RCache: Design and Analysis of Scalable, Fault Tolerant Multimedia Stream Caching Schemes

Katherine Guo
Dept. Network Software Research,
Lucent Bell Labs,
kguo@dnrc.bell-labs.com

Milind M. Buddhikot
Dept. Network Software Research,
Lucent Bell Labs,
milind@dnrc.bell-labs.com

Youngsu Chae
College of Computing,
Georgia Institute of Technology,
yschae@cc.gatech.edu

Subhash Suri
Dept. of Computer Science,
University of California, Santa Barbara,
suri@cs.ucsb.edu

ABSTRACT

In the current Internet, web content is increasingly being cached closer to the end-user to reduce network and web server load and therefore improve performance and user perceived quality. Existing web caching systems[1] typically cache entire web documents and attempt to keep them consistent with the origin server. This approach works well for text and images; for bandwidth intensive multimedia data such as audio and video, caching entire documents is not cost effective and does not scale. An alternative approach is to cache parts of the multimedia stream on different caches in the network and coordinate stream playback from these caches. In such a case, the collection of cooperating distributed caches act as a single cache that is both scalable and fault-tolerant. This paper focuses on the design and the evaluation of novel data placement and replacement techniques for such distributed caches. Specifically, we propose schemes that work together: (1) *RCache*, a family of easy-to-implement, fault-tolerant multimedia data layout schemes that incorporate novel intra- and inter-clip replication controls, (2) *TwoD*, a two-dimensional *local* data replacement scheme based on the concept of segment popularity used for data replacement ordering at each cache.

Our schemes optimize storage space, start-up latency, server load, network bandwidth usage, and overhead from playback switch-overs. Our simulation results show that the RCache schemes provide 4 - 9 times higher cache hit ratio than a comparable traditional web caching system with the same amount of storage space.

Keywords: Web caching, streaming media, distributed caching

1. INTRODUCTION

The exponential increase in the number of Internet users in recent years has strained the Internet infrastructure. Scalable, cluster-based web servers and intelligent data caching have been two of the many tools used to alleviate this problem. However, with the proliferation of new Internet services that use multimedia data such as audio and video, this performance problem will become worse.

In the area of multimedia streaming services, extensive research has attempted to devise solutions that scale in terms of number of users and stream bandwidth at the server and client side. This research includes: (1) Scalable, high performance server and file system or operating systems architectures.[2,8,13,24] (2) Network and application level multicast techniques[6,7,18–20,22,27] that trade bandwidth and buffer space for interactivity and personalization. (3) Limited caching schemes such as stream patching[17] and prefix caching[23] that improve latency and playback quality perceived by the end-user. Recent investigation of audio/video data on the Internet and in large test-beds[3,5] reveals some important properties: (1) Bandwidth, duration, and therefore size of video files are gradually increasing. (2)

Video, audio files are WORMS, i.e. Write Once Read Many files; once created they seldom change and have a long life. (3) Accesses to such data exhibits high temporal locality. These properties suggest that in addition to the above techniques, efficient in-network caching of these multimedia objects can reduce network and server load, and improve start-up latency and quality perceived by the end-user.

Current web caching techniques for streaming multimedia content[1] typically use stand-alone systems that independently cache entire multimedia clips in response to user requests. This approach of demand-driven replication of same content in different caches is only adequate temporarily for small sized clips and does not work well for the large, high bandwidth clips that are becoming common.[5] For example, a single, two hour long MPEG-2 movie requires about 4.5 GB of disk space. Such large objects pose two problems: (1) Replicating the movie at different caches leads to waste of space. (2) Given a fixed investment in storage space at each cache, only a few media clips can be stored, and therefore, cache hit ratio and cache efficiency is limited.

An incremental improvement to this approach is *selective replication* that distributes clips in their entirety to a selected number of caches in the network. However, optimal replication is difficult and the scheme can lead to severe load-imbalance, high server load, and storage inefficiency.

An alternate solution to the selective replication is to break media objects into smaller segments and distribute one or more copies of each segment in a system of caches connected by a storage, local or metro area network. This idea of data striping has been used widely in disk arrays such as RAIDs,[16] and distributed cluster-based video-on-demand servers.[2,8,9,13] Such a scheme can reduce the required storage space compared to replication schemes and provide the added advantages of load-balance, fault tolerance, large parallelism, high cache hit ratio, and reduced network and server load. The scheme used for segment layout and replacement directly affects the efficiency of the caching system. This paper focuses on the design and analysis of such a distributed caching system.

In this paper, we propose a novel distributed caching architecture and associated schemes that work together. Specifically, we describe the following: (1) *RCache*, a family of randomized, easy-to-implement distributed cache layout schemes that account for long term clip popularity and intra-clip segment popularity metrics and provide parameters to tune storage efficiency and cache hit performance. (2) *TwoD*, a two-dimensional *local* data replacement scheme based on the concept of *segment access potential*, used for data replacement at each cache. Our schemes optimize storage space, start-up latency, server load, network bandwidth usage, and overhead from playback switch-overs. Our simulation results show that the RCache scheme provides 4 - 9 times as high cache hit ratio as that of traditional web caching system while utilizing the same amount of storage space.

The rest of this paper is organized as follows: Section 2 compares and contrasts relevant work in the areas of co-operative distributed caching and distributed data layouts in Video-on-Demand servers. Section 3 describes our streaming cache architecture and introduces popularity measures we use in our design. Section 4 presents the design and analysis of our RCache distributed data layout and provides guidelines for choosing various associated parameters. Section 5 describes our cache replacement policy that exploits the special structure inherent in data layouts to optimize storage performance. We also describe the performance results from simulation experiments in Section 6. Finally, Section 7 presents the conclusions and future work.

2. RELATED WORK

There are two bodies of work related to streaming cache design, namely web caching for regular objects and distributed video server design.

In co-operative caching systems, instead of operating independently, multiple caches cooperate to exploit cached content at neighboring caches. Such systems require protocols to organize caches into a topology and also for discovering nearest caches from whom to retrieve content. Harvest[11,15] represents one of the first *hierarchical caching* systems wherein the caches are *statically* configured in a hierarchy such that on a cache miss, the request is forwarded to the parent cache. The obvious drawback of such a system is the additional latency incurred in traversing the hierarchy when the requested object could be present in a neighboring (sibling) cache. This problem is fixed in Squid[1] using *static cooperative caching* where a cache always checks a fixed set of neighboring caches for any missing object before requesting it from its parent. A more sophisticated approach called *dynamic cooperative caching* used in Akamai[1] and Lucent *immi*net[1] extends the Squid ideas. It uses a centralized server that constantly computes a global map of the caching system. Using a cost function, the client's request is then sent to a cache with the least cost. Our data layout schemes are designed for such cooperative caching systems and can be easily implemented in their architectural frameworks.

The Middleman proxy architecture described in[4] has some similarities to our work but also, significant drawbacks as follows: (1) Like our architecture, it operates a collection of proxies as a scalable cluster cache. It consists of collection of two kinds of proxies; storage proxies that store segments of clips and local proxies that serve the clients. The collection of these proxies is coordinated by a centralized coordinator which requires backup *cohorts* to ensure fault-tolerance. Our architecture does not use a centralized coordinator. (2) Similar to our layouts, Middleman splits clips into smaller sized segments and caches only one copy of each segment in the proxy cluster. In our layouts, we account for global and local popularity of clip and its segments and store variable number of copies of each segment. Compared to middleman, our approach guarantees better load balancing, fault tolerance and parallelism. (3) Middleman uses demand-driven data fetch and a modified version of LRU-K local cache replacement policy called *HistLRUpick*. We use a novel *TwoD* data replacement policy that accounts for the clip and segment popularity metrics. We believe our scheme accurately captures popularity of clip segments.

Several research efforts, such as Project MARS,[13] Microsoft TIGER file system,[9] Server Array,[8] and research reported in the literature[16,24,25] have addressed the problem of distributed or striped layouts. The primary focus of these schemes however, has been on video servers constructed using disks or tightly coupled clusters. Also, they focus on optimizing data layouts for high concurrency and balanced operation of clusters under normal playout and interactive playback operations such as fast-forward and rewind and reduce origin server and network load. Our layouts are meant for loosely coupled caching systems and are quite different.

3. DISTRIBUTED CACHING ARCHITECTURE

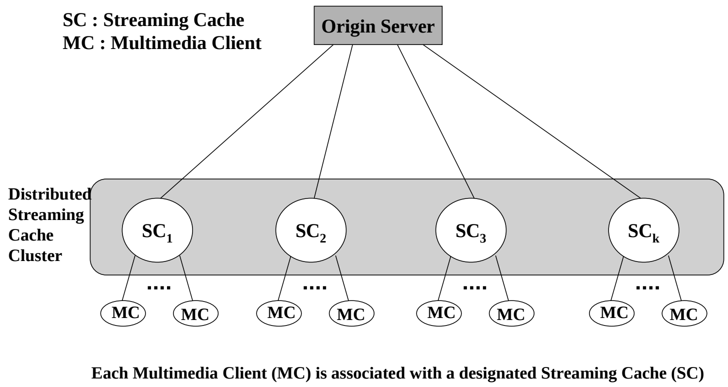


Figure 1. Distributed streaming cache architecture

Figure 1 presents our distributed caching architecture in which requests from a multimedia client (MC) for multimedia clips on an origin server are always sent to a designated Streaming Cache (SC) or proxy[1] associated with it. This association may be established manually (by entering the address of SC in a "preferences" field in the multimedia player such as RealPlayer, Windows Media Player or QuickTime Player) or may be established transparently using a layer-4 switch or a DNS server for redirection as in.[1] The origin server in our architecture is a streaming server in the content provider infrastructure such as `cnn.com` or a Video-on-Demand (VoD) server in a regional data center of a Metro Service Provider (MSP). Depending on the type of the network that connects the proxies, different scenarios are possible. For example, (1) **Cluster Caching**, where each proxy can be a storage node

[1]Here after, we use the terms SC, cache, and proxy interchangeably.

in a large cluster cache interconnected by a Storage Area Network (SAN). (2) **Metro Caching**, where each proxy can be an independent cache deployed in buildings interconnected by a Metro Area Network (MAN). (3) **Enterprise Caching**, where each proxy can be an independent cache deployed in different departments in an enterprise building and interconnected by a fast enterprise network. (4) **Wide Area Network Caching**, where each proxy can be an independent cache in the Internet. Recent results in the literature show that cooperative caching is useful in small-to-medium scale networks, principally in LAN and MAN environments.[29] Therefore, we are primarily interested in scenarios 1, 2, and 3 above. In all these cases, the proxies together form a *proxy or cache cluster* with K members and each proxy is aware of the other members or siblings. Also, the proxies exchange state information among themselves using a inter-cache protocol, for example a variant of ICP.[28]

In our architecture, we divide each media clip into relatively small segments, and distribute them among different proxies in the network using a demand-driven approach. The design of our RCache layouts combined with demand-driven caching, guarantees that each segment is stored in *at least* one cache. Each proxy keeps a *potential local segment map* recording if a segment of a clip is *supposed* to be stored in the cache according to the RCache layout scheme and an *actual local segment map* recording if a segment is *actually* stored in the cache. For example, for segment j of the clip M, $PotentialLocalSegmentMap[M, j] = 1$, and $ActualLocalSegmentMap[M, j] = 0$ means that this segment is supposed to be stored locally but it has not be stored yet. The state exchange of local segment maps among proxies guarantees that each proxy knows the clips and segments present in the cluster and has a *global segment map* that describes which proxies store which segments of every clip both potentially and actually.

Next, we describe the mechanics of data layout creation and cache behavior, namely cold start, cache hit and cache miss for a clip M. (Note that when we talk about cache hit and miss for segments, we use the terms "segment cache hit" and "segment cache miss".)

Cold Start for Clip M (Data Layout Creation): Suppose one of the proxies SC_i receives a request from a client MC for a clip M which is not currently present in the cache cluster. In other words, SC_i is the designated SC of MC, and there is no information about clip M in the global segment map. The proxy undertakes three steps: (1) It queries the origin server to find out popularity information for M and uses the design of our distributed *RCache* layout (Section 4) to compute a potential local segment map $PotentialLocalSegmentMap$ that describes which segments in the clip it should cache. (2) It fetches the first segment of the clip from the origin server, stores it locally, sets $ActualLocalSegmentMap[M, 1] = 1$, and concurrently streams it to the client. (3) It informs the other proxies of the appearance of this new clip. They each compute their own potential local segment map $PotentialLocalSegmentMap$ based on the data layout design. The proxies exchange this map immediately to construct a consistent global segment map that describes the *supposed presence* of segments in the cluster.

For later segments, there are two cases: (1) If the potential local segment map at SC_i says the proxy should store the particular segment, SC_i will fetch the segment from the origin server, store it locally, concurrently streams it to the client, and set $ActualLocalSegmentMap[M, 1] = 1$. (2) If the potential local segment map at SC_i says the proxy should not store the particular segment, it consults the global segment map and redirects the request to another cache SC_j that is supposed to have the segment. If SC_j does not have the segment in its local storage, it first checks to see if any of its siblings have it. If they do, it fetches the segment, stores it locally, and also streams it to SC_i. If no sibling proxies have the segment, SC_j fetches the segment from the origin server, stores a local copy, and also streams it to SC_i. Upon receiving the beginning of the stream from SC_j, SC_i immediately streams it to the MC. Whenever the segment can be requested from multiple proxies, a proxy server selection algorithm can be used to balance load in the proxy cluster. Our layout design virtually guarantees that each segment is fetched from the origin server only once. Also, the demand driven mechanism of layout creation guarantees that storage is utilized just-in-time when there is a request for the clip.

Cache Hit for Clip M: When the proxy SC_i receives a request for a segment j of clip M, one of the following scenarios happens: (1) Requested segment j is present in the local storage: In this case, SC_i streams that segment to the client. (2) Segment j is supposed to be present in the cache ($PotentialLocalSegmentMap[M, j] = 1$), but it has not been cached yet or has been flushed out due to local replacement($ActualLocalSegmentMap[M, j] = 0$): In this case, SC_i consults our *TwoD* local cache replacement policy to allocate space for this segment and fetches the segment from a sibling cache or the origin server and streams it to the client. (3) Segment j is not supposed to

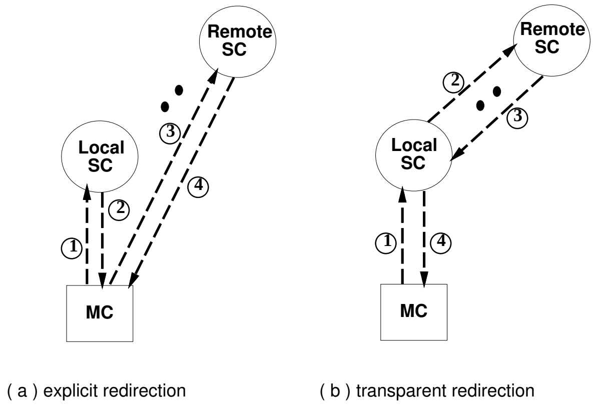


Figure 2. Segment request redirection

be cached locally ($PotentialLocalSegmentMap[M, j] = 0$), but the global segment map says it is present in one or more sibling caches. In this case, SC_i redirects the request to the next cache S_{next} that is selected by consulting a proxy server selection algorithm that attempts to balance the load in the proxy cluster.

Cache Miss for Clip M***:*** A cache miss happens for a clip in two cases: (1) when the clip is requested for the first time, (2) when all of its segments are flushed out of the proxy cluster. The first case is identical to cold start, and the second case can be handled segment by segment as in case (2) or (3) of cache hit. We later use analysis and simulations to show that our architecture provides excellent cache miss performance.

Note that when the popularity of a clip changes, the total amount of storage used in the caching system should reflect this change and modify the data layout accordingly. Also, periodic updates of global and local cache maps remove information about clips that are no longer cached anywhere in the proxy cluster.

Clearly, the cluster of proxies acts as a single, loosely-coupled streaming cache to serve client playback requests. One undesirable artifact of data segment distribution is that two consecutive segments may have to be retrieved from different caches, requiring signaling to switch between two streams coming on two different connections. Such switch-over is a potential source for disruption of client playback and deterioration of quality. However, using techniques such as double buffering or fetching data at a higher rate, playback from different proxies can be pipelined and disruption can be minimized. Figure 2 shows two different mechanisms to deal with the request redirections that occur in case of cache miss or stream switch-overs.

The first mechanism is an explicit redirection as in Figure 2(a). When the requested segment is missing from the designated proxy SC_i , SC_i sends the requesting MC the information of a remote proxy SC_j that stores the requested segment or it sends the information of the origin server if no proxy in the cluster stores the segment. Then the MC fetches the segment directly from the remote SC_j or the origin server. In the explicit redirection mechanism, the MC must be modified to handle the segment request redirection. However, modifying client software is highly undesirable for the deployment of any caching system.

For this reason, we follow a transparent redirection mechanism shown in Figure 2(b) where the request redirections for segment cache miss are handled by the designated proxy SC_i and transparent to the MC. When the requested segment is not stored in SC_i, SC_i fetches the segment either from a remote SC_j or from the origin server, and concurrently streams the segment to the requesting MC. In the transparent redirection mechanism, the MCs are transparent to cache miss and switch-overs. In addition, the transparent mechanism can support request aggregation by adopting the ring buffer mechanism[10] and can substantially reduce network traffic by supporting multiple requests by a single stream from the remote SC.

In this architecture, we use the following performance metrics to assess our schemes: (1) total storage requirement, (2) playback start-up latency, (3) number of switch-overs, (4) origin server hit ratio and (5) cluster cache hit ratio, also called system-wide cache hit ratio. The goal for designing a data layout algorithm is to balance the trade-offs between these parameters.

3.1. Clip and Segment Popularity

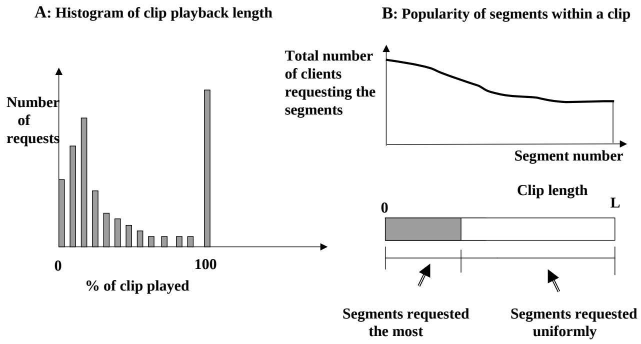


Figure 3. Bimodal distribution of requests for clip segments

In our algorithms, the popularity of clips and segments plays a critical role. We use two popularity metrics: (1) *Global Clip Hotness Rating (GCHR)* a macro-metric that captures the global popularity of each clip, and (2) *Local Segment Hotnessq Rating (LSHR)*, a micro-metric that captures the popularity of segments of the clips.

Global Clip Hotness Rating(GCHR) of a clip is defined as the the total number of requests for the clip per unit time in the proxy cluster. At each proxy, the total number of requests to the clip per unit time is called Local Clip Hotness Rating (LCHR). When a new media clip is stored on the origin server, the proxy cluster may set its GCHR value based on a priori knowledge of the expected popularity of the clip. of the week, CNN Each proxy also maintains a LCHR for each clip, and the proxy cluster can compute the GCHR for a clip by adding its LCHR value at different proxies. Current understanding of access patterns of streaming media on the web is rather limited and often the Zipf distribution is used to model them.[12,5] Zipf's law relates probability of access p of a clip and the popularity of the clip described by a rank metric i as $p = k(\frac{1}{i^\gamma})$ where $k = \sum_{j=1}^{n}(1/j^\gamma)$, n is the total number of clips in the proxy cluster and γ is in the range[2] $0.8 \leq \gamma \leq 1.2$. In our work, GCHR can be approximated by a Zipf distribution or dynamically collected over time.

Recent measurment studies have shown that all parts of a clip are not requested with the same probability and in fact, the segment playback probability follows a bimodal distribution in playback length.[5] As shown in Figure 3 (a), a fraction of the requests for a clip complete the playback, whereas the rest terminate after partial playback. Among the partial playback requests, the probability of playback exceeding a threshold drops rapidly as the playback length is increased. Clearly, segments in the beginning of clips are requested more often than later segments as in Figure 3 (b). The popularity of a segment includes all playback requests that last long enough to access the segment.[3] Two important implications of such access patterns are: (1) A clip does not need to be cached in its entirety at each cache; even partial segment caching can provide caching gain.[4] (2) Segment layout must account for *segment popularity* in addition to clip popularity. Each proxy in our architecture maintains a Local Segment Hotness Rating (LSHR) for each segment, which is the total number of requests for the segment during unit time. We use the GCHR and LSHR in the design of our data placement and replacement schemes. Note that these parameters can be measured over time and their avergaed values can be used in actual layouts.

[2]The exact value of γ is determined by solving a curve fitting problem of mapping the series of GCHR for all the clips in the system to the series of their ranks. In other words, we calculate γ from GCHR of all the clips.

[3]Segment popularity in Figure 3 (b) can be regarded as a complementary cumulative distribution function of the playback time distribution in Figure 3 (a).

4. RCACHE: RANDOM CACHING OF MEDIA FILE SEGMENTS

This section starts with the basic random data layout scheme *RCache-I*, and later expands it into a family of layout schemes.

4.1. RCache-I: Basic RCache Scheme

The *RCache-I* data layout scheme randomly distributes segments of media clips among the cluster of K SCs. It divides the media clip of length L into N equal-length segments and lets each SC independently decide which segment to cache. The goal of this scheme is to store all the segments within the proxy cluster, so that requests for the clip can be served entirely out of the cluster, without involving the origin server. Specifically, for a constant a ($0 \leq a < K$), each SC independently decides with a fixed probability a/K whether to store a particular segment. Notice the probability is proportional to $1/K$ and independent of both the segment number and the proxy location. The algorithm is as follows:

```
/* Given constant 1 ≤ a < K, compute p = a/K */
FOR (all segments in the clip )
     /* Generate random number with uniform distribution from 0 to 1 */
     r = UniformRandomGen(IPAddr)
     IF (0< r < p)) THEN
        Cache the segment
     ELSE
        ignore the segment
     END
END
```

The estimated total number of segments of the clip stored at each SC is then $(a \times N)/K$. We need to determine the constant a such that the probability for the entire clip to be stored in the proxy cluster is sufficiently large. In other words, we are interested in the probability of the event – "there exists no segment which is not cached in the cluster of K SCs" or "each segment is cached at least once in the cluster of K SCs."

$$\begin{aligned}
\text{Pr(Media clip is cached in the proxy cluster)} &= PR(\text{Each segment is cached at least once in } K \text{ SCs}) \\
&= 1 - Pr(\text{At least one segment is not cached in any SC}) \\
&= 1 - Pr(\bigcup_{1 \leq i \leq N} E_i) \qquad (1)
\end{aligned}$$

where E_i is the event that segment i ($1 \leq i \leq N$) is not cached at any of K SCs. Using the Inclusion-Exclusion Principle of probability: $Pr(\bigcup_{1 \leq i \leq (N)} E_i) \leq NPr(E_i)$, we can rewrite Equation 1 as follows.

$$\text{Pr(Media clip is cached in the cluster)} \geq 1 - NPr(E_i) \qquad (2)$$

The probability that segment i is not stored by cache SC_k is $P_{uncache}(i,k) = 1 - \frac{a}{K}$. The probability that segment i is not stored by any SC is then

$$Pr(E_i) = P_{uncache}(i) = (1 - \frac{a}{K})^K \qquad (3)$$

Therefore, Equation 2 can be rewritten as

$$\text{Pr(Media clip is cached in the cluster)} \geq 1 - N(1 - \frac{a}{K})^K \qquad (4)$$

Since $(1-\frac{1}{K})^K \rightarrow e^{-1}$, as $K \rightarrow \infty$, we have $N(1-\frac{a}{K})^K \leq Ne^{-a}$ or $1-N(1-\frac{a}{K})^K \geq 1-Ne^{-a}$. Thus, if we choose $a = c\ln N$ where c is a constant, we get

$$\begin{aligned} \Pr(\text{Media clip is cached in the cluster}) &\geq 1-\frac{N}{e^{c\ln N}} \\ &\geq 1-\frac{1}{N^{c-1}} \end{aligned} \tag{5}$$

As per Equation 5, the probability that the entire clip is cached in the cluster is *at least* $(1-\frac{1}{N^{c-1}})$, which is a desirable *lower* bound. Clearly, we can control this lower bound by changing c and N, and *virtually* guarantee that the entire clip is cached in the cluster. Even with $c = 2$ and $N = 10$, clip caching probability is very high – $1-(1/N) = 0.9$, which may be sufficient in practice.

To evaluate storage requirement, the naive replication approach where each clip is stored in every proxy in the system is used as a baseline which requires a LK storage. The storage efficiency factor S_{eff} is defined as the ratio of the total storage of any scheme over that of the baseline scheme.

The *RCache-I* scheme will be a desirable scheme if its storage requirement is smaller than LK. That is, S_{eff} should be less than 1. From the discussion above we can see that the storage requirement of RCache scheme is $K(aN/K)$. Therefore, the $S_{eff} = aN/NK$ or $(c\log N)/K$. Clearly, as long as $c\ln N < K$, $S_{eff} < 1$ and the RCache strategy is superior.

Observe that for a given c, the number of segments in the stream N that makes RCache superior be computed as $N < e^{K/c}$. Consider a simple example: if we have $K = 10$ caches, and $c = 2$, then the number of clip segments should be less than $e^{10/2} = e^5 = 149$, in order for RCache-I to be better than the naive strategy. Notice with a fixed c, a decreasing N decreases S_{eff}, and therefore increases storage efficiency. Also notice, decreasing N decreases lower bound of the probability that the entire clip is cached in the cluster, $(1-\frac{1}{N^{c-1}})$. In practice, the value of N should be chosen to balance the tradeoff between the these two factors.

In the RCache-I scheme above, segments of constant length are stored at caches with constant probability. Each segment is virtually guaranteed to be in one of the caches while keeping the storage requirement less than $O(NK)$ (or NK to be precise) – the simple case where each segment is stored in every cache. However, under RCache-I, the user perceived start-up latency can be large if the client's designated SC does not store the first segment or even worse, the first segment is not stored in any of the SCs in the cluster. In the second scenario, the request has to be redirected to the origin server. We can eliminate these scenarios if the layout guarantees that the first segment of the clip is always available on the designated cache. The *RCache-II* scheme achieves this by caching the first segment of each clip with unit probability at each cache.

We have two degrees of freedom in the design of RCache algorithm: (1) how to divide a clip into segments, (2) how to store the segments. In the following, we describe two variants of RCache-II, namely (1) *RCache-III* scheme that accounts for intra-clip segment popularity and (2) *RCache-IV* scheme that accounts for both long term clip popularity and intra-clip segment popularity. We also provide guidelines for how relevant parameters for these data layouts should be selected. The schemes that use segment size variation are more involved and are considered in another paper.[14]

4.2. RCache-III with Intra-clip Replication Control

Given a clip of length L, RCache-III divides it into N segments of equal size δ and uses the segment number and a intra-clip segment caching probability (SCPR) β parameter to decide the segment caching at each SC. Specifically, the probability to store the first segment is 1, but the probability to store the second segment drops to $1/\beta$. In general, up to the m^{th} segment, the probability with which segment i is stored at each cache is $1/\beta^{i-1}$. For the m^{th} segment, the caching probability is set to its its minimum value and is fixed at $1/\beta^{m-1}$ from that point on.

Table 1 illustrates the segment size and the caching probability variation. The main principles used in this design are: (1) *Minimize playback start-up latency:* To this end, the layout always caches the first segment of the clip at each cache. (2) *Achieve better load balance and fault-tolerance*: Unlike Middleman[4] proxy architecture that stores only one copy of constant length segments, RCache layout stores redundant copies of popular segments and attempts to guarantee at least one copy of each segment. This requires modestly more storage but statistically guarantees better load distribution in the proxy cluster, better fault tolerance, and higher parallelism.

Table 1. Details of RCache-III data layout

Segment number	Segment size	Pr(Cache Seg)	Expected # of caches that store the segment	Expected total storage for the segment
1	δ	1	K	δK
2	δ	$1/\beta$	K/β	$\frac{\delta K}{\beta}$
3	δ	$1/\beta^2$	K/β^2	$\frac{\delta K}{\beta^2}$
...				
m	δ	$1/\beta^{m-1}$	K/β^{m-1}	$\frac{\delta K}{\beta^{m-1}}$
$m+1$	δ	$1/\beta^{m-1}$	K/β^{m-1}	$\frac{\delta K}{\beta^{m-1}}$
$m+2$	δ	$1/\beta^{m-1}$	K/β^{m-1}	$\frac{\delta K}{\beta^{m-1}}$
...				
...				
$m+R$	δ	$1/\beta^{m-1}$	K/β^{m-1}	$\frac{\delta K}{\beta^{m-1}}$

From the last column in the Table 1, we can see that the total storage requirement for the generalized RCache scheme is

$$T = K\delta\left(\sum_{i=0}^{m-1}(\frac{1}{\beta}^{i}) + \frac{R}{\beta^{m-1}}\right) \tag{6}$$

The storage efficiency factor in this case is

$$\begin{aligned} S_{eff} &= \frac{T}{LK} \\ &= \frac{\delta\left(\frac{R}{\beta^{m-1}} + \sum_{i=0}^{m-1}(\frac{1}{\beta})^i\right)}{L} \end{aligned} \tag{7}$$

Note that as long as the segment caching probability after the first segment is less than 1, S_{eff} is less than 1.

4.3. RCache-IV with both Inter- and Intra-clip Replication Controls

If there are multiple identical-length media clips and all the clips are equally popular, each clip can be segmented in the same way using identical values of the parameter set (δ, β, m). However, in reality, media clips differ in popularity and therefore, the caching system must allocate storage for clips in accordance to their popularity; more popular clips should be allocated more storage than less popular ones.

The RCache-III layout accounts for segment popularity within a clip using a probability sequence $U = (1, 1/\beta, 1/\beta^2, ...1/\beta^{m-1}, 1/\beta^{m-1}, ...1/\beta^{m-1})$; it caches popular segments with higher probability. A natural way to extend this for multiple clips ordered by global popularity rating (GCHR) is as follows: given two clips with popularity ratings

$GCHR_1$, $GCHR_2$ and associated caching probability sequences U_1, U_2, if $GCHR_1 > GCHR_2$, then set $U_1 > U_2$. Specifically, the probability sequence for a clip with GCHR i is

$$e(i,j) \otimes U = (e(i,1)u_1, e(i,2)u_2, \ldots, e(i,k)u_k)$$

The $e(i,j)$ is called the *caching ratio* and is defined as follows:

$$e(i,j) = \begin{cases} 1 & \text{if } j = 1 \\ x, \quad 0 \leq x \leq 1 & \text{otherwise} \end{cases}$$

Using the Zipf's distribution, for a movie of rank i, we set $e(i,j) = 1/i^\gamma$, for $j \neq 1$, where γ $(0.8 \leq \gamma \leq 1.2)$ controls decreasing ratio of inter-clip segment caching probability (ISCPR) and the relative caching preference among clips with different ranks. Higher γ gives more preference to popular clips. Table 2 shows the details of the configuration parameters and the cache layout for a clip of rank i.

