Chase Display of Social Live Streams (SOLISs) *

Sumita Barahmand, Shahram Ghandeharizadeh
Computer Science Department
University of Southern California
Los Angeles, California
barahman,shahram@usc.edu

ABSTRACT

Advances in networking, processing, and mass storage devices have enabled social live streams (SOLISs) and their chase display. This paper focuses on public SOLISs and presents the user interface of RAYS and its system architecture. In addition, we present several novel memory management techniques that produce summary data to minimize the likelihood of cache misses. One technique, named Data-Aware CLRU (DA-CLRU), stands out for both enhancing the cache hit rate and utility of data. This technique is parallelizable and ideal for multi-core CPUs.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Performance evaluation (efficiency and effectiveness); H.3.5 [**Online Information Services**]: Web-based services; H.5.2 [**User Interfaces**]: Interaction styles/Graphical user interfaces (GUI)

General Terms

Algorithms, Design, Performance, Experimentation, Human Factors, Measurement

Keywords

Social Networking, Social Livestreams, Chase Display, Summary Generation, Stream Caching, Recency-based Cache Management Policies

1. INTRODUCTION

A social stream or a lifestream [7] is a chronological sequence of documents. The tail of the stream contains documents from the past. As a user moves to-wards the head of the stream, the stream contains more recent documents. The user may move beyond the present and into the future with the stream containing documents pertaining to reminders, calendar items, and to-do lists. This future facilitates remembering of intentions [12].

This paper focuses on those social streams that are continuous media generated by smartphones such as iPhone and shared with different viewers using a social networking site such as RAYS [8], Qik [10] or Bambuser [3]. With RAYS, for example, the owner of a device may invite other users to view her device at a pre-scheduled time in the future. By default, the invitee may not record the stream produced by the device[1]. The owner may grant recording privileges when extending an invitation to a user. With those invitations that allow no recording, the owner may specify a chase duration (Δ) for the display. Δ is the duration of time shifted viewing allowed for the invitee. We use the term SOcial LIve Stream, *SOLIS*, to refer to streams with such characteristics, i.e., no recording privilege with chase duration Δ.

Future data of a SOLIS is unknown because it has not occurred as yet. Its present data that is dropped by either the recording device or the social networking infrastructure is lost permanently [4]. Chase data pertains to the past and resembles a constantly evolving pre-recorded clip with display time Δ. The value of Δ might be different for each invitee and is specified when the owner of a device extends an invitation. Invitee i with $\Delta_{i,j}$ for SOLIS j should not be able to display data produced more than $\Delta_{i,j}$ time units ago by SOLIS j.

A chase display refers to an invitee viewing time shifted data produced within the past Δ time units. Figure 1 shows an example of a SOLIS with the owner of an iPhone streaming an event and two different viewers performing chase displays, see Section 2 for details of the user interface. While one may adapt and apply pre-fetching techniques to chase data to facilitate their display, these techniques cannot be applied to actively produced data because they do not exist.

In addition to smartphones, a user may employ inexpensive wireless cameras provided by Linksys and Panasonic (see Figure 3) to produce SOLISs. Thus, SOLISs may support diverse applications ranging from entertainment to surveillance and scientific collaborations.

Devices typically support a handful of simultaneous viewers due to either limited network bandwidth, processing capability, or both. For example, in our experiments with the Panasonic BL-C131 camera configured to use its wired network connection, we observed it to support 9 to 12 simulta-

*Supported in part using an un-restricted cash gift from Oracle Inc.

WSM'11, November 30, 2011, Scottsdale, Arizona, USA.

[1]The invitee may violate terms of use and utilize other resources to capture a stream and record it. Such violations are important. At the same time, they fall beyond the focus of this technical workshop paper.

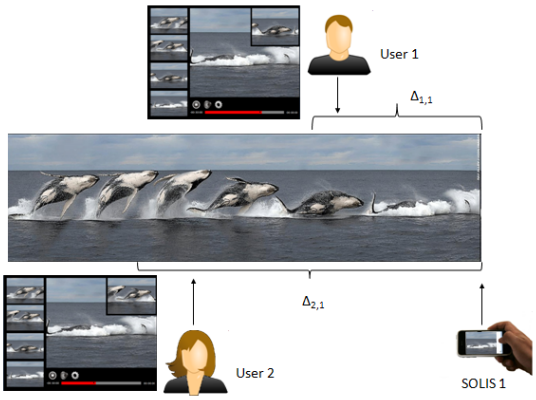


Figure 1: A SOLIS with two chase displays.

