Silo, Rainbow, and Caching Token: Schemes for Scalable, Fault Tolerant Stream Caching

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

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

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

Ellen W. Zegura
College of Computing,
Georgia Institute of Technology,
ewz@cc.gatech.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 improve performance. 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 independent caches. From the perspective of the clients, the collection of cooperating distributed caches act as a single fault-tolerant, scalable cache [4]. In this paper, we focus on data placement and replacement techniques for such cooperating distributed caches. Specifically, we propose the following new schemes that work together: (1) A family of distributed layouts, consisting of two layouts, namely *RCache* and *Silo*. The *RCache* layout is a simple, randomized, easy-to-implement layout that distributes constant length segments of a clip among caches and provides modest storage efficiency. The *Silo* scheme improves upon *RCache*; it accounts for long term clip popularity and intra-clip segment popularity metrics and provides parameters to tune storage efficiency, server load, and playback switch-overs; (2) *Rainbow*, a local data replacement scheme based on the concept of *segment access potential* that accurately captures the popularity metrics. (3) *Caching Token*, a dynamic *global* data replacement or redistribution scheme that exploits existing data in distributed caches to minimize data distribution overhead.

Our schemes optimize storage space, start-up latency, server load, network bandwidth usage, and overhead from playback switch-overs. Our analytical and simulation results show that the Silo scheme provides 3 - 8 times higher cache hit ratio than a comparable traditional web caching system that has the same amount of storage space.

1 Introduction

In recent years, an exponential increase in the number of Internet users has strained the Internet infrastructure. Scalable, cluster-based web servers and smart 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 worsen.

Extensive research on multimedia streaming services and servers has attempted to devise solutions that scale in terms of number of users and stream bandwidth. This research includes: (1) Scalable, high performance server and file system or operating systems architectures[2, 8, 12, 23]. (2) Network and application level multicast techniques [6, 7, 17, 18, 19, 21, 27] that trade bandwidth and buffer space for interactivity and personalization. (3) Limited caching schemes such as stream patching[16] and prefix caching[22] 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 techniques mentioned above, 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.

Existing web caching systems [1] are typically 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 files 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 space wastage. (2) For a fixed investment in storage space at each cache, only a few media objects can be stored, and therefore, the hit ratio and the efficiency of the cache is limited.

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

An alternate solution 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 [15], and distributed cluster-based video-on-demand servers [2, 8, 9, 12]. 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. In this paper, we focus on design and analysis of such a distributed caching system.

1.1 Research Contributions

In this paper we propose a novel distributed caching architecture and associated schemes that work together. Specifically, we describe the following: (1) A family of distributed layouts, consisting of two layouts, namely *RCache* and *Silo*. The *RCache* layout is a simple, randomized, easy-to-implement layout that distributes constant length segments of a clip among caches and provides modest storage efficiency. The *Silo* scheme improves upon *RCache*; it accounts for long term clip popularity and intra-clip segment popularity metrics and provides parameters to tune storage efficiency, server load, and playback switch-overs; (2) *Rainbow*, a local data replacement scheme based on the concept of *segment access potential* that accurately captures the popularity metrics. (3) A dynamic *global* data replacement or redistribution scheme called *Caching Token* that exploits existing data in distributed caches to reduce distribution overhead. Our schemes optimize storage space, start-up latency, server load, network bandwidth usage, and overhead from playback switch-overs. Our analytical and simulation results show that the Silo scheme provides 3 - 8 times as high cache hit ratio as that of traditional web caching system while utilizing the same amount of storage space.

1.2 Outline

The rest of this paper is organized as follows: Section 2 compares and contrasts relevant work in the areas of cooperative distributed caching and distributed data layouts in VoD servers. Section 3 describes our streaming cache architecture and introduces popularity measures we use in our design. In Section 4, we present design and analysis of our RCache and Silo distributed data layouts and provide guidelines for choosing various associated parameters. In Section 5, we describe cache replacement policies that exploit the special structure inherent in data layouts to optimize storage performance. In Section 6, we discuss clip redistribution mechanisms for clip rank changes, and propose a new *Caching Token* mechanism that reduces network traffic and server load. We also describe the performance results from both analytic and simulation models in Section 7. Finally, Section 8 presents the conclusions and future work.

2 Related Work

In general, two bodies of work are 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 [10, 13] 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 resource-based caching algorithm in [24] is designed for web servers and proxies, and deals with storage of static and dynamic content on a disk system based on their bandwidth, storage requirement, and potential caching gain. In an environment that has single writer/multiple readers, dynamic content are stored entirely or partially by the unit of intervals where there is a single writer and multiple readers. We look at caching for streaming media only, therefore our algorithms are quite different than these.

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 Rainbow data replacement policy based on concept of access potential. We believe our scheme accurately captures popularity of clip segments.

Several research efforts, such as Project MARS [12], Microsoft TIGER file system [9], Server Array [8], and research reported in the literature [15, 23, 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.

Several researchers [6, 17, 18, 19, 21, 27] have proposed broadcast or multicast schemes that transmit variable or constant length segments of media clips on multiple broadcast or multicast channels at fixed or variable rate and offer near-video-on-demand (nVoD) services with limited interactivity. In contrast to a service model that uses a dedicated session per user, such schemes aim to use a fixed amount of network bandwidth and large client buffer to offer limited interactivity. Our layouts use variable length segments common with these schemes.

3 Distributed Caching Architecture

In our distributed cache architecture in Figure 1, 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" block in the web-browser) or may be established transparently using a layer-4 switch or 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 in a large cluster cache interconnected by a

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

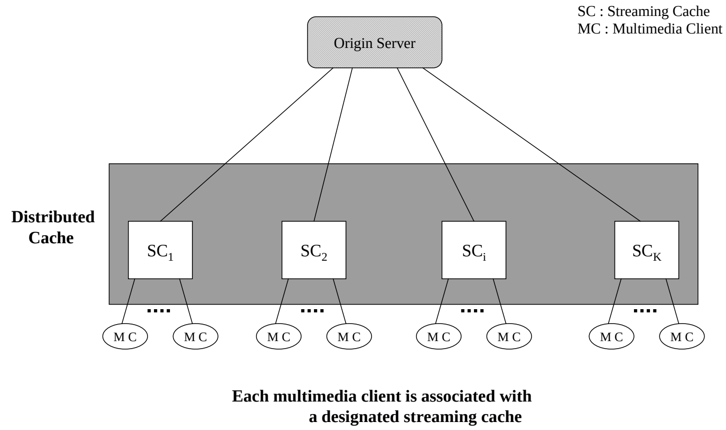


Figure 1: Distributed streaming cache architecture

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 among themselves using a inter-cache protocol, for example a variant of ICP[28].

In our architecture, each media clip is divided into relatively small segments that are distributed among different proxies in the network. The design of our layouts combined with demand-driven caching, guarantees that each segment is stored in *at least* one cache. The state exchange among proxies guarantees that each proxy knows the clips present in the cluster and has a *global segment map* that describes which proxies store segments of every clip.

The mechanics of creation of data layout and cache behavior are described next.

Cold Start for clip M ***(Data Layout Creation):*** Suppose one of the proxies SC_i receives a request for a clip M which is not currently present in the cache cluster. 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* or *Silo* layout (Section 4.2) to compute a *local segment map* 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, 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 *local segment map* 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, if the local segment map proxy at SC_i says it is should not store the particular segment, it consults the global segment map and redirects the client 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 the client. 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 the client. 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 clip M that is already present in the cluster, for each segment j of the 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 ($LocalSegmentMap[M, j] = 1$), but it has not been cached yet or has been flushed out due to local replacement: In this case, SC_i consults our *Rainbow* local 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 be cached locally ($LocalSegmentMap[M, j] = 0$), 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 clip cache miss happens either when the clip is requested for the first time, or 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. Our *Caching Token* scheme guarantees that this is accomplished just-in-time with minimal redistribution overhead and origin server load. Periodically global cache map and local cache map are updated deleting information about clips that are no longer cached anywhere in the proxy cluster.

When responding to client requests, the cluster of proxies together act as a single, loosely-coupled streaming cache. 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 switching 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. Also, our Silo data layouts reduce the number of switch-overs by appropriate design of segment lengths.

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

The popularity of clips and segments plays a critical role in our algorithms. 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 Hotness Rating (LSHR)*, a micro-metric that captures the popularity of segments of the clips.

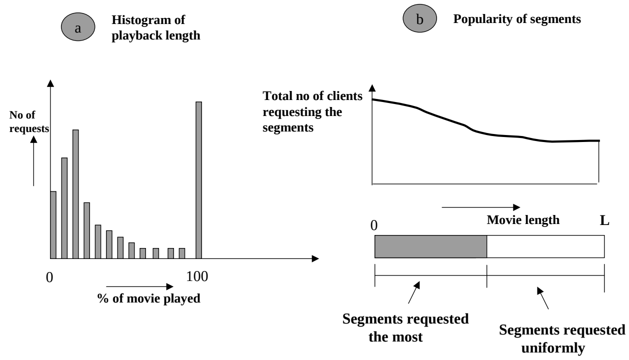


Figure 2: Bimodal distribution of clip segment requests

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. 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 meager and often the Zipf distribution is used to model them [11, 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] $0.8 \leq \gamma \leq 1.2$.

With respect to segments within a clip, the observed user behavior follows a bimodal distribution in actual playback length [5]. As shown in Figure 2 (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, different parts of the clip are requested with different probability; segments in the beginning of clips are requested more often than later segments as in Figure 2 (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 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.

4 Data Layouts in Distributed Caches

In this section we first describe a simple random data layout scheme called *RCache*, and motivate the need for better layouts. We then present the details of our *Silo* 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 2 (b) can be regarded as a complementary cumulative distribution function of the playback time distribution in Figure 2 (a).

4.1 RCache: Random Caching of Media File Segments

The basic *RCache* 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."

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

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.

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

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

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

Therefore, Equation 2 can be rewritten as

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

Since $(1-\frac{1}{K})^K \to e^{-1}$, as $K \to \infty$, 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. *RCache* 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 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 basic RCache 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, 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 we can guarantee that the first segment of the clip is always available on the designated cache. This requires that the first segment be stored in every SC in the cluster.

Given that the segments in the early part of the clip are requested more frequently than those in the later part, it makes sense to distribute later segments to fewer SCs. Following this hypothesis, a layout that employs segments of increasing duration will decrease the average number of switch-overs for each user. In other words, instead of making each segment the same length and the probability to cache each segment the same, we can have a variety of RCache schemes, each with different segment lengths and different probability to cache each segment. We have two degrees of freedom in the design of RCache algorithm: one is how to divide a clip into segments, another is how to store the segments. In the following section, we discuss the *Silo* scheme for data layout, named after the shapes of the stacked-bar graphs of segment size versus segment number.

4.2 Silo Data Layouts

We first describe the *Silo* data layout scheme for a single clip and then describe the most general layout for multiple clips. We also provide guidelines for how relevant parameters for these data layouts should be selected.

4.2.1 Basic Silo Data Layout For Single Clip

Consider a clip with length L, divided into N segments. The length of each segment and the probability with which an SC decides to cache a segment are controlled by two paramaters: an increasing ratio of segment size (SSR) α and a decreasing ratio of intra-clip segment caching probability (SCPR) β. For the Silo layout, the size of the first segment is set to a small fraction of the entire clip, δ, and each cache stores it with probability 1. The size of the second segment is $\alpha\delta$, and the probability to store the segment drops to $1/\beta$. In general, up to the m^{th} segment, the probability with which segment i of length $\alpha^{i-1}\delta$ is stored at each cache is $1/\beta^{i-1}$.

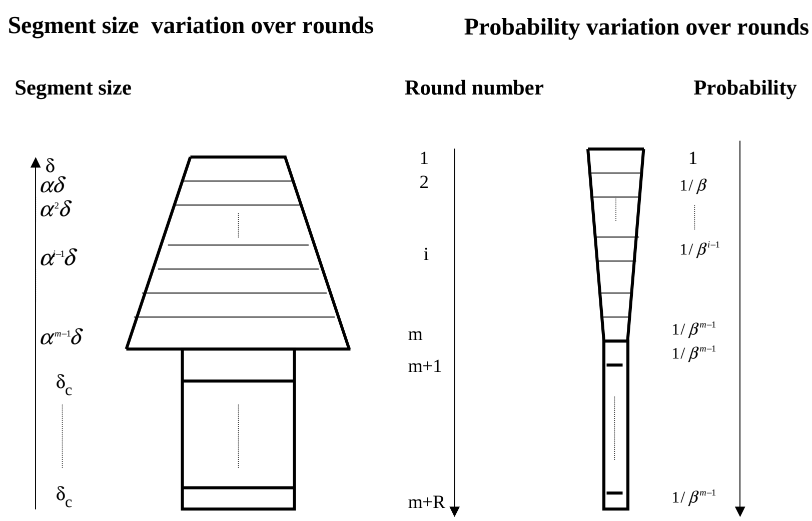


Figure 3: Generalized Silo layout: segment size and probability variation

At the m^{th} segment, the segment size reaches its maximum value of $\alpha^{m-1}\delta$. From that point on, the segment size is held constant at δ_c and the caching probability is fixed at $1/\beta^{m-1}$. We can either set (1) the constant segment size δ_c of the $(m+1)$-th and all subsequent segments to the size of the m^{th} segment or (2) reduce it to smaller values as in Figure 3. The choice depends on the trade-off between switch-overs and flexibility of storage management in cache replacement mechanism. As the segment size becomes large, the number of switch-overs is reduced at the expense of flexibility of cache replacement.

Table 1 illustrates the segment size and caching probability variation. The main principles used in this design are: (1) *Minimize playback startup latency:* To this end, the layout always caches the first segment of the clip at each cache. (2) *Minimize playback switch-over for most requests:* To this end, the layout uses increasingly longer segments for the portions of the clip that are most requested. The Silo shape matches the bimodal request distribution for segments. (3) *Achieve better load balance and fault-tolerance*: Unlike in Middleman [4] proxy architecture that stores only one copy of constant length segments, Silo layout stores increasingly large number of 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 basic silo 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	$\alpha\delta$	$1/\beta$	K/β	$\frac{\alpha}{\beta}\delta K$
3	$\alpha^2\delta$	$1/\beta^2$	K/β^2	$(\frac{\alpha}{\beta})^2\delta K$
...				
m	$\alpha^{m-1}\delta$	$1/\beta^{m-1}$	K/β^{m-1}	$(\frac{\alpha}{\beta})^{m-1}\delta K$
$m+1$	δ_c	$1/\beta^{m-1}$	K/β^{m-1}	$\delta_c\frac{K}{\beta^{m-1}}$
$m+2$	δ_c	$1/\beta^{m-1}$	K/β^{m-1}	$\delta_c\frac{K}{\beta^{m-1}}$
...				
...				
$m+R$	δ_c	$1/\beta^{m-1}$	K/β^{m-1}	$\delta_c\frac{K}{\beta^{m-1}}$

Note that the sum of the size of first m segments is

$$\delta + \alpha\delta + \alpha^2\delta + \ldots + \alpha^{m-1}\delta = \frac{\delta(\alpha^m - 1)}{\alpha - 1} \tag{6}$$

After round m, all the segments are of the same size δ_C . Therefore, for a clip of duration L, the number of segments needed for the bottom half of the generalized Silo shape is

$$R = \frac{(L - \frac{\delta(\alpha^m-1)}{\alpha-1})}{\delta_c} \tag{7}$$

¿From the last column in the Table 1, we can see that the total storage requirement for the basic Silo scheme is

$$T = K\left(R\frac{\delta_c}{\beta^{m-1}} + \delta\sum_{i=0}^{m-1}(\frac{\alpha}{\beta})^i\right) \tag{8}$$

The storage efficiency factor is

$$\begin{aligned} S_{eff} &= \frac{T}{LK} \\ &= \frac{\left(R\frac{\delta_c}{\beta^{m-1}} + \delta\sum_{i=0}^{m-1}(\frac{\alpha}{\beta})^i\right)}{L} \end{aligned} \tag{9}$$

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

4.2.2 Silo Data Layout for Multiple Clips

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, 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 Silo layout for a single clip 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.

Parameters: (1) $\boldsymbol{\delta}$: Size of the first segment. (2) $\boldsymbol{\delta_C}$ **:** Size of the constant segments. (3) $\boldsymbol{\alpha}$ **:** Increase Segment Size Ratio (SSR). (4)$\boldsymbol{\beta}$ **:** Decreasing Intra-clip Segment Caching Probability Ratio (SCPR). (5) $\boldsymbol{\gamma}$: 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	$\alpha\delta$	$\frac{1}{\beta i^{\gamma}}$	$\frac{K}{\beta i^{\gamma}}$	$\alpha\delta\frac{K}{\beta i^{\gamma}}$
3	$\alpha^2\delta$	$\frac{1}{\beta^2 i^{\gamma}}$	$\frac{K}{\beta^2 i^{\gamma}}$	$\alpha^2\delta\frac{K}{\beta^2 i^{\gamma}}$
$\vdots$				
m	$\alpha^{m-1}\delta$	$\frac{1}{\beta^{m-1} i^{\gamma}}$	$\frac{K}{\beta^{m-1} i^{\gamma}}$	$\alpha^{m-1}\delta\frac{K}{\beta^{m-1} i^{\gamma}}$
$m+1$	δ_c	$\frac{1}{\beta^{m-1} i^{\gamma}}$	$\frac{K}{\beta^{m-1} i^{\gamma}}$	$\delta_c\frac{K}{\beta^{m-1} i^{\gamma}}$
$\vdots$			$\vdots$	
$m+R$	δ_c	$\frac{1}{\beta^{m-1} i^{\gamma}}$	$\frac{K}{\beta^{m-1} i^{\gamma}}$	$\delta_c\frac{K}{\beta^{m-1} i^{\gamma}}$

Table 2: General Silo cache layout for rank i clip

Table 2 shows the details of the configuration parameters and the cache layout for a clip of rank i. In this general layout, during the exponential increase phase, the segment size is increased by a factor α for successive segments until the maximum $\alpha^{m-1}\delta$ is reached. After this, the remaining segments have the same constant size of δ_C.

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

$$\begin{aligned} S_i &= \delta + (\delta \frac{\alpha}{\beta i^\gamma} + \delta \frac{\alpha^2}{\beta^2 i^\gamma} + \ldots + \delta \frac{\alpha^{m-1}}{\beta^{m-1} i^\gamma}) + R \frac{\delta_C}{\beta^{m-1} i^\gamma} \\ &= \delta + \frac{(L - \frac{\delta(\alpha^m - 1)}{\alpha - 1})}{\beta^{m-1} i^\gamma} + \frac{\delta}{i^\gamma} \sum_{k=1}^{m-1} (\frac{\alpha}{\beta})^k \end{aligned} \tag{10}$$

Therefore, the total size of the Silo cache with K proxies and M clips is

$$T = \sum_{i=1}^{M} K S_i \tag{11}$$

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 M clips is computed as $S_{eff} = \sum_{i=1}^{M} \frac{S^i_{eff}}{M}$.

Note that the Middleman [4] proxy architecture stores only one copy of each segment and therefore always requires only LN amount of storage. Clearly, Silo layout requires $\sum_{i=1}^{N} KS_i/LN = KS_{eff}$ times larger storage than Middleman.

4.3 Analytical model

In this section, we introduce a simple analytical model of the static Silo configuration, providing a valuable means to understand the system dynamics and also providing basis for our *local Rainbow* replacement scheme and proxy load balancing schemes. The analytical model provides three important performance metrics: (1)Local Cache Byte Hit Ratio (LHR), Φ_l, representing the percentage of stream data that is retrieved from local Silo caches. (2)Remote Cache Byte Hit Ratio (RHR), Φ_r, representing the percentage of stream data that is retrieved from remote Silo caches. (3)Server Byte Hit Ratio (SHR), Φ_s, representing the percentage of stream data that is retrieved from the origin server. For the sake of brevity, we will refer to Byte Hit Ratio as Hit Ratio from here on.

Assume that the number of Silo proxies is K and the number of movie clips is M. The Silo cache layout is shown in Table 2. From Table 2, the segment caching probability of the segment j of a clip with rank i, $P_{i,j}$, is

$$P_{i,j} = \begin{cases} 1 & \text{if } j = 1 \\ \frac{1}{\beta^{j-1} i^\gamma} & 2 \le j \le m \\ \frac{1}{\beta^{m-1} i^\gamma} & m < j \le m + R \end{cases}$$

Then the probabilities that a segment can be accessed from local cache, remote caches, and the origin server, $P^l_{i,j}$, $P^r_{i,j}$, and $P^s_{i,j}$, respectively, are

$$P^l_{i,j} = P_{i,j},\ P^s_{i,j} = (1 - P_{i,j})^K,$$
$$P^r_{i,j} = (1 - P_{i,j})(1 - (1 - P_{i,j})^{K-1})$$

Also, let ρ_i be the probability that the movie clip with rank i is selected, then ρ_i is

$$\rho_i = \frac{\frac{1}{i^\gamma}}{\sum_{k=1}^{M} \frac{1}{k^\gamma}}$$

Given that a clip is selected, the probability that a segment j is selected, ψ_j, is

$$\begin{aligned}
\psi_1 &= Prob\{Playback\ time \geq Segment\ 1^{st}\ start\ time\} \ = \ 1 \\
\psi_j &= Prob\{Playback\ time \geq Segment\ j^{th}\ start\ time\} \\
&= Prob\{Playback\ time \geq \sum_{k=1}^{j-1} \tau_k\},\ 2 \leq j \leq m+R
\end{aligned}$$

where τ_j is the length of the segment j. The probabilities ψ_j can be calculated from either LSHR or the bimodal distribution [5].

The average number of bytes retrieved from local cache, remote caches, and origin server, θ_l, θ_r, and θ_s, respectively, are

$$\theta_l = \sum_{k=1}^{M} \rho_k (\sum_{j=1}^{m+R} \eta_j \psi_j P_{k,j}^l),\ \theta_r = \sum_{k=1}^{M} \rho_k (\sum_{j=1}^{m+R} \eta_j \psi_j P_{k,j}^r)$$
$$\theta_s = \sum_{k=1}^{M} \rho_k (\sum_{j=1}^{m+R} \eta_j \psi_j P_{k,j}^l)$$

Finally, byte hit ratios are

$$\Phi_l = \frac{\theta_l}{\theta_l + \theta_r + \theta_s}, \Phi_r = \frac{\theta_r}{\theta_l + \theta_r + \theta_s}, \Phi_s = \frac{\theta_s}{\theta_l + \theta_r + \theta_s}$$

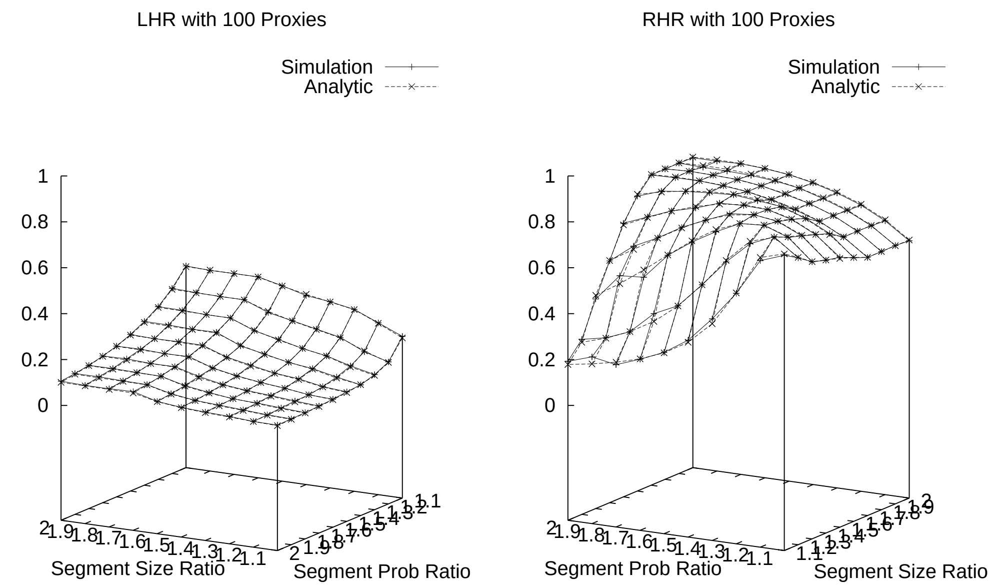
The Figures 4 shows the results from both an ns-2 simulation and our analytical model for a Silo system with 100 clips of 3 GB each stored on a cluster of 100 SCs interconnected by a mesh topology. Figure 4.3 shows the local hit ratio, fraction of data served from the designated local SCs, with varying segment size ratio (α) and segment probability ratio (β). Figure 4.3 shows the remote hit ratio, fraction of data served from the sibling remote SCs, with varying α and β parameters. It is clear that the results from our analytical model matches very well with simulation, validating the accuracy of the model.

4.4 Guidelines for Choosing Parameters

Given the parameters L, K, S_{eff} the parameters that need to be decided before the Silo layout for the clip can be realized are δ, α, β, m, and γ. The smallest 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 initial segment size of 1 to 2 minutes is sufficient. With an MPEG-2 stream of 5 Mbps average bandwidth, this results in the first segment size of roughly $\delta = 35 - 75$ MB. For α and m, the size of the Silo head is supposed to be large enough to cover the short beginning period that gets the maximum number of requests. Given the first ρ fraction of the clip playback period gets most accesses, the size of Silo head structure from Equation(6) should satisfy the following condition:

$$\frac{\delta(\alpha^m - 1)}{\alpha - 1} \geq \rho L \tag{12}$$

where ρ can be obtained from either LSHR or the observed bimodal distribution of user playback time [5]. Equation(12) provides a set of candidate (α, m) tuples. Note also that a set of candidate



(a) Local Hit Ratio (LHR) (b) Remote Hit Ratio (RHR)

Figure 4: Local and remote hit ratios with 100 Silo proxies ($\delta = 50$ MB)

(β, m) tuples can be calculated from either the bimodal distribution of playback time [5] or the LSHR. The final (α, β, m) can be determined from the both candidate tuples of (α, m) and (β, m). Figure 4 shows that there could be multiple (α, β, m) triples that provide the similar cache hit ratio performance.

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. As γ becomes larger, S_{eff} becomes better while decreasing the cache hit ratio for less popular clips.

5 Local Cache Replacement: The Rainbow Algorithm

Each component streaming cache or node 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 data segments to create space for new segments. The purpose of a cache replacement algorithm is to retain segments that maximize the hit ratio. Clearly, such a replacement algorithm must account for the Global Clip Hotness Rating (GCHR) and the Segment Hotness Rating (SHR) metrics defined in Section 3.

The Silo data layouts have 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 be transformed into a one dimensional ordering based on a single quantity that captures both popularity metrics. To this end, we propose a novel cache replacement algorithm, *Rainbow*, that uses a single metric called *caching potential* to order segments.

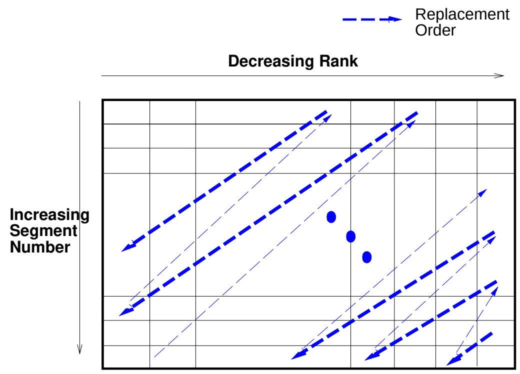


Figure 5: Cache replacement order of Rainbow algorithm

The two main ideas in Rainbow are: (1) Assignment of caching potential to each segment. (2) Quantization of caching potential into "bands" of colors (potentials) to minimize the overhead of sorting the segments.

The cache potential function $F(i,j)$ is proportional to the probability that a segment j of a clip with rank i is accessed when a user request for clip playback arrives. The higher the probability is, the higher the potential that the segment is requested for playback. Clearly, $F(i,j)$ is based both on clip popularity distribution and playback time distribution and successfully captures both inter-clip and intra-clip orderings of importance. The potential function $F(i,j)$ can be formally written as

$$
\begin{aligned}
F(i,j) &= Prob\{ClipRank = i\} \times Prob\{SegmentPlayed = j\} \\
&= \frac{\frac{1}{i^{\gamma}}}{\sum_{k=1}^{N} \frac{1}{k^{\gamma}}} \times Prob\{PlaybackTime \geq StartTime(j)\} \\
&= \rho_i \times \psi_j
\end{aligned}
$$

Using the analytical model in Section 4.3, we can compute ρ_i, ψ_j, and therefore, $F(i,j)$. Using this potential function, the cache replacement algorithm can sort all the segments in its store in decreasing order of the potential and replace segments from the end of the sorted list whenever it needs more space. Figure 5 shows the new diagonal ordering of the Rainbow cache replacement algorithm. Segments in the lower right corner have lowest potential and segments in the upper left corner have highest potential.

Clearly, with a large number of clips, the number of segments in the cache can be very large and the overhead incurred to sort caching potentials every time ranks change or new clips are introduced can be prohibitively large. However, note that the exact ordering of the segments is not strictly required. Instead we adopt a quantization mechanism that is based on coarse grain sorting. Instead of using the exact values of the potential function and the exact ordering, the entire range of the values of the potential function is divided into subranges, each of which is assigned a color code, hence the name "Rainbow". The quantization can be either linear or non-linear. Non-linear quantization can give more flexibility by dividing lower potential region with finer granularity and higher potential region with more coarse granularity. On the other hand, a linear quantization gives simplicity. The segments are stored into bins with matching color and the segment ordering within a bin is not maintained. When adequate space cannot be found to cache a new segment with an assigned potential and color band, the Rainbow algorithm attempts to replace the segments from the lowest color bin until it finds enough space for the new segment.

The Rainbow algorithm gives a good one dimensional ordering of replacement, combining both dimensions of replacement ordering of Silo cache in a novel manner. Furthermore, it is fully scalable in a sense that it can seamlessly work under various configurations of Silo cache layout.

6 Global Replacement: Dynamic Clip Redistribution

In this section, we present an efficient *caching tokens* scheme that implements global data redistribution required to reflect changes in clip popularity ranks.

Recall that in our caching architecture, the rank of a clip plays an important role since it affects segment caching probabilities and decides the number of cached copies of segments. As the rank of the clip changes over time with changing client preferences, data layout must be changed accordingly to account for the new rank information. If the rank of a clip goes up (down), the number of cached segments of the clip should be increased (decreased).

A naive way to deal with the rank change is to redistribute the entire clip and rebuild a new cache layout. However, this approach has obvious drawbacks. Complete redistribution generates considerable load on the network and the origin server. During the data redistribution phase, each proxy builds a new cache layout from scratch. Until this process completes, the Silo cache maps are inconsistent and a lot of client requests for the clip will be served from the origin server. Also, this process can trigger increased synchronized ICP message exchanges among proxies. In the following we present an efficient solution to these problems.

6.1 Lazy Caching and Caching Token

We introduce two mechanisms – *lazy caching* and *caching token* – to deal with the aforementioned problems associated with the simple redistribution method. The caching token mechanism is based on the concept of moving the "pointer to the data location" rather than the data itself. Specifically, each cache maintains two bits of state information for each segment: (1) *Segment presence bit (C):* indicates if the segment is currently present in the local storage. (2) *Token bit (T):* indicates if the cache is "legally" holding the segment. If this bit is set, the segment is supposed to be cached in the new cache layout.

When a Silo proxy receives new rank information from the server, it recomputes the segment caching probabilities of the clip and determines a new cache layout. However, the new layout takes effect lazily, on an as-needed basis when there is a client request for the segment. The changes are realized using a caching token bit exchange on a segment-by-segment basis instead of the entire clip. Thus, exchange of a few bits of information per segment, instead of segments themselves, reduces the network traffic and load on Silo proxies. The goal of the token bit exchange is to maintain the

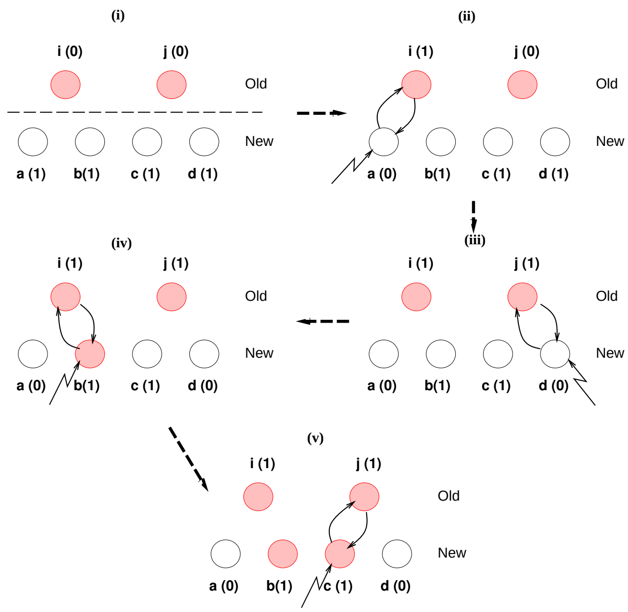


Figure 6: Silo cache layout change: rank up

correct number of copies of the segment, while moving data as little as possible.

When a client request is received for a segment for which rank information and therefore layout has changed recently, two cases can arise: (1) no proxy has cached the requested segment, and (2) at least one proxy has cached the segment. For the first case, the proxy reroutes the client request to the origin server and also obtains a copy of the segment. For the second case, the caching token exchange mechanism attempts to utilize currently cached copies of the segment. Note that recomputing the data layout results in two sets of proxies: (1) an old set of proxies which currently cache the segment, and (2) the new set of proxies which are supposed to cache the segment. Any proxy belonging to both sets does not need to change its cache layout. Only the remaining proxies are involved in token exchanges to build the new cache layout.

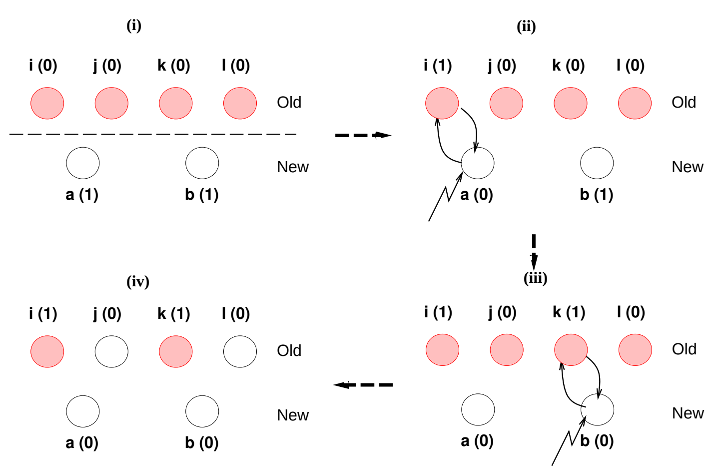


Figure 7: Silo cache layout change: rank down

We illustrate these concepts with two example scenarios when clip rank is altered. Figures 6 (i)-(v) illustrate the token exchange dynamics for one segment of a clip, when the clip rank goes up. In these figures, each circle corresponds to a proxy and the number $x(y)$ represents value of the token bit (y) at node x for the segment under consideration. In the old layout (Figure 6 (i)), before the rank change, the two proxies i, j shown by shaded circles store the segment. After the rank change, a new set of Silo proxies, a, b, c and d, have determined they should have a copy of the segment in their local caches and will set the token bit for this segment to 1 indicating their right to hold the segment. The proxies i, j on the other hand conclude that they should not keep this segment and therefore set the token bit to zero indicating they are illegally holding the segment. However, change of layout does not begin until there is a client request for the segment. As in Figure 6 (ii), when proxy a receives a request for the segment, its local state says that it is supposed to have this segment, but it does not have it in the storage. Also, it knows from the previous layout information that i has this segment. So instead of fetching the segment, proxy a decides to handover the "legal rights" for this segment to proxy i. To achieve this, it sends i a segment request with the token bit set, indicating a token exchange (See Figure 6 (ii)).

Upon arrival of a segment request with the token bit set, a proxy performs two checks: First, if the segment cached bit is not set, indicating the segment is not in the cache and therefore an error is returned to the cache requesting token exchange. Otherwise, it checks the token bit of the segment; if it is clear, the token is grabbed, else an error is returned. In our example, proxy i has has the segment with a clear token bit, so it grabs the token and sets the token bit of the segment. By doing this, proxy i gains the right to retain the segment in its cache. It also sends a segment reply message with token bit clear to proxy a, which then concludes that the token exchange is successful and clears its token bit for the segment. Notice the segment reply message does not contain the actual data segment, proxy a redirects the client to proxy i which streams the request segment to the client. This terminates a successful cycle of token exchange. In Figure 6 (iii), proxies d and j have exchanged the token successfully. In Figure 6 (iv), proxy b tries to exchange its token. However, the token exchange fails since proxy i already has the token for the segment. Thus, proxy i sends the segment with token bit set to proxy b. Proxy b learns the token exchange has failed and caches the segment with token bit set to 1. The same thing happens to proxy c in Figure 6 (v). In the end, proxies i, j, b and c cache the segment instead of the new set of proxies, a, b, c, and d. By exchanging tokens, proxies i and j can keep the segment in their caches, eliminating the movement of the segment to proxies a and d.

Next, consider the case when clip rank is decreased and so the number of proxies that cache the segment need to be reduced. In Figure 7 (i), currently, proxies i, j, k and l have the segment in their local caches. With the rank change, a new set of proxies, a and b, have determined to cache the segment. In Figure 7 (ii), when proxy a receives a request for the segment for which it holds a token but does not have the segment, it attempts a token exchange with proxy i which will succeed. Similarly, proxy b and k have exchanged a token successfully in Figure 7 (iii). In Figure 7 (iv), the local replacement scheme lazily removes the segments at proxies j and l when additional space is needed. In this example too, no segments are moved for building up the new cache layout, thus reducing network and system load.

Appendix A shows the pseudo code for handling rank change request, the initiation and termination of lazy caching and token exchange, and how to handle token exchange request.

Clearly, compared to the simple redistribution mechanism, our new mechanism has several advantages: (1) The origin server is relieved from the redistribution of an entire clip and participates in reconstruction only if there is no proxy that caches a segment. (2) The token bit exchange method minimizes the required segment copying to build up a new cache layout. (3) The on-demand nature of token exchange eliminates synchronized ICP message exchange and reduces instances of inconsistent cache map.

Note if there are N segments in a clip, in the worst case N bits of information need to be exchanged for token exchanges per clip. Clearly, a single maximum size IP packet can exchange token bits for a large number segments of multiple clips and thus, the caching tokens require negligible overhead.

7 Performance Analysis

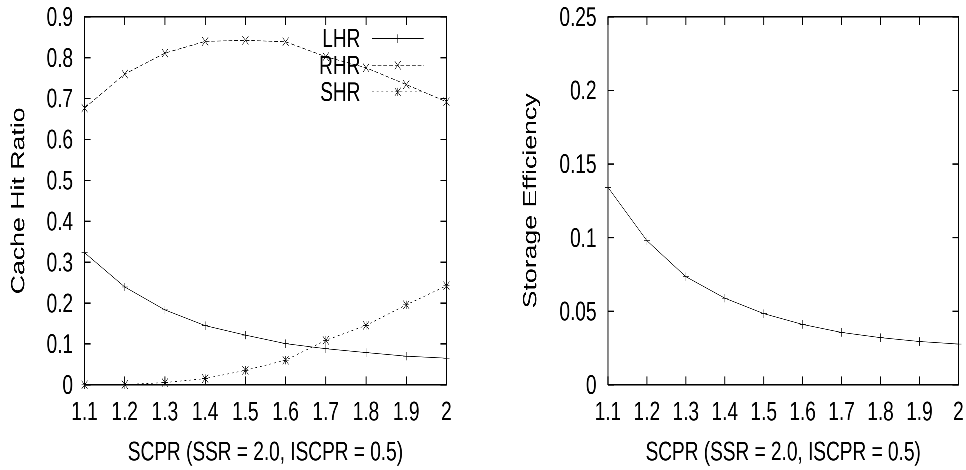
To analyze performance, we use the ns-2 [26] simulation package to model the Silo distributed cache system. In case of static configuration of Silo, we assume that the entire cache layout is built at start-up and remains the same through the end of simulation. We believe that the static configuration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

Recall that important parameters that control the Silo caching system: (1) Segment Caching Probability Ratio (SCPR) β, (2) Segment Size Ratio (SSR) α, (3) Inter-Clip Segment Caching Probability Ratio (ISCPR) γ. In this section, we discuss how the control parameters affect the performance of Silo caching system. We measured the performance metrics defined in Section 4.3, namely: (1) Local hit ratio (LHR), (2) Remote hit ratio (RHR), (3) Server hit ratio (SHR), and (4) Number of switch-overs. The start-up latency is not an issue for Silo since SCs always cache the first segment in their local cache. Table 3 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.

Table 3: Default Values of Simulation Parameters

Parameter	Default Value
Initial Segment Size	50 MB
Segment size of Silo body	50 MB
Clip size	3 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	0.8 - 4.5 GB depending on parameters

Varying Segment Caching Probability Ratio (SCPR)



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

Figure 8: Cache Hit Ratios and Storage Efficiency with varying segment caching probability ratio

First, we consider the performance metrics with varying SCPR. Figure 8(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.

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 increases. However, for the Silo cache system, 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 Silo cache system can serve client requests either through local designated SC or remote SCs, keeping the server hit ratio (SHR) almost zero. For example, in Figure 8, with less then 8% of S_{eff} at SCPR of 1.3, the Silo caching system 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. The ratio between local and remote hit ratios is strongly related to S_{eff}. To serve more requests from the local SC, more segments should be cached locally, increasing S_{eff} as in Figure 8(b). As SCPR increases beyond the threshold value, however, the chance that the segments are being cached within the Silo cache system decreases, resulting in increase of the server hit ratio.

In Figure 9, we compare the performance of Silo to that of traditional web caching system, where SCs independently cache entire clips upon client requests. For the traditional web caching system, we have implemented two different cache replacement algorithms: Least Recently Used (LRU) and Least Frequently Used (LFU). Compared to traditional web caching systems, Silo provides excellent cache hit ratio performance. At S_{eff} of 3%, Silo provides $3-8$ times higher cache hit ratio than those of traditional web caching system with LRU or LFU. As S_{eff} increases, cache hit ratio of Silo also grows much faster than those of traditional web cache system, providing almost 100% of

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

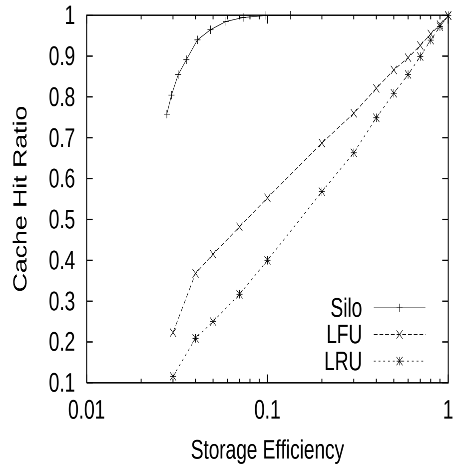


Figure 9: Silo vs. Traditional Web Cache with varying cache size

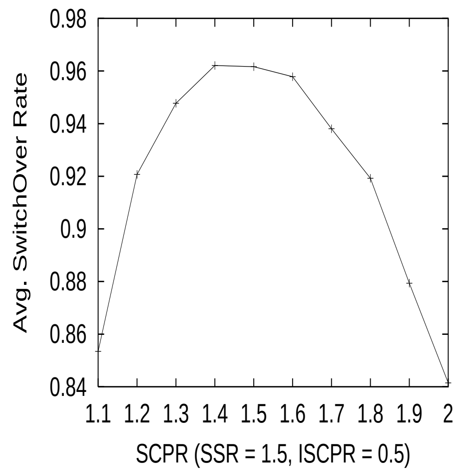
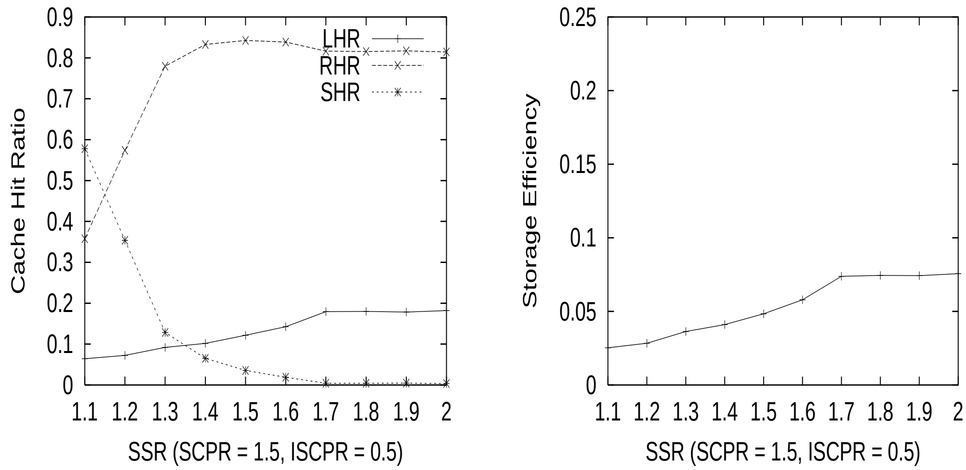


Figure 10: Switch-overs with varying SCPR

cache hit ratio roughly at S_{eff} of 7%. To obtain near 100% of cache hit ratio, traditional web caching system needs ≈ 15 times more storage space than Silo. One more interesting point is LFU outperforms LRU over all range of S_{eff}.

We also studied the impact of SCPR on the number of switch-overs averaged over all requests in the simulation. Figure 10 shows the average switch-over rate with varying SCPR. The average switch-over rate is the ratio of the total number of switch-overs to the total number of segment boundaries. 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 10 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 (RHR). 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 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 Silo cache system.[5]

In short, SCPR controls the intra-clip segment replication by controlling the decreasing ratio of the segment caching probability within a clip. In the following section, we discuss Segment Size Ratio (SSR) that also controls the intra-clip segment replication.



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

Figure 11: Cache Hit Ratios and Storage Efficiency with varying SSR

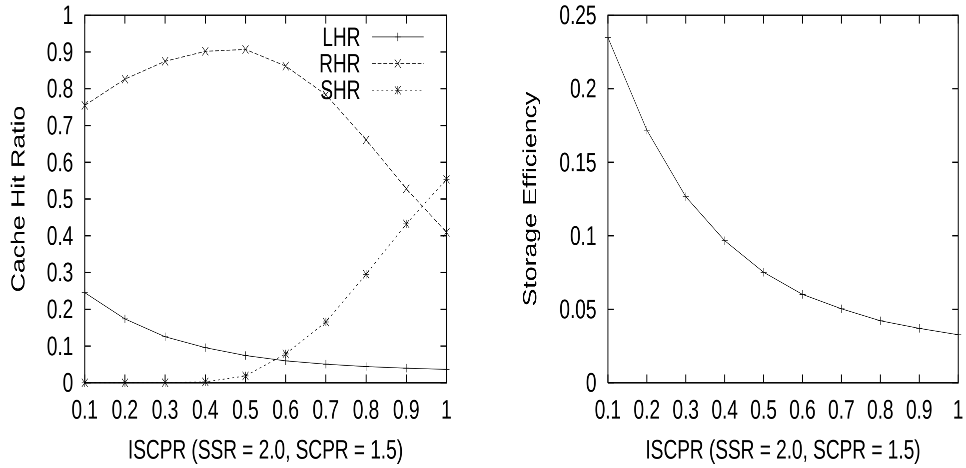
Varying Segment Size Ratio (SSR)

Figure 11 indicates that varying SSR has the opposite effect of varying SCPR on the performance metrics. This is because, as SSR increases, the number of segments in the portion of the Silo layout with varying segment sizes (the *"Silo roof"*) decreases. This results in smaller number of steps in decreasing segment caching probability and higher segment caching probability for the segments in the Silo body. By controlling the number of decreasing steps in segment caching probability, SSR achieves the intra-clip segment replication control. All the performance metrics show a plateau in the region of large values of SSR (Figure 11). In this region, the number of segments in the Silo roof does not change due to the fixed total size of the Silo roof, although SSR changes.

Varying Inter-Clip Segment Caching Probability Ratio (ISCPR)

Although varying ISCPR shows the similar performance results with the SSR and SCPR cases (Figure 12), the reason for this performance change is quite different from SSR and SCPR. The ISCPR controls inter-clip segment replication while SSR and SCPR control intra-clip segment replication. By varying ISCPR, the Silo system can control the preference to clips with different ranks. If the ISCPR is high, more storage space is allocated to popular clips. Since the inter-clip

[5] One approach to solve this problem is segment clustering where more consecutive segments are clustered together.



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

Figure 12: Cache Hit Ratios and Storage Efficiency with varying ISCPR

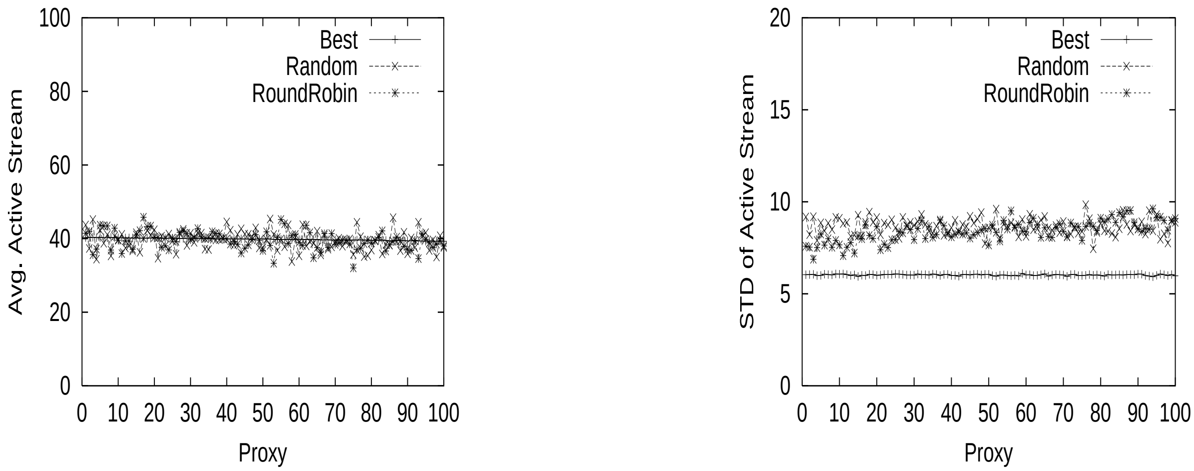
segment replication control through varying ISCPR is enforced clip-wide, the performance metrics are more sensitive to ISCPR than SSR and SCPR as in Figures 8, 11 and 12.

The three control parameters can be used in combination to configure proper operational region of the Silo system, where the server hit ratio can be kept very small and the ratio between the local and the remote hit ratios can be controlled with varying storage efficiency.

Figure 13 shows the average and standard deviation of the number of active streams per proxy. As discussed in Section 4.2, Silo replicates segments according to their popularity to achieve better load-balancing performance. When there are more than one copies of a segment in Silo, selection of one of the proxies should be made. We implemented three remote proxy selection methods: *Best*, *Random*, and *Round-Robin*. The *Best* method selects the proxy that currently has the least load among the candidate proxies. We use the number of current active streams as the indicator for load. We assume instant update of proxy state for the *Best* method for comparison purpose. The *Random* method randomly selects a target remote proxy from the candidate proxies. The *Round-Robin* method selects a target proxy from the candidate proxies in a round-robin manner. The *Best* method shows the almost perfect load-balancing performance as in Figure 13 (a). However, the *Best* method requires prompt update of proxy states and its performance strongly depends on the freshness of the state information. We use the *Best* method as an "ideal" case for comparison. The Random and the Round-Robin methods show comparable load-balancing performance considering they do not use any state information. Figure 13 (b) shows the standard deviation of the number of active streams on each proxy. The *Best* method shows the least variance of the number of active streams – a direct outcome of selecting the proxy with the minimum active streams. The Random and the Round-Robin methods show a slightly higher variance performance since they do not have any explicit load-balancing mechanism.

Performance of Rainbow cache replacement algorithm

We define relative cache size (RCS) of a SC as the ratio of its available cache size (ACS) to the minimum cache size (MCS) for which no cache replacement occurs. For example, 100 % RCS



(a) Average number of active streams per proxy (b) Standard deviation of number of active streams per proxy

Figure 13: Load balancing among proxies

represents no cache replacement, whereas any value of RCS less than 100 % will result in cache replacement. In the following evaluation we use the same RCS for all SCs. Using the parameters listed in Table 3, and the analysis of Silo layout in Section 4, we calculate the MCS for each SC. In the simulations, we vary RCS from 50% to 100%, compute corresponding ACS, and measure LHR, RHR and System-wide Hit Ratio. The results are shown in Figure 14.

The Rainbow algorithm performs very well over a wide range of RCS. When RCS changes from 100 % to 50 %, system-wide cache hit ratio only drops from 100 % to 90 %. In other words, 50 % reduction in cache size costs only 10 % loss of system-wide cache hit ratio.

The Rainbow algorithm also shows almost constant LHR value over the wide range of RCS from 50 % to 100 %. As RCS decreases, RHR behaves similar to system-wide cache hit ratio, decreasing very slowly untill 78 % at RCS of 50 %. The constant LHR and the decreasing RHR results from the fact that the Rainbow algorithm replaces segments from the least popular ones in the increasing order of segment caching potential. The replacement of unpopular segments does not affect the LHR since these segments are cached only on a very small number of SCs and user requests to the segments are usually satisfied through remote cache hits. As the unpopular segments are entirely replaced out from the Silo system, user requests to the segments cannot be satisfied within the Silo system and have to be satified by the origin server, decreasing the RHR and increasing SHR. Clearly, our experiement shows that the Rainbow algorithm scales well with varying RCS.

Performance comparison of RCache vs Silo

To analyze the performance of RCache scheme, we have implemented a set of different RCache models: (1) RCache with first segment caching and Inter-clip segment replication control (RCache-FI), where each SC always caches the first segment of all the clips and uses Inter-clip segment replication control. (2) RCache with no first segment caching but with Inter-clip segment replication control (RCache-NF), where each SC does not necessarily cache the first segment of all clips but uses the Inter-clip segment replication control. (3) RCache with no first segment caching and no Inter-clip segment replication control (RCache-NFNI), where each SC does not necessarily cache

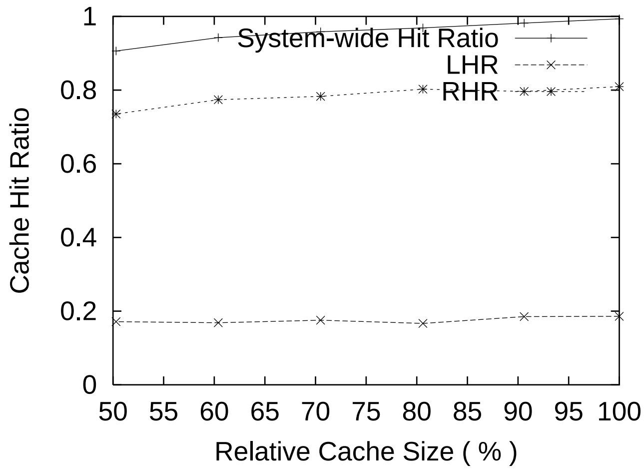
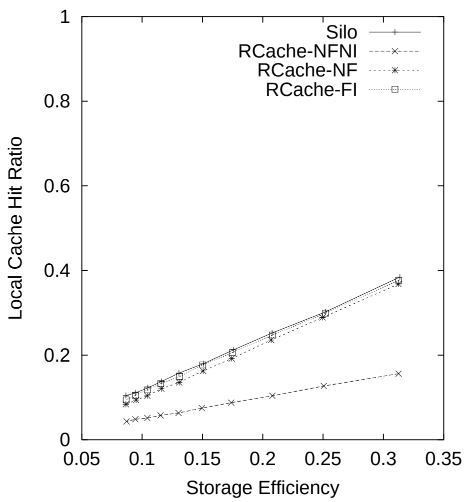


Figure 14: Cache Hit Ratios with varying cache size limit

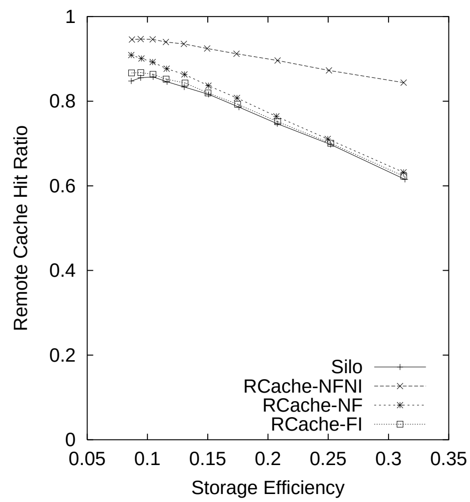
the first segment of all clips and does not use the Inter-clip segment replication control.

Figure 15 shows the cache hit performance of various RCache configurations and Silo. For this comparison, we fix the segment size of RCache to be the same as the first segment size of Silo. For local cache hit ratio, Silo performs best as in Figure 15 (a). Both RCache-FI and RCache-NF schemes show the next best performance with slightly lower cache hit ratio, while RCache-NFNI shows the worst performance. Figure 15 (a) shows an important fact that caching schemes with Inter-clip segment replication control give higher local cache hit ratio. As in Figure 15 (d), the system-wide cache hit ratios for all the caching schemes are similar, while the schemes with first segment caching show slightly lower cache hit ratio. Since the first segment caching option takes more space for storing all the first segments, less space are reserved for caching the remaining segments, resulting in slightly lower overall system-wide cache hit ratio. This shows a trade-off between the start-up latency and system-wide cache hit ratio. The caching schemes with first segment caching give better start-up latency performance at the expense of system-wide cache hit ratio. Figure 15 (b) shows the similar difference between the caching schemes with Inter-clip segment replication control and the caching scheme without Inter-clip segment replication control. The Inter-clip segment replication control gives higher segment caching probability to the segments of popular clips, giving higher chance of local cache hit for the popular segments. Due to the low local cache hit ratio, RCache-NFNI should utilize more segments from remote SCs. (see Figure 15 (b))

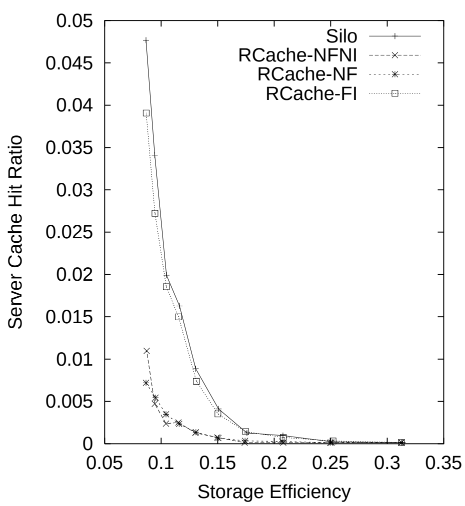
Figure 16 shows the switch-over performance. One big difference between Silo and RCache schemes is the number of switch-overs. Since the Silo utilizes larger segment size for the early segments, it allows less number of switch-overs for the large set of clip requests with small playback time [5], showing the smallest number of switch-overs. Figure 16 also shows that the Inter-clip segment replication control plays an important role in decreasing the number of switch-overs. RCache schemes with Inter-clip segment replication control, RCache-FI and RCache-NF, shows smaller number of switch-overs than that of RCache-NFNI. The difference between RCache-NFNI and RCache-NF shows the effect of Inter-clip segment replication control on the number of switch-overs.



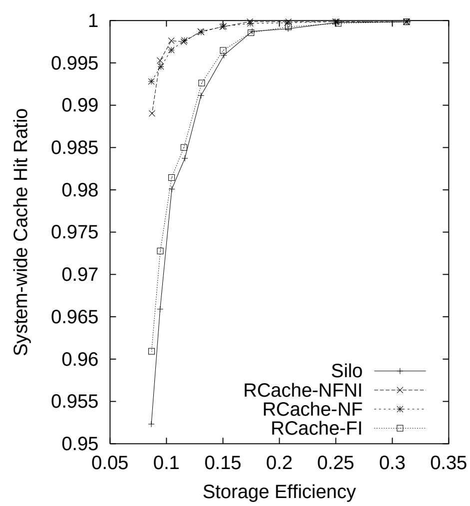
(a) Local Cache Hit Ratio



(b) Remote Cache Hit Ratio



(c) Server Cache Hit Ratio



(d) System-wide Cache Hit Ratio

Figure 15: Cache Hit Ratios with varying Storage Efficiency

In short, the first segment caching option gives better start-up latency performance while marginally sacrificing the system-wide cache hit ratio. By allowing higher chance of being cached for popular clips, the Inter-clip segment replication control plays an important role in increasing the local cache hit ratio and in decreasing the number of switch-overs. Finally, increasing the segment size for the early segments in Silo also gives better switch-over performance.

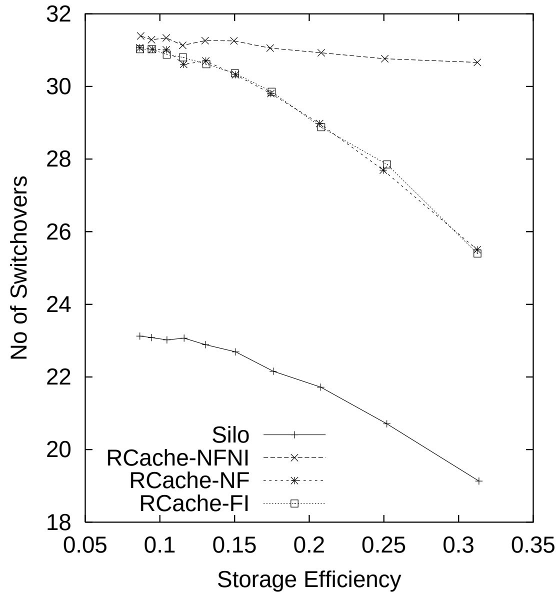


Figure 16: Number of Switch-overs with varying Storage Efficiency

8 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*, a randomized, easy to implement distributed data layout scheme with good storage efficiency. (2) *Silo*, a novel, fault-tolerant multimedia data layout scheme that accounts for the clip popularity and intra-clip segment popularity. This scheme is superior to naive clip replication and RCache layout. Using various parameters associated with this layout, one can control its storage efficiency and tradeoff storage for performance. (3) *Rainbow*, a *local* replacement scheme based on the concept of segment access potential used for data replacement at each cache. The Rainbow scheme captures the micro and macro popularity of clip segments and replaces only the least useful segments. (4) *Caching Token*, a dynamic *global* data replacement or redistribution scheme that exploits the existing data in the distributed cache to minimize distribution overhead. This scheme reduces the network and origin server load dramatically and is straight forward to implement.

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 analytical and 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. Silo provides 3 - 8 times as high cache hit ratio as that of traditional web caching system while utilizing the same amount of storage space. Silo 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) How to reduce or eliminate inter-cache state exchange and instead use mechanisms such as Anycast [20] to acquire state information just-in-time. (2) Investigation of simple proxy selection algorithms and/or segment distribution algorithms to achieve better load-balance. (3) Building a prototype system of Silo 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 NOSS-DAV'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] 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.

[11] 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.

[12] 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.*

[13] 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.

[14] Chen, M., Kandlur, D., and Yu, S., P., "Support for Fully Interactive Playout in a Disk-Array-Based Video Server," *Proc. of Second International Conference on Multimedia, ACM Multimedia'94*, 1994.

[15] Chen, P., et al., "RAID: High-performance, Reliable Secondary Storage," *ACM Computing Surveys*, Jun, 1994.

[16] Hua, K., Chai, Y. and Sheu, S., "Patching: A Multicast Technique for True Video-on-Demand Services", *Proc. of ACM Multimedia'98*, Sept, 1998.

[17] Hua, K., A., Sheu, S., "Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems," *Proc. of ACM SIGCOMM*, Sept, 1997.

[18] 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

[19] 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.

[20] Partridge, C., Mendex, T., and Milliken, W., "Host anycasting service," *RFC 1546*, Nov, 1993.

[21] 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.

[22] Sen, S., Rexford, J., and Towsley, D., "Proxy prefix caching for multimedia streams", *Proc. of IEEE Infocom'99*, Mar, 1999.

[23] Shenoy, P., "Symphony: An integrated Multimedia File System," *Doctoral Dissertation,* Dept. of Computer Science, University of Texas at Austin, Aug, 1998.

[24] Tewari, R., Vin, H., Dan, A., and Dias, D., "Resource Based Caching for Web Servers," *ACM Multimedia Systems Journal*, 1999.

[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.

Appendix A: Details of Caching Token Mechanism

When a Silo proxy receives a rank change request about a particular clip from the server, it determines a new cache layout as in Figure 17.

```
RECEIVE_RANK_CHANGE_REQUEST:
  pkt = receive_new_rank_pkt();
  new_rank = pkt.rank;
  clip = get_clip(pkt.clip);
  clip.rank = new_rank;
  /* calculate new segment caching probabilities */
  for ( i = 0; i < clip.last_seg; ++i )
        seg(i).caching_prob =
                new_segment_caching_prob(new_rank, i);
        old_token = seg(i).token;
        /* determine whether cache or not */
        if (uniform(0,1) <= seg(i).caching_prob)
            seg(i).token = 1;
        else
            seg(i).token = 0;
```

Figure 17: Handling rank change request

```
(a) SEND:
  pkt = receive_segment_request();
  segment = get_segment(pkt.segment);
  if(segment.cached == 1)
        start_streaming(segment);
  else
  /* if the proxy does not cache the segment and is supposed to */
  /* cache the segment, try token exchange */
        pkt.segment = segment;
        pkt.token = segment.token;
        send_segment_request(pkt);


(b) RECEIVE:
  pkt = receive_first_streaming_segment();
  segment = get_segment(pkt.segment);
  if(segment.token == 1)
        if(pkt.token == 0)
            /* token exchange occurs */
            segment.token = 0;
        else
            /* if token exchange does not occur, */
            /* the proxy needs to cache the segment */
            segment.cached = 1;
            cache_segment(segment);
```

Figure 18: Lazy caching and token exchange : initiator

Figure 18 illustrates the pseudo code for the initiation and termination of lazy caching and token exchange. The discrepancy between cached bit and token bit of a segment means that the Silo cache layout should be changed. As in Figure 18 (a), lazy caching is initiated by a proxy that does not cache a segment and is supposed to cache the segment. When a Silo proxy receives a streaming request for a segment that is not cached on the proxy but is supposed to be cached, it sends a segment request packet with token bit set to one of the remote proxies that cache the segment. As in Figure 18 (b), upon arrival of the first streaming packet for the requested segment, the proxy checks whether it has initiated a token exchange or not. If the proxy has initiated a token exchange and the returned token bit is clear, it means the remote proxy has grabbed the token for the segment and retains the segment in its local cache. Therefore, the proxy should release the right to cache the segment by clearing the token bit of the segment, terminating a successful token exchange. If the token bit is still set, it means the remote proxy has denied the token exchange. In this case, the initiating proxy should have the segment in its local cache.

```
RECEIVE_SEGMENT_REQUEST:

  pkt = receive_segment_request();
  clip = get_clip(pkt.clip);
  segment = get_segment(clip,pkt.segment);

  /* if token exchange is tried */
  if(pkt.token ==  1)
        /* if the proxy already has the token */
        if(segment.token == 1)
            schedule.segment = segment;
            schedule.token = 1;
            start_streaming(schedule);
        else
        /* if the proxy does not have the token */
            new_cache_size = cache_size(clip) + segment.size;
           /* if there is no concern about hot spot, do token exchange */
           if(new_cache_size <= (average_cache_size(clip) + threshold))
                segment.token = 1;
                schedule.segment = segment;
                schedule.token = 0;
                start_streaming(schedule);
           else
           /* if not, do not exchange token */
                schedule.segment = segment;
                schedule.token = 1;
                start_streaming(schedule);


  else
        schedule.segment = segment;
        start_streaming(schedule);
```

Figure 19: Lazy caching and token exchange : receiver

Figure 19 shows how to handle token exchange request. Upon arrival of a segment request, the proxy checks whether the requesting proxy has initiated a token exchange by examining the token bit of the request packet. Although the requesting proxy has initiate a token exchange, the proxy does not exchange token if it already has the token for the segment. In this case, the proxy informs the denial of token exchange to the requesting proxy by streaming the segment with token bit set. If not, the proxy tries to do the token exchange. If a proxy allows all the token exchanges, there is a chance that the proxy may cache more than expected number of segments, leading to a possible hot spot problem. Therefore, the proxy should check whether the current token exchange is safe or not. If the token exchange is allowed, the proxy grabs the token of the requesting proxy and set its token bit, changing the new cache layout. The proxy also informs the requesting proxy that the token exchange is granted by sending streaming packets with token bit clear. If the token exchange is not allowed, the proxy informs the denial of token exchange to the requesting proxy by sending streaming packets with token bit set.