Parameters: (1) δ: Size of each segment. (2)β **:** Decreasing Intra-clip Segment Caching Probability Ratio (SCPR). (3) γ: Exponent of the decreasing Inter-clip Segment Caching Probability Ratio (ISCPR).				
Segment number	Segment size	Prob(seg caching)	Expected no. of proxies caching the seg	Expected total storage for the seg
1	δ	1	K	δK
2	δ	$\frac{1}{\beta i^\gamma}$	$\frac{K}{\beta i^\gamma}$	$\delta \frac{K}{\beta i^\gamma}$
3	δ	$\frac{1}{\beta^2 i^\gamma}$	$\frac{K}{\beta^2 i^\gamma}$	$\delta \frac{K}{\beta^2 i^\gamma}$
$\vdots$				
m	δ	$\frac{1}{\beta^{m-1} i^\gamma}$	$\frac{K}{\beta^{m-1} i^\gamma}$	$\delta \frac{K}{\beta^{m-1} i^\gamma}$
$m+1$	δ	$\frac{1}{\beta^{m-1} i^\gamma}$	$\frac{K}{\beta^{m-1} i^\gamma}$	$\delta \frac{K}{\beta^{m-1} i^\gamma}$
$\vdots$			$\vdots$	
$m+R$	δ	$\frac{1}{\beta^{m-1} i^\gamma}$	$\frac{K}{\beta^{m-1} i^\gamma}$	$\delta \frac{K}{\beta^{m-1} i^\gamma}$

Table 2. RCache-IV data layout for rank i clip

The average per-proxy cache size of the clip of rank i is

$$S_i = \delta + \frac{\delta}{i^\gamma}\left(\frac{1}{\beta} + \frac{1}{\beta^2} + \ldots + \frac{1}{\beta^{m-1}} + \frac{R}{\beta^{m-1}}\right) \tag{8}$$

$$= \delta + \frac{\delta}{i^\gamma}\sum_{k=1}^{m-1}(\frac{1}{\beta})^k + \frac{\delta}{i^\gamma}\frac{R}{\beta^{m-1}} \tag{9}$$

Therefore, the total size of the RCache layout with K proxies and n clips is

$$T = \sum_{i=1}^{n} KS_i \tag{10}$$

The storage efficiency for a clip with rank i can then be computed as $S^i_{eff} = S_i/L$, and the total storage efficiency for n clips is computed as $S_{eff} = \sum_{i=1}^{n} \frac{S^i_{eff}}{n}$.

Note that the Middleman[4] proxy architecture stores only one copy of each segment and therefore always requires only Ln amount of storage for total of n clips each of length L. Clearly, RCache-IV requires $\sum_{i=1}^{n} KS_i/(Ln) = KS_{eff}$ times larger storage than Middleman.

4.4. Guidelines for Choosing Layout Parameters

Given the parameters L, K, S_{eff} the parameters that need to be decided before the RCache layout for the clip can be realized are δ, β, m, and γ. The constant segment size δ must be long enough to allow the client to start the pre-fetch pipeline for subsequent segments and to cover some initial portion of the bimodal distribution for user playback time. We believe that segment size of 1 to 2 minutes is sufficient. With an MPEG-2 stream of 5 Mbps average bandwidth, this results in the segment size of roughly $\delta = 35 - 75$ MB.

In RCache-III and RCache-IV, the first m segments have higher caching probability than the last R segments. It is natural to match these segments with the region of higher request probability. Assume the first ρ fraction of the clip playback period gets most accesses, m should be chosen to satisfy the following condition:

$$m\delta \geq \rho L \tag{11}$$

where ρ can be obtained from either LSHR or the observed bimodal distribution of user playback time.[5] A set of candidate (β, m) tuples can be calculated from either the bimodal distribution of playback time[5] or the LSHR.

Finally, γ can be obtained directly from GCHR. The γ can also be adjusted in conjunction with S_{eff} with a trade-off between S_{eff} and cache hit ratio for less popular clips. Note that increasing γ, improves S_{eff} and decreases the cache hit ratio for less popular clips.

5. LOCAL CACHE REPLACEMENT ALGORITHM

Each SC in our distributed caching system has a fixed amount of storage space. In the event its storage space is exhausted, it requires a local segment replacement policy that removes cached segments to create space for new segments. Each SC independently executes a local data replacement algorithm, the goal of which is to retain segments that maximize the cache hit ratio. RCache-IV data layout has an inherent two dimensional structure of relative ordering of importance among segments based on inter-clip and intra-clip popularity. In other words, clips themselves can be ordered in decreasing order of popularity (GCHR) and the segments in a clip can again be ordered based on the segment popularity. Efficient selection of segments for replacement requires that the two dimensional ordering structure to be taken into consideration, therefore, the name *TwoD* for our replacement algorithm.

The objective of the *TwoD* replacement algorithm is to keep the more popular clips and segments in the cache cluster, therefore maximize both local and system-wide cache hit ratio. In our architecture, each SC ranks all the clips according to their GCHR. Segments of each clip at each SC are ordered by timestamps. According to the bimodal distribution of clip request length, segments with higher timestamps are less popular than segments with lower timestamps. Figure 4 presents the local view of the state of the clips stored in the SC. Recall that the segments in the clip are stored at different SCs in the system and not every segment is present at each SC. This local view of the state of the clips stored at a SC is obtained by accounting presence or absence of segments. In this figure, the presence of a segment in a clip at a SC is represented by a "1", whereas absence by "0".

The two important observations that we use in the *TwoD* replacement algorithm are : (1) Since GCHR of the media clip reflects its popularity, clips with lower rating should be replaced first. (2) Since segments within a clip have decreasing popularity as their time stamps increase, segments within a clip should be replaced from its end. Our objective is to keep more popular clips in the cache: so during replacement if the GCHR of a clip is lower than a threshold λ, complete removal of the clip should be allowed. However, once that threshold is reached it is not advisable to completely remove a clip from the system; instead it is better to replace segments of all clips from the end.

Therefore, when the disk space is low, the replacement algorithm at each SC proceeds in two phases to create space for new segments: In the first phase, the clip with the lowest GCHR is dropped segment by segment from its end. This process repeats until the lowest rating for the remaining clips reaches a threshold λ. If still more space is needed, the algorithm enters a phase two where the clip with the lowest rating is chosen and its segments are flushed from the clip's end until a threshold μ % of the clip is left. Then the flushing process moves to the clip with the second lowest rating. The algorithm is as follows:

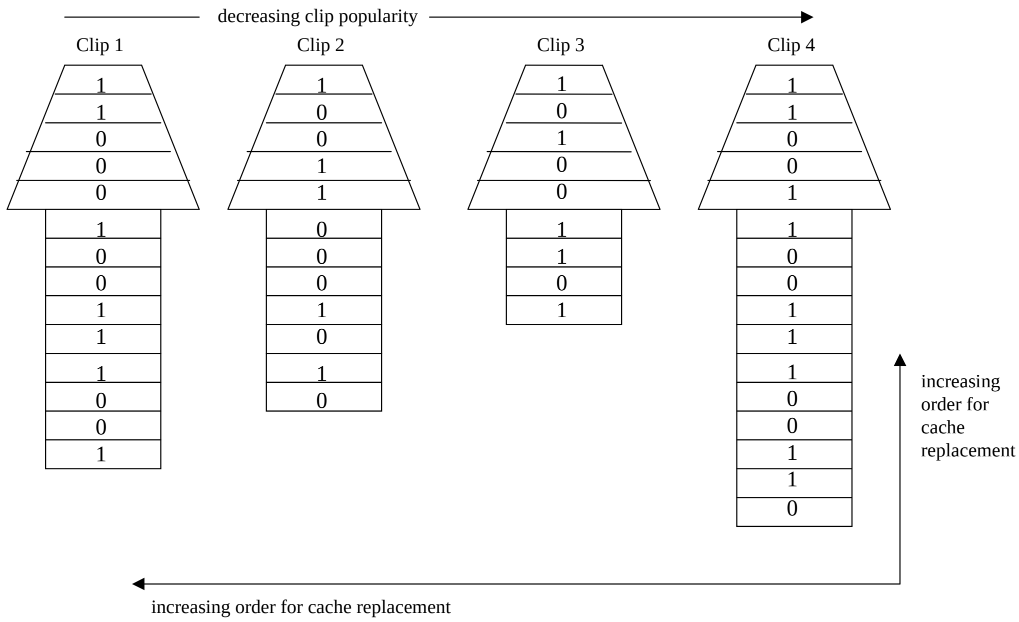


Figure 4. Two dimensional replacement order in $TwoD$

ALGORITHM 5.1 (CACHE REPLACEMENT).

```
[Complete]
rating = LowestRating();
while NeedMoreSpace() do
        x = GetClipWithRating(rating);
        comment:  PHASE 1
        if (rating ≤ λ)
          then
                ReplaceFromEnd(x, 1);
                if Empty(x)
                  then
                        rating = NextLowestRating();
                fi
                comment:  PHASE 2
        elsif (rating > λ)
              then
                   h = length(x);
                   comment: Replace clip x segment at a time from its end until the next
                   comment: segment to replace is at μ % from the clip's beginning.
                   ReplaceFromEnd(x, 1);
                   if (length(x) <= (100 − μ) ∗ h/100)
                     then
                           rating = NextLowestRating();
                   fi
        fi
od
```

Because GCHRs are consistent across all SCs in the system, for popular clips and for the beginning part of all the clips, the number of SCs that store each segment still follow the same pattern defined in the original data placement scheme. Therefore, the cache replacement algorithm with GCHR effectively preserves the data layout property of RCache data layout scheme.

If Local Segment Hotness Rating(LSHR) for each clip is maintained at each SC, then the replacement algorithm can be designed accordingly. Each SC ranks each segment according to its LSHR and segments are flushed one by one starting with the one with the lowest rating. Unlike GCHR, LSHR has local significance. The replacement algorithm based on local SHR does not require any global SHR, therefore there is no need to communicate LSHR among SCs. However, LSHRs are not consistent across SCs, therefore segment replacement is not consistent across SCs. This means the data layout determined by the RCache scheme is not likely to be preserved.

6. PERFORMANCE ANALYSIS

To analyze performance, we use the ns-2[26] simulation package to model the RCache distributed cache system. In case of static configuration of RCache, we assume that the entire cache layout is built at start-up and remains the same through the end of simulation. We believe that the static configuration is sufficient to capture the important performance characteristics of RCache caching system.

Table 3. RCache schemes with different set of features

RCache Scheme	First Segment Caching	Intra-clip Replication Control	Inter-clip Replication Control
RCache-I	No	No	No
RCache-II	Yes	No	No
RCache-III	Yes	Yes	No
RCache-IV	Yes	Yes	Yes

We compare the performance of a set of different RCache schemes as discussed Section 4. Table 3 shows the four different RCache schemes with various combinations of the features. For performance comparison, We measured the following metrics: (1) Local hit ratio (LHR), representing the percentage of stream data that is retrieved from local SCs, (2) Remote hit ratio (RHR), representing the percentage of stream data that is retrieved from remote SCs, (3) Server hit ratio (SHR), representing the percentage of stream data that is retrieved from the origin server, (4) Switch-over rate, representing the ratio of the total number of switch-overs to the total number of segment boundaries, and (5) First segment missing ratio, representing the percentage of the stream requests that fail to fetch the first segment from local SCs. All RCache schemes except RCache-I have zero first segment missing ratio since SCs always cache the first segments in their local cache.

Table 4 shows the set of parameter values used throughout the simulation study. Aggregate arrival rate of client requests per SC is 1 request/min. We use a simple mesh topology among SCs and unlimited link capacity since they are not important to our performance study.

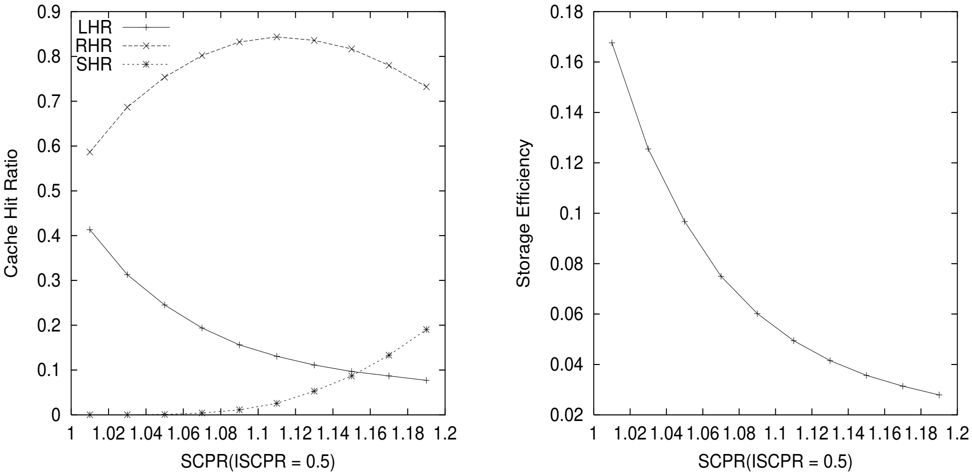
First, we consider the cache hit ratios of the RCache-IV with varying SCPR. Figure 5(a) shows the local, remote, and server hit ratios with varying SCPR. Local hit ratio (LHR) decreases as the SCPR increases. This is because, as SCPR increases, segment caching probability decreases more quickly and becomes smaller for later segments, resulting in less chance for the segments to be cached. The storage efficiency (S_{eff}) also shows the similar behavior to the LHR (see Figure 5 (b)), decreasing as the SCPR increases. This shows a trade-off between LHR and S_{eff}, requiring more storage space to improve the LHR. Thus, the SCPR parameter can be used to control the trade-off between LHR and S_{eff}.

Remote hit ratio (RHR), however, shows an interesting behavior. It increases as SCPR increases till a certain threshold value, but starts decreasing as SCPR increases further. This can be explained by considering the system-wide hit ratio – the sum of local and remote hit ratios.[4] In general, the system-wide hit ratio decreases as SCPR

[4] It is clear that the server hit ratio becomes (1 - system-wide hit ratio).

Table 4. Default Values of Simulation Parameters

Parameter	Default Value
Segment size	50 MB
Clip size	4.5 GB
Clip type	5 Mbps MPEG-2
Number of Clips	100
Number of SCs	100
Client request	Poisson process
Clients' preference to Clips	Zipf's distribution with $\gamma = 1$
Playback time distribution	Bimodal distribution[5]
Local Cache Size	12 - 75 GB depending on parameters



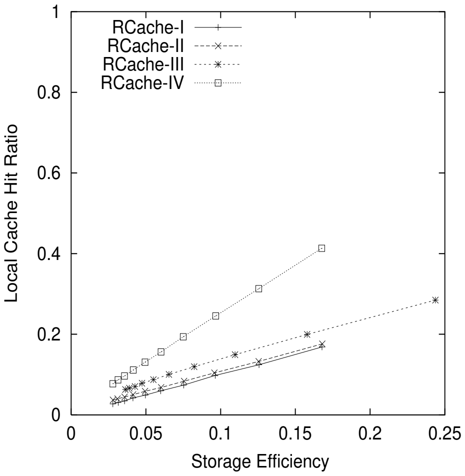
(a)Cache Hit Ratios (b)Storage Efficiency S_{eff}

Figure 5. Cache Hit Ratios and Storage Efficiency with varying SCPR

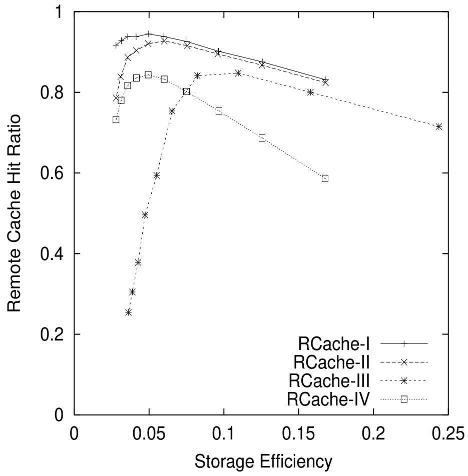
increases. However, for the RCache-IV, the system-wide hit ratio is almost 1 in some range of SCPR, where at least one SC inside the system caches each segment. In that range, the RCache-IV can serve client requests either through local designated SC or remote SCs, keeping the server hit ratio (SHR) almost zero. For example, in Figure 5 (a), with less then 8% of S_{eff} at SCPR of 1.07, the RCache-IV can cover almost 100% of stream requests through either local SC or remote SCs. The only difference, as SCPR varies in that range, is whether the requests are served through the local designated SC or from remote SCs. As SCPR increases beyond the threshold value, however, the chance that the segments are being cached within the RCache-IV decreases, resulting in increase of the server hit ratio.

Figure 6 shows the cache hit performance of the RCache schemes with varying S_{eff}. For LHR, RCache-IV performs best as in Figure 6 (a). The LHR strongly affects the performance of the entire caching system. Remote(server) cache hits have the request redirection overhead. Higher local cache hit ratio is more preferable considering the complexity of the redirection and the switch-over mechanisms. RCache-IV with both the intra-/inter-clip replication controls gives the popular segment higher chance of being cached, resulting in the best LHR. RCache-III shows the next best LHR performance with the aid of the intra-clip replication control. However, the intra-clip replication control does not show the significant improvement compared to the inter-clip replication control. RCache-II slightly outperforms RCache-I with the aid of the first segment caching option.

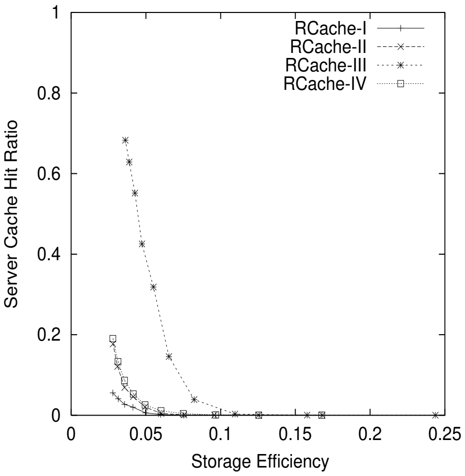
For RHR, all RCache schemes show the similar shape as in Figure 6 (b), increasing up to some point and starting



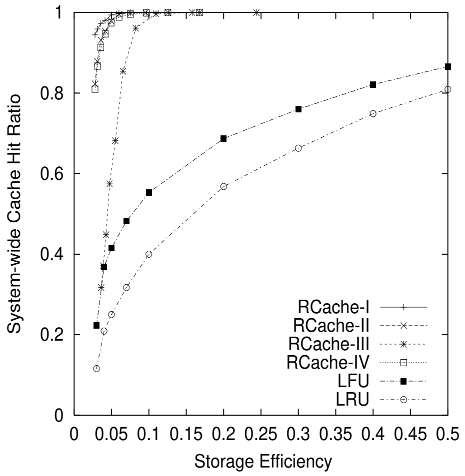
(a) Local Cache Hit Ratio



(b) Remote Cache Hit Ratio



(c) Server Cache Hit Ratio



(d) System-wide Cache Hit Ratio

Figure 6. Cache Hit Ratios with varying Storage Efficiency

to decrease as S_{eff} increases. The decrease after a certain point comes from the increase of the local cache hit ratio while the system-wide cache hit ratio is almost constant as one in that region. Although RCache-IV shows smaller RHR compared to that of RCache-I and RCache-II, it has higher LHR, giving the almost same system-wide cache hit ratio as in Figure 6 (d). RCache-III shows the worst RHR performance while slightly improving LHR compared to RCache-I and RCache-II. Since the RCache-III requires more space to cache the earlier segments, it has less space reserved for the remaining segments, giving the later segments higher probability of not being cached in the entire caching system.

Figure 6 (d) shows the system-wide cache hit ratios of the RCache schemes and the two traditional non-cooperative web-caching systems where SCs independently cache entire clips upon client requests. For the traditional web caching systems, we have implemented two different cache replacement algorithms: Least Recently Used (LRU) and Least Frequently Used (LFU). RCache-I shows the best system-wide cache hit performance. However, it suffers from the high first segment missing ratio as in Figure 7. The first segment missing ratio strongly affects the user-perceived start-up latency. Even in the region of $S_{eff} \leq 0.08$ where the RCache-I shows higher system-wide cache hit ratio than those of all the other schemes, RCache-I shows more than 90% of the first segment missing ratio, requiring high start-up latency to fetch the first segments from remote SCs or the server. RCache-II and RCache-IV perform next best and show the almost identical system-wide cache hit ratio. However, RCache-IV has higher fraction of local cache hits than that of RCache-II, requiring less overhead associated with request redirections and switch-overs.

Compared to traditional web caching systems, RCache schemes provide excellent cache hit ratio performance. At S_{eff} of 3%, RCache-I provides $4-9$ times higher cache hit ratio than those of traditional web caching system with LRU or LFU. As S_{eff} increases, cache hit ratio of RCache systems also grows much faster than those of traditional web cache system, providing almost 100% of cache hit ratio roughly at S_{eff} of 8%. To obtain near 100% of cache hit ratio, traditional web caching system needs $\approx$ 15 times more storage space than RCache systems. One more interesting point is LFU outperforms LRU over all range of S_{eff}.

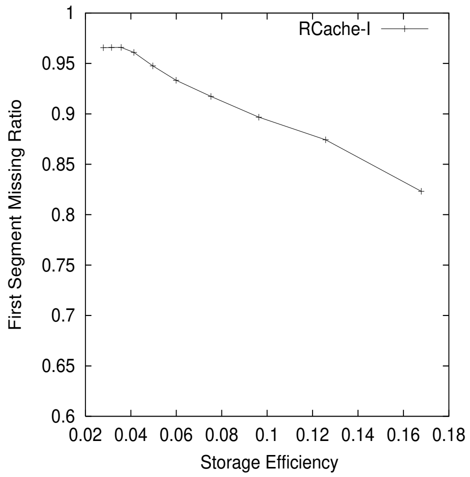
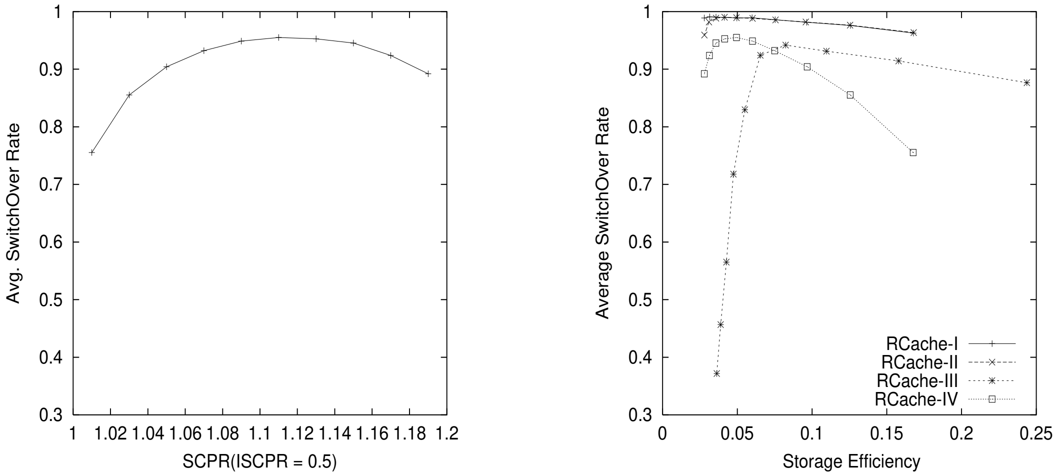


Figure 7. First segment missing ratio of RCache-I

We also studied the impact of SCPR on the switch-over ratio averaged over all requests in the simulation. Figure 8 (a) shows the average switch-over rate with varying SCPR. Clearly, user requests that entirely fall into the first segment do not have the switch-overs. The average switch-over rate only accounts for the user requests that require more than one segment. Figure 8 (a) shows the interesting behavior of this metric. There are fewer switch-overs at both end regions and more in the middle region of SCPR. This can be explained by the behavior of the remote cache hit ratio. The region of the high rate of switch-overs matches with the region of the high remote hit ratio. In both extreme regions, where SCPR is either very small or very large, there are large fraction of cache



(a) Switch-overs of the RCache-IV with varying SCPR (b) Switch-overs with varying S_{eff}

Figure 8. Average Switch-over Rate

hit from the local proxy or the origin server, increasing the chance of consecutive segments being fetched from either the local proxy or the origin server. However, in the region where the remote cache hit ratio is high, there is less chance that consecutive segments are serviced from the same remote proxy, resulting in higher rate of switch-overs. The behavior of switch-overs adversely affects the performance of our RCache system.[5] Figure 8 (b) shows the switch-over performance of RCache schemes with varying S_{eff}. In general, RCache-IV shows the best switch-over performance.[6] The switch-overs can be further categorized into two cases: (1) *To-local* switch-overs, where the next segment is fetched from the local SC, (2) *To-remote* siwtch-overs, where the next segment is fetched from remote SC or the server. Clearly, the overhead associated with *To-remote* switch-overs is much higher than that of *To-local* ones. Since RCache-IV with higher LHR has more chance of the *To-local* switch-overs, it can further reduce the overall overhead associated with the switch-overs. The difference between RCache-III and RCache-II in the region[7] of $S_{eff} \geq 0.1$ shows that the intra-clip replication control improves the switch-over performance. The difference between RCache-IV and RCache-III in the same region also shows the inter-clip replication control further improves the switch-over rate.

In short, RCache-IV with all the three options performs best in terms of cache hit ratio, switch-over rate and storage efficiency. The first segment caching option dramatically improves the start-up latency performance while marginally sacrificing the storage efficiency. The intra-clip replication control improves the local cache hit ratio, but the improvement comes with more space requirement. To achieve 100% system-wide cache hit ratio, RCache-III requires twice as much storage space as that for RCache-I. The inter-clip replication control in RCache-IV dramatically improves the local cache hit ratio and shows the comparable system-wide cache hit ratio with the RCache-I while utilizing almost the same storage space. It also improves the switch-over rate, decreasing the system overhead associated with the switch-overs.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we focused on scalable, fault tolerant data placement and replacement techniques for caching multimedia streams in cooperating distributed caches. Specifically, we proposed following new schemes that work together: (1)*RCache*, randomized, easy to implement distributed data layout schemes that account for the clip popularity and the intra-clip segment popularity. Using various options and parameters associated with this layout, one can control

[5] One approach to solve this problem is segment clustering where more consecutive segments are clustered together.
[6] Although RCache-III shows the smallest switch-over rate in the region of small S_{eff}, this comes from the high SHR in the region.
[7] The region where RCache-III has the same system-wide cache hit ratio with that of RCache-II

its storage efficiency and tradeoff storage for performance. (2) *TwoD*, a two-dimensional *local* replacement scheme based on the concept of segment popularity used for data replacement ordering at each cache.

These schemes together optimize various performance metrics, namely: (1) storage space, (2) start-up latency, (3) overhead from playback switch-overs, and (4) network bandwidth. Our simulation results show excellent performance and provide a means to achieve optimal tradeoff between various design parameters such as a number of caches, origin server bandwidth, and layout parameters. RCache provides 4 - 9 times as high cache hit ratio as that of traditional web caching system while utilizing the same amount of storage space. RCache also provides almost 100 % of cache hit ratio with less than 10 % of storage space required for traditional web cache to achieve the same cache hit ratio.

Our ongoing work is along the following directions: (1) Design and analysis of *Silo* data layouts that control the segment size and also, account for inter-clip and intra-clip popularity. (2) Schemes to reduce or eliminate inter-cache state exchange and instead use mechanisms such as Anycast[21] to acquire state information just-in-time. (3) How to improve the switch-over performance to further reduce the overhead associated with the switch-overs. (4) Investigation of simple proxy selection algorithms and/or segment distribution algorithms to achieve better load-balance. (5) Building a prototype system of RCache in our caching test-bed.

REFERENCES

1. www.lucent.com/businessunit/icdd.html, www.akamai.com, www.cacheflow.com, www.inktomi.com, www.infolibria.com, www.networkappliance.com.
2. http://www.ncube.com.
3. Acharya, S. and Smith, B. "An Experiment to Characterize Videos on the World Wide Web," *Multimedia Computing and Networking 1998.*
4. Acharya, S. and Smith, B., "Middleman: A Video Caching Proxy Sever," *Proc. of NOSSDAV'00*, June, 2000.
5. Acharya, S., Smith, B., and Parnes, P., "Characterizing User Access to Videos on the World Wide Web," *Multimedia Computing and Networking 2000.*
6. Aggarwal, C.,C., Wolf, J., L., and Yu., P., S., "A Permutation-based Pyramid Broadcasting Scheme for Video-on-demand Systems," *Proc. of IEEE International Conference on Multimedia Computing and Systems*, Jun, 1996.
7. Almeroth, K. and Ammar, M., "On the use of multicast delivery to provide a scalable and interactive video-on-demand service", *Journal of Selected Areas of Communication*, Aug, 1996.
8. Bernhardt, C., and Biersack, E., "The Server Array: A Scalable Video Server Architecture," *High-Speed Networks for Multimedia Applications*, editors, Danthine, A., Ferrari, D., Spaniol, O., and Effelsberg, W., Kluwer Academic Press, 1996.
9. Bolosky, W., et al., "The Tiger Video File-server," *Proc. of NOSSDAV96*, Apr, 1996.
10. Bommaiah, E., Guo, K., Hofmann, M., and Paul, S. "Design and Implementation of a Caching System for Streaming Media over the Internet," *Proc. of IEEE RTAS'2000*, May, 2000.
11. Bowman, C., M., Danzig, P., B., Hardy, D., R., et al. "Harvest: A Scalable, Customizable Discovery and Access System," Tech Report, CU-CS-732-94, University of Colorado, Boulder, USA, 1994.
12. L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," *Proc. of IEEE Infocom'99*, Mar, 1999.
13. Buddhikot, M., Parulkar, G., and Cox, J., R., Jr., "Design of a Large Scale Multimedia Storage Server," *Journal of Computer Networks and ISDN Systems, pp. 504-517, December 1994.*
14. Chae, Y., Guo, K., Buddhikot, M., Suri, S., and Zegora, E., "Silo, Rainbow, and Caching Token: Schemes for Scalable, Fault Tolerant Stream Caching," *Bell Labs Technical Memorandum BL1009670-001017-04TM*, Oct, 2000.
15. Chankhunthod, A., Danzig, P., B., Neerdaels, C., Scwartz, M., F., and Worrell, K., J., "A Hierarchical Internet Object Cache," Proc. of the 1996 Usenix Technical Conference, 1996.
16. Chen, P., et al., "RAID: High-performance, Reliable Secondary Storage," *ACM Computing Surveys*, Jun, 1994.
17. Hua, K., Chai, Y. and Sheu, S., "Patching: A Multicast Technique for True Video-on-Demand Services", *Proc. of ACM Multimedia'98*, Sept, 1998.

18. Hua, K., A., Sheu, S., "Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems," *Proc. of ACM SIGCOMM*, Sept, 1997.
19. Juhn, L-S., and Tsend, L-M., "Harmonic Broadcasting for Video-on-demand Service," *IEEE Transaction on Broadcasting*, Vol. 43, No. 3, pp, 268-271, Sept, 1997
20. Juhn, L-S, and Tseng, L-M., "Staircase Data Broadcasting and Receiving Scheme for Hot Video Service", *IEEE Transaction on Consumer Electronics*, Vol, 43, No. 4, pp. 1110-1117, Nov, 1997.
21. Partridge, C., Mendex, T., and Milliken, W., "Host anycasting service," *RFC 1546*, Nov, 1993.
22. Ramesh, S., Rhee, I., and Guo, K., "Multicast with Cache (Mcache): An Adaptive Zero Delay Video-on-Demand Service," *Proc. of IEEE Infocom'01*, Apr, 2001.
23. Sen, S., Rexford, J., and Towsley, D., "Proxy prefix caching for multimedia streams", *Proc. of IEEE Infocom'99*, Mar, 1999.
24. Shenoy, P., "Symphony: An integrated Multimedia File System," *Doctoral Dissertation,* Dept. of Computer Science, University of Texas at Austin, Aug, 1998.
25. Tewari, R., Mukherjee, R., Dias, D., M., and Vin, H., M., "Design and Performance Tradeoffs in Clustered Video Servers," *Proc. of the IEEE ICMCS'96*, May, 1996.
26. UCB/LBNL/VINT, "Network Simulator, ns," http://www.isi.edu/nsnam/ns/.
27. Viswanathan, S., and Imielinski, T., "Metropolitan Area Video-on-demand Service using Pyramid Broadcasting" *ACM Multimedia Systems*, Vol. 4, pp. 197-208, 1996.
28. Wessels, D. and Claffy, K., "Internet Cache Protocol (ICP)", version 2, *IETF RFC 2186*, Sept, 1997.
29. Wolman A., Voelker, G., Sharma N., Cardwell, N., Karlin A., and Levy, M., "On the scale and performance of cooperative Web proxy caching", *Proceedings of ACM SOSP'99*, Dec., 1999.