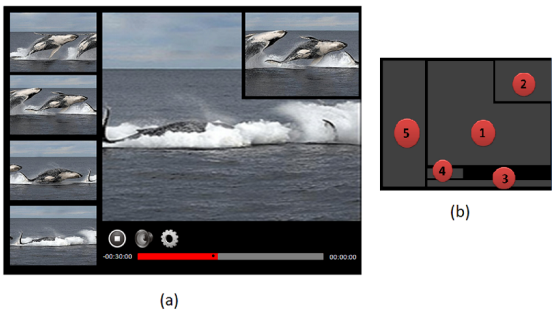


Figure 2: SOLIS user interface.

neous views. In order to support a large number of viewers for a device, it is important to utilize a social networking infrastructure such as RAYS. As an example, on April 20th of 2011, President Obama held a town hall meeting that was available for Internet viewing using Facebook's infrastructure. In addition, the social networking infrastructure may facilitate chase displays of SOLISs.

We taxonomize SOLISs into public and private ones. President Obama's Facebook event is an example of a public SOLIS. The number of viewers for a public SOLIS is typically not known in advance. Moreover, viewers may join and leave at arbitrary times. With a private SOLIS, the owner of the device extends an invitation to a fixed number of users. Thus, the maximum number of viewers for the SOLIS is known in advance. The system may construct a profile of different device owners and how their invitees access the system in order to manage resources intelligently. This paper focuses on public SOLISs. We assume the values of Δ is fixed[2]. Private SOLISs are a future research direction.

A large number of SOLISs consume system resources such as network bandwidth, available memory, and disk bandwidth. The exact amount is a product of the bandwidth of the stream and the value of Δ. A larger Δ value requires the system to maintain more data. Similarly, a higher data quality (resolution) results in a higher amount of data produced as a function of time (stream bandwidth), consuming more resources. When the system starts to exhaust resources, it may summarize data [6]. For example, with continuous media, we can summarize data by maintaining original audio along with a few key images [1]. Another possibility is to employ CPU resources to reduce the resolution of both audio and video, reducing both size and bandwidth requirements [11]. Ideally, summarizing should be applied to those chase data that are anticipated to be accessed rarely, providing users with the highest quality data.

This study presents an architecture to support SOLISs and their chase display. It uses a cache manager to store chase data for the various SOLISs in the system. Management of memory and a replacement policy for chase data is a key contribution of this paper. We describe several novel replacement techniques that produce summary data to maximize data utility.

The rest of this paper is organized as follows. Section 2 presents the user interface for a SOLIS and its chase display. We present an architecture of a system in Section 3. Section 4 presents several variants of LRU to manage both the data produced by a device and its summary when the system runs out of memory. Section 5 compares these alternatives, showing the superiority of one technique. Brief future research directions are offered in Section 6.

2. SOLIS USER INTERFACE

Figure 2 shows the SOLIS user interface with a user invoking chase display. This interface consists of five key components. We present these in turn based on the the numbering shown in Figure 2.b.

First, the SOLIS windowpane shows the in-progress SOLIS, the head of the stream.

Second, chase windowpane is an overlaying pane positioned on a corner of the SOLIS windowpane. In Figure 2, it is shown in the top right corner. When the user initiates a chase display, the system shows the chase windowpane. Note that the display of the in-progress SOLIS continues in the background windowpane.

Third, chase *progress* bar implements the concept of Δ. Using this bar, the user navigates chase data. A click at the rightmost limit of the progress bar, i.e., present, causes the interface to deactivate and remove the chase windowpane, focusing on the SOLIS windowpane and bringing it to the foreground.

Fourth, display *control* bar consists of three buttons. The stop button stops the SOLIS. Users employ the volume button to control their audio experience. The settings button controls the position of the chase windowpane and the number of tapestries, see below.

The stop and volume buttons have either a SOLIS or a chase display context. For example, if the chase windowpane is active, the user's manipulation of the volume button impacts the audio track of chase data. In SOLIS mode, the buttons function on the SOLIS display windowpane. Below, we elaborate on this.

When displaying the head of the SOLIS, if the user presses the volume button then the SOLIS display audio is muted. A second click of this button unmutes the audio. With SOLIS display mode, a click of the stop button terminates the stream and causes the player to display a black screen.

Once in chase display mode, the user pressing the volume

[2]A social networking site such as Facebook may utilize the profile of a viewer to control the value of Δ. Extensions of our techniques and their presented results with variable Δ values is trivial.

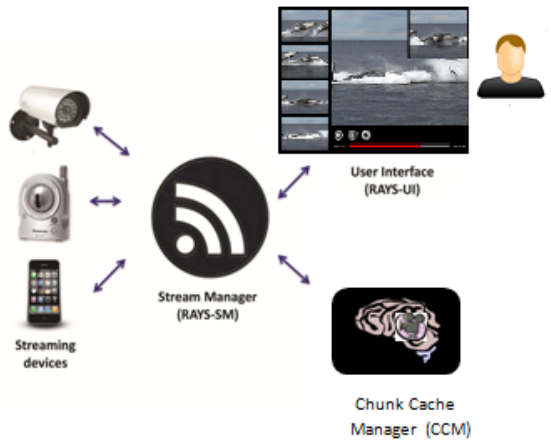


Figure 3: Architecture of RAYS and its components.

button impacts the chase audio (either mutes or unmutes it). Similarly, a click of the stop button terminates the chase display This closes the chase windowpane, bringing the SOLIS windowpane to the foreground.

Fifth, tapestry windowpane [5] displays several tapestries for the chase portion of the SOLIS. It indexes chase data. Once a user clicks on a tapestry, the interface activates the chase window to display chase data from the appropriate reference point.

This windowpane can be configured (using the settings button in the control bar) to show either all tapestries or a fixed number of them. With the later, the user may configure this pane to show those tapestries pertaining to either most recent, least recent, equi-interval, or highest entropy portion of chase data. For example the user may configure this pane to show four tapestries pertaining to the last twenty seconds of the chase portion of a SOLIS. Figure 2 illustrates the scenario where the user configured this windowpane to show four high entropy tapestries. These tapestries are generated by the infrastructure and updated based on the current chase portion of a SOLIS.

3. ARCHITECTURE AND DATA UTILITY

CCM manages fast dynamic memory and persistent memory such as magnetic disk and flash memory. Due to the page limits on this submission, we ignore alternative forms of persistent storage and focus on dynamic memory and its management, see Section 4.

When a user references chase data using RAYS UI (see Section 2), the user request is mapped to a data chunk of a SOLIS. The publisher of data (Red5 media server) uses this information to construct a unique key that identifies the data in the cache, see Section 4. CCM may return either the original chase data chunk (ChaD), chase summary chunk (ChaSm), or no data at all (cache miss). In the first two cases, the data is displayed. With summary chunk, it is possible for the viewer to dislike the display and navigate away from it using the chase progress bar or the tapestry windowpane. A cache miss is undesirable and we try to avoid it by managing memory intelligently, see discussions of Section 4. A cache miss causes the chase windowpane in the UI to display a blank screen while the in-progress SOLIS is displayed in the background.

Term	Definition
CCM	Chase cache manager.
ChaD	A chunk of chase data produced by a device.
ChaSm	A summarized ChaD chunk.
Parameter	**Definition**
$\mathcal{N}$	Number of SOLISs in the system.
Δ	Duration of a chase display.
δ	Display time of a ChaD/ChaSm.
T_N	Current system time.
T_S	Stream initiation time.
T_C	Time units before T_N for chase display.
α	User satisfaction with a ChaD.
β	User satisfaction with a ChaSm.
ψ	User dis-satisfaction with a cache miss.

Table 1: Terms and parameters.

To capture the different possibilities, we define the following utility function or measure of relative sastisfcation for a user referencing data:

$$\mu = \alpha \times D + \beta \times S + \psi \times M \quad (1)$$

D and S are the percentage of user references served using ChaDs and ChaSms, respectively. M denotes the percentage of requests that the system fails to service due to a cache miss. β and α are the level of satisfaction associated with ChaSms and ChaDs, respectively. ψ is assigned a negative value, reflecting user's level of dis-satisfaction with the system failing to display the referenced chase data.

Values of α, β and ψ are application specific. It is possible to assign a negative value to β. Moreover, ψ might be zero for certain classes of applications, reflecting no user dissatisfaction with the system's failure to show chase data. With social networking applications, we speculate (a) β to be a positive value and a fraction of α, and (b) ψ to be a negative value, see discussions of Figure 5.

4. MEMORY MANAGEMENT

We assume CCM is a main memory key-value store such as memcached [2] or COSAR [9]. With many SOLISs and/or large values of Δ, CCM may exhaust its available memory and require a replacement policy. Below, we detail how the system constructs the key for a chunk and the role of summary chunks. Subsequently, we explain why we do not use frequency based techniques and outline several recency techniques. Section 5 quantifies the tradeoff associated with these techniques.

Each SOLIS has a start time T_S defined by when the owner of the device initiated it. The resulting stream is partitioned into fixed δ time units, numbered in a monotonically increasing value (id): 0, 1, ..., k. k identifies the latest chunk being produced by a device and streamed by the Red5 server. Once it is δ time units in length, it becomes the k^{th} chase data, ChaD, and inserted into CCM with its time to live (TTL) set to Δ. Its key is the URL of the device concatenated with id (k) of the chunk. Note that each ChaD is identified by a unique key because each SOLIS has a unique URL and produces ChaDs that are assigned unique ids.

When a user references chase data corresponding to T_C time units ago (relative to now, T_N), Red5 computes the referenced ChaD id as: $\lfloor \frac{T_N - T_C - T_S}{\delta} \rfloor$. Red5 concatenates

Δ	10 minutes for all SOLISs and users
δ	1 second for all SOLISs
$\frac{Size(ChaSm)}{Size(ChaD)}$	$\frac{25KBytes}{125KBytes}$=0.2
Memory size	1 GigaByte
$\frac{Size(SCache)}{Size(DCache)}$	0.25
Access pattern	Uniform

Table 2: Values of simulator parameters.

the URL of the device with this computed id to look up the cache for data.

The cache manager deletes those ChaDs and ChaSms whose TTL expires, freeing memory for new ChaDs produced by streaming devices. Assuming ζ_i denotes the amount of chase data pertaining to SOLIS i, it is possible for sum of ζ_is to exceed the memory capacity. In this case, CCM must evict ChaD chunks from memory to insert a newly produced ChaD. Ideally, CCM should evict a ChaD that is not displayed by a future user reference. This is challenging to model for those applications whose future chase display requests are not known. For these, we strive to identify those chunks with the lowest probability of being referenced and summarize them to free memory. This ChaSm chunk is inserted into the CCM with the same key as the evicted ChaD. When a viewer performs a chase display causing the UI (and the Red5 server) to reference the key, CCM returns either a ChaD or ChaSm. Both are displayable by the UI.

It is challenging to compute the probability of reference to ChaDs and ChaSms because the number of requests for chase data is content dependent. Moreover, with small Δ values, the computed frequencies might be based on a few samples and suffer from significant noise. Below, we focus on recency of references to present several novel replacement techniques for CCM.

We taxonomize our variants of LRU based on whether they partition memory (termed *partitioned*) between ChaDs and ChaSms or require them to compete for the same memory space (termed *non-partitioned*). Below, we describe each approach and its techniques.

4.1 Non-Partitioned CCM

With a non-partitioned CCM, ChaD and ChaSm chunks of different SOLISs compete for the same memory space. Different devices produce ChaDs that are identified by unique keys and inserted into CCM. CCM assigns the current time stamp as their recency of reference because the Red5 server is streaming them for display. When CCM evicts a ChaD and summarizes it to produce a ChaSm, a key research question is what is the recency time stamp of the ChaSm? Below, we describe two possibilities named Inherit-time LRU, ILRU, and Current-time LRU, CLRU. A variant of CLRU, named Data-Aware CLRU, DA-CLRU, victimizes data chunks first. Below, we detail these techniques.

With Inherit-time LRU, *ILRU*, the recency of last access to a newly produced ChaSm is set to that of its ChaD. We implement it using a queue where its head corresponds to the least recently referenced CCM chunk and its tail points to the most recently referenced CCM chunk. The queue is a doubly linked list and a hash table maintains the mapping between the key for the chase data and an element in the list. When chase data is referenced, the server computes its key and performs an order one look up to determine if the data is in CCM. If this is the case then there is a cache hit and the data is retrieved from CCM. Moreover, the entry in the doubly linked list is pushed to the tail of the queue.

ILRU evicts the chunk that is at the head of the queue, the least recently referenced chunk. This might be either a ChaD or a ChaSm. Depending on the size of data being inserted, ILRU evicts as many elements of the head to accommodate the incoming data. Every time ