategy, the average prefetched data wastage is

$$\frac{1}{4}L^2\frac{B}{E} \times (1 - \frac{B}{E}) \tag{6}$$

While for BISC, the average prefetched data wastage is

$$\frac{\frac{1}{4}L^2\frac{B}{E} \times (1 - \frac{B}{E})}{S} \tag{7}$$

The average wastage in BISC is only $\frac{1}{S}$ of that in CSC! This also indicates that with a larger number of segments (and the smaller size of the segments), the average prefetched data wastage will be less. However, in practice, the length of a segment cannot be too small. The change of the size of the segment affects the number of the segments, which in turn affects the distance between segments. For effective support of jump accesses, capturing the jump distance is important. If S segments are allowed to be cached, the ideal jump distance J is $\frac{L}{S} - \frac{L(1-\frac{B}{E})}{S} = \frac{\frac{B}{E}L}{S}$. Similarly, if we find J (by characterizing the client jump access pattern), we can determine how many segments for this object as $S = \frac{\frac{B}{E}L}{J}$, and thus the length of each cached segment should be $\frac{J}{B}(E-B)$. This is the principle we use in our algorithm designs.

Additional changes can be made on the BISC strategy for different purposes. As shown in Figure 6 (c), in a local range, segment 3,4 can be cached successively instead of caching segment 3, 5 if necessary. In a larger range, given a jump distance, there are different segment chains that can be cached within BISC, such as caching segment 2, 4, 6, 8 instead of 1, 3, 5, 7 in Figure 6 (b). These adjustments can be utilized to improve cache performance, such as when the segment popularity changes. In the following subsection, we present our dynamic interleaved segment caching algorithm, in which the adjustment described above can apply. Particularly, dynamic interleaved segment caching considers the client sequential access pattern and the jump access pattern dynamically.

4.2 Dynamic Interleaved Segment Caching

BISC always caches objects in interleaved segments. However, in practice, some objects are accessed sequentially most of the time and will benefit more from continuous caching. Thus, we propose our heuristic algorithm, DISC, which can provide sufficient support to jump accesses while improving the cache performance over BISC. Except for this point, BISC shares the same components with DISC as we presented below.

In DISC, each object gets cached fully when it is accessed for the first time to set up an observation period for the client access pattern. Later, when the fully cached object is selected as a victim by the replacement policy, a quota is calculated to determine the number of its segments that can remain in the cache. (The quota computation will be explained later in the section.) The algorithm needs to decide whether it should try to favor future jump accesses or future sequential accesses. If the frequency of the jump accesses is significant, it may be worthwhile to favor future jump accesses by caching dispersed segments of the media object. Otherwise, it may be worthwhile to favor future sequential accesses by caching sequential segments of the object. The caching strategy (BISC or CSC) of the object is adjusted periodically to accommodate the present client access pattern. Two object replacement policies are used to reclaim the cache space when needed from fully cached and partially cached objects, respectively. To improve cache performance, self replacement is active for interleaved cached segments based on segment popularity.

To facilitate the DISC algorithm, the following items in a data structure are maintained and updated in the proxy along the client accesses for each object.

- T: the pre-determined period to adjust the quota of each object;
- E: the average of the extracted object encoding rate samples in T;
- B: the average of the bandwidth sampled from the proxy to the server for the prefetching of this object in T;
- W: the entirely accessed object length from clients;
- Q: the quota of the object;
- P: the popularity of the object (the number of accesses excluding jump accesses), and the client average access length is $\frac{W}{P}$;
- N: the total number of the jump accesses to the object;
- D: the total of the jump distances of client accesses, and the average is thus $J = \frac{D}{N}$;
- P_j: the popularity of the j^{th} segment of the object, reset at the beginning of each period;
- T_0: the first time the object is accessed;
- T_{now}: the current time;
- R: the ratio of client jump accesses to object popularities, which is $\frac{N}{P}$;
- R_{th}: the threshold for R;
- F_{th}: the threshold for client unit jump frequency ($\frac{N}{P \times L}$);
- P_{th}: the threshold for popularity ratio changes.

4.2.1 Object Admission and Replacement

Upon a new object access, the object admission policy always tries to store the object fully so that an observation window can be set. In the observation window, the client accesses (P), the sum of the access length (W), the number of jump accesses (N), the total jump distance (D), the average channel bandwidth (B), and streaming bandwidth (E) (which is extracted from the object encoding rate) are updated accordingly.

If there is insufficient cache space, the replacement policy is called. The replacement policy looks for the fully cached object first. Among the fully cached objects, the oldest one, based on $T_{now} - T_0$, is always selected as a victim. Upon the selection, the corresponding quota and caching initialization will be applied (see Section 4.2.2). If no sufficient space is found after all fully cached objects are checked, the popularity based replacement policy applies to partially cached objects. The replacement always replaces the segments of the least popular object, based on P, from its tail until sufficient cache space is found.

4.2.2 Quota and Cache Initialization

Once a fully cached object is selected as a victim by the replacement policy, the following steps are involved in the DISC algorithm initialization.

- Allocating Quota: when the average bandwidth from the proxy to the server is less than the playback rate, the quota Q is calculated to allocate sufficient cache space for this object to ensure in-time prefetching. It is the larger of the average access length of this object and the calculated value of X_{min} in equation 2. Mathematically, $Q = max(L(1 - \frac{B}{E}), \frac{W}{P})$. If the playback rate is less than the bandwidth, then no prefetching is required and $Q = 0$.

- Segmenting Object: Having the quota Q, and calculating the average jump distance as $J = \frac{D}{N}$, the number of segments of this object S and the segment length are calculated as $\frac{B \times L}{E \times J}$ and $\frac{J}{B}(E - B)$. If N is zero, the object has one segment cached.

- Initializing Continuous Segment Caching: If the client unit jump frequency, calculated as $\frac{N}{P \times L}$, is less than the threshold F_{th}, the quota is filled with the beginning portions of the object. The rest segments are evicted from the proxy. Note that large objects tend to have more jump accesses. Hence, we include the object length L in the calculation of F_{th}.

- Initializing Basic Interleaved Segment Caching: If the client unit jump frequency is larger than or equal to the threshold F_{th}, segments starting from the first one are cached in interleaved fashion with the jump distance of J. The rest segments that are not on this interleaved segment chain are evicted.

4.2.3 Self Replacement

The self replacement policy is active in each period for objects with interleaved cached segments. It compares the popularity of cached segments to uncached segments in the same chain to determine if the cached segments are sufficiently popular. If not, the cached segments are replaced with more popular uncached segments if the popularity of the cached chain is less than any of the uncached one by P_{th}. The popularity measure can be either the number of jump accesses or the number of total accesses. Any replacement is performed on a one-to-one basis, and therefore the self replacement neither reduces nor increases the amount of cache space occupied by the media object.

For example, if a media object has k segments in the cache. Then the algorithm first determines which k segments of the object are most popular right now. If not all of them are in the cache, then the algorithm can swap them in provided that the difference in popularity is greater than P_{th}. The actual replacement is performed in an on-demand manner, i.e., only when some client accesses them.

4.2.4 Implicit Periodical Update

Since network conditions and client accesses change over time, the quota of each object needs to be updated periodically to reflect the current situation. When a period finishes, the current conditions and the new average access length are taken into consideration to get a new quota for this object. Also, the average jump distance J may have deviated from its original value with time. Lastly, a continuous cached object may have more client jump accesses and thus its R may become larger than the threshold R_{th}, while an interleaved cached object may receive diminishing jump accesses, causing its R to be less than the threshold. The implicit periodical update policy takes care of these changes implicitly at the end of each period with the following steps for partially cached objects:

- Recalculate the quota of the object with the newly updated B, E, W, and P. If the quota gets larger, the object replacement policy is called to reclaim the additional space.

- Upon the success of space reclamation if necessary, the ratio R is recalculated and different steps are taken accordingly as follows.

- If $\frac{N}{P \times L} \geq F_{th}$, the BISC initialization is activated with the current value of J. With the new object division, a new chain of segments is selected and marked to be cached. The cached segments without a mark are evicted. However, if the to-be-cached segments are not in the cache, their caching is delayed until some client accesses them. This saves updating overhead.

- If $\frac{N}{P \times L} < F_{th}$, the CSC strategy is activated. The corresponding segments are marked for caching while the rest are evicted from the cache. Again, if the to-be-cached segments are not in the cache, their caching is delayed until some client accesses them to save updating overhead.

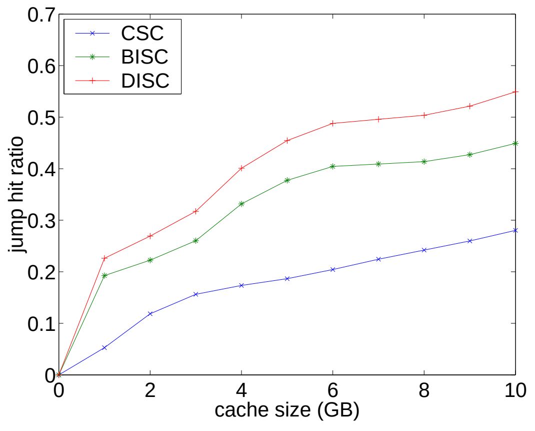

Figure 7. Jump Access Hit Ratio

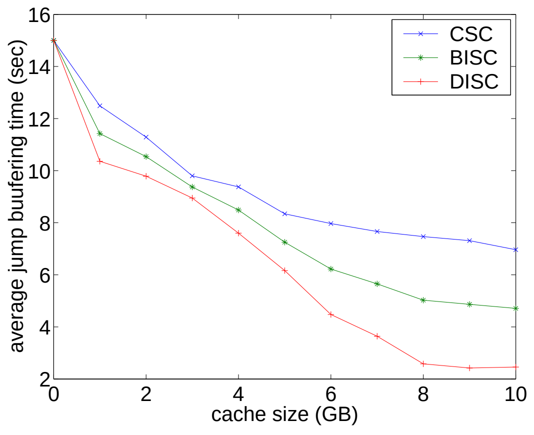

Figure 8. Average Jump Buffering Time

4.2.5 Proactive Prefetching

To improve the client perceived QoS with jump accesses and sequential accesses, the proactive prefetching technique is utilized. For a client access to a continuously cached object, if the access starts from a uncached data segment, it is delayed until the demanded segment is fetched from the server. If the client accesses the cached data segment, the proactive prefetching prefetches the uncached data segments continuously, until the end or the client termination. For a client access to an interleaved cached object, if the access starts from an uncached data segment, it is always re-directed to the closest cached segment. Upon the client accessing the cached segment, proactive prefetching is activated to prefetch the succeeding uncached segments in case the client is going to access them.

5 Simulation Results

To evaluate the performance of our proposed heuristic algorithms, we conduct extensive trace-driven simulations based on our collected workload described in Section 3. In our experiments, we select workload of different time periods (1 hour to 3 days) and set different buffer sizes for media players, ranging from 5-30 seconds of media data to be played. Due to page limits, we only present the results for workload of one day. The buffer size of media players is set to 15 seconds of data for the requested object. F_{th} is 0.025 while P_{th} is 0.5. Experiments with other parameters have similar results. In the following figures, *CSC* represents the continuous segment caching approach. *BISC* represents the basic interleaved segment caching method, sharing all other components with DISC as we presented, except that it enforces interleaved segment caching for each object. *DISC* denotes the dynamic interleaved segment caching algorithm we presented in Section 4.

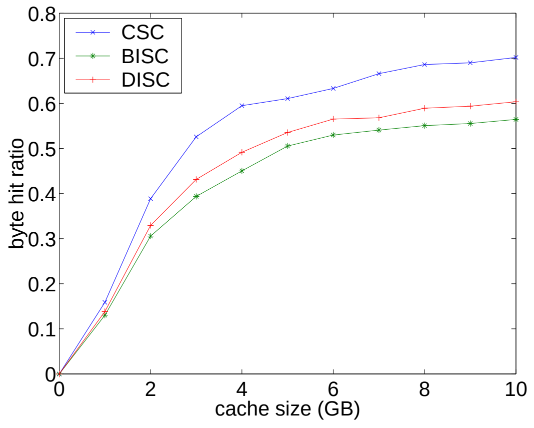

Figure 9. Cache Performance

Figure 7 shows the *jump hit ratio* of the media proxy. Jump hit ratio denotes how many client jumps hit in the proxy, averaged by the total number of jump accesses from clients. It reflects the percentage of client jump accesses that do not depend on the slow buffering rate. As shown on the figure, with the increase of proxy cache size, both BISC and DISC can significantly improve the number of jump access hits over CSC by about 17 and 12 percentage points on average, respectively.

A jump access hit does not mean that buffering can be totally avoided unless all data to be buffered are cached on the proxy. For the three different approaches, Figure 8 shows the *average jump buffering time*, which is the buffering time to jump accesses averaged over the number of jump accesses. As expected, DISC outperforms BISC, BISC outperforms CSC in general. But it is interesting to note that the improvement fluctuates when the cache size increases due to changes of segment sizes.

Since BISC and DISC sacrifice cache performance, in Figure 9 we measure the cache performance in terms of *byte*

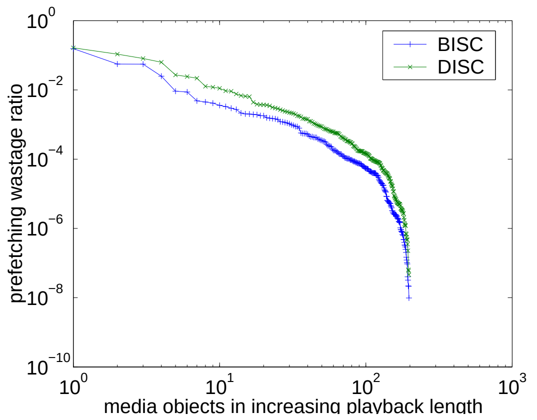

Figure 10. Prefetching Efficiency

hit ratio, which reflects the reduction of the outgoing traffic due to proxy caching. DISC reduces relatively more network traffic with respect to BISC. But both of them degrade the cache performance when compared with CSC by about 8 and 10 percentage points.

As presented in Section 4, BISC and DISC outperform CSC in terms of reducing the prefetched data wastage when the prefetching is needed. Figure 10 shows the *prefetching wastage ratio* of the media proxy, which is the prefetching wastage of DISC and BISC relative to that of CSC. In the figure, the x-axis represents the objects which have larger encoding rate than the network bandwidth from the proxy to the server, ordered in object playback length. The number of such media objects is not large because the workload is collected from the broadband users. Shown in the figure, BISC achieves better results than DISC. However, when the object playback length increases, the number of the segments are getting larger for the object, thus the difference between BISC and DISC becomes smaller, as we analyzed in Section 4.

6 Related Work

In recent years, Internet media content delivery has attracted a lot of research effort from the community [12, 3, 4, 14, 15]. A number of studies are focused on workload characterization of client interactive requests to the rapidly increased Internet streaming media contents. The client interaction patterns are studied for frequencies of different types of client interactions and distributions of session on and off times for MANIC audio content system [19], the educational eTech and BIBS media servers [1], and the educational internal server of a large international corporation [13]. Study [1] also presents the session arrival process and proposes caching strategies for interactive workloads. In [6], four audio and video workloads from educational and entertainmental sites are studied for client interactions, which indicates that 99% of the client interactive requests are jump accesses. The difference and similarities in typical client behavior of three workload classes are also presented. Recently, authors in [18] consider the buffering impact in the Windows Media on the response time to client requests. Different from the previous studies where the workloads are collected from a special entertainment server or educational environment, the workload in our study is collected from a large ISP, servicing a large number of broadband home users, who access all kinds of Web sites. And our analysis finds the impact of the network to the response time to interactive requests, particularly to the jump accesses.

Most of the previous research has studied the interactive request support on the media server. Three typical approaches are proposed for the media server to support fast forward. One is to display frames at a rate n-times faster than normal playback, such as [8], which requires n-times frames to be retrieved, leading to an n fold load to the server. In the second approach, skipping frames and its variations based on segments [2], only the n^{th} frames are displayed at the normal playback rate. Building streams for interactive operation only by special encoding schemes is the third strategy [10, 22]. Some other researchers have considered the disk model, such as the RAID [21] disk model, and multicast [20] for client interactive request support. On the proxy side, some solutions to support client interactive requests for Internet streaming delivery have also been proposed to create additional data files, called summarization [16] or hotspots [9], containing discontinuous but representative scenes of a media object, and cache them separately from the media files. In [16], the proxy is responsible for shot boundary detection and key-frame selection and summarization. A client can preview the summarization to decide whether he/she is going to continue the request. The summarization and media data can be both cached in multiple cooperative proxies. While in [9], the proxy only caches the hotspots to reduce client interaction requests (normally short) to the video servers which house the videos. By relating these summarization and hotspots to the video data, they provide support to fast forward and rewind. In addition, they can also support the preview function. However, they have not considered the support of client jump accesses.

Compared to existing work, our approach is the first to consider client jump access support, which is a majority of client interactive requests. Our dynamic interleaved segment caching not only considers the client jump access pattern, but also the sequential access pattern.

7 Conclusion

With the increase of the amount and playback length of streaming media objects over the Internet, more and more client interactive requests demand efficient support. Previous research efforts in the media server or the proxy paid little attention to client jump access, which is one of the dominant client interactive requests. By studying the streaming media workload collected from thousands of broadband home users through a large ISP, we found that not only has client access locality improved significantly in the past three years, but the dominant client jump accesses often suffer large delay due to the slow buffering rate through the network. To reduce the response time to interactive requests, we propose dynamic interleaved segment caching, where the object segments are cached according to the client access pattern dynamically. Conducting simulation-based evaluation, we show our proposed algorithm can greatly reduce the response time to client jump accesses, while slightly sacrificing proxy cache performance.

Our future work includes evaluations of DISC against BISC and CSC based on synthetic and collected Internet workloads. To generate typical synthetic workloads for our evaluations and for other researchers, we will study more detailed client jump patterns in extensive traces. Evaluations of different thresholds to give other recommendations under different patterns are also planed.

Acknowledgments

We would like to thank Oliver Spatscheck for several technical discussions about this study. The comments and suggestions from anonymous reviewers are constructive and helpful. We thank William L. Bynum for reading the paper and his comments. This work is partially supported by the National Science Foundation under grants CNS-0098055, CCF-0129883, and CNS-0405909.

References

[1] J. M. Almeida, J. Krueger, D. L. Eager, and M. K. Vernon. Analysis of educational media server workloads. In *Proc. of ACM NOSSDAV*, June 2001.

[2] M. Chen, D. D. Kandlur, and P. Yu. Support for fully interactive playout in a disk-array-based video server. In *Proc. of the International Conference on Multimedia*, Oct. 1994.

[3] S. Chen, B. Shen, S. Wee, and X. Zhang. Adaptive and Lazy Segmentation Based Proxy Caching for Streaming Media Delivery. In *Proc. of ACM NOSSDAV*, June 2003.

[4] S. Chen, B. Shen, S. Wee, and X. Zhang. Designs of high quality streaming proxy systems. In *Proc. of IEEE INFOCOM*, Mar. 2004.

[5] M. Chesire, A. Wolman, G. Voelker, and H. Levy. Measurement and analysis of a streaming media workload. In *Proc. of the 3rd USENIX Symposium on Internet Technologies and Systems*, Mar. 2001.

[6] C. Costa, I. Cunha, A. Borges, C. Ramos, M. Rocha, J. Almeida, and B. Ribeiro-Neto. Analyzing client interactivity in streaming media. In *Proc. of the International World Wide Web Conference*, May 2004.

[7] C. Cranor, T. Johnson, and O. Spatscheck. Gigascope: a stream database for network applications. In *Proc. of ACM SIGMOD*, June 2003.

[8] J. Dey-Sircar, J. Salehi, J. Kurose, and D. Towsley. Providing VCR capabilities in large scale video servers. In *Proc. of the International Conference on Multimedia*, Oct. 1994.

[9] H. Fahmi, M. Latif, S. Sedigh-Ali, A. Ghafoor, P. Liu, and L. Hsu. Proxy servers for scalable interactive video support. *IEEE Computer, Special Issue on Continuous Media on Demand*, 34(9):54–60, Sept. 2001.

[10] D. Gall. MPEG: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, 1991.

[11] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer fi le-sharing workload. In *Proc. of ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2003.

[12] L. Guo, S. Chen, Z. Xiao, and X. Zhang. Analysis of multimedia workloads with implications for Internet streaming. In *Proc. of the 14th International World Wide Web Conference*, May 2005.

[13] L. He, J. Grudin, and A. Gupta. Designing presentations for on-demand viewing. In *Proc. of ACM Conference on Computer Supported Cooperative Work*, Dec. 2000.

[14] E. Kusmierek, D. H. Du, and Y. Dong. Proxy-assisted periodic broadcast architecture for large-scale video streaming. *Journal of Internet Technology*, Oct. 2004.

[15] E. Kusmierek, Y. Lu, and D. H. Du. Optimizing periodic broadcast resource requirement with proxy. In *Proc. of IEEE ICME*, June 2004.

[16] S. Lee, W. Ma, and B. Shen. An interactive video delivery and caching system using video summarization. *Computer Communications*, 25:424–435, Mar. 2002.

[17] J. Liu, X. Chu, and J. Xu. Proxy cache management for fi ne-grained scalable video streaming. In *Proc. of IEEE INFOCOM*, Mar. 2004.

[18] J. Nichols, M. Claypool, R. Kinicki, and M. Li. Measurements of the congestion responsiveness of windows streaming media. In *Proc. of NOSSDAV*, June 2004.

[19] J. Padhye and J. Kurose. An empirical study of client interactions with a continuous media courseware server. In *Proc. of ACM NOSSDAV*, July 1998.

[20] R. Rangaswami, Z. Dimitrijevic, E. Chang, and S. G. Chan. Fine-grained device management in an interactive media server. *IEEE Transactions on Multimedia*, 5:558–569, Dec. 2003.

[21] M. Reisslein, F. Hartanto, and K. W. Ross. Interactive video streaming with proxy servers. *Information Sciences: An International Journal*, 140(1):3–31, Dec. 2001.

[22] P. Shenoy and H. Vin. Effi cient support for scan operations in video servers. In *Proc. of the International Conference on Multimedia*, Nov. 1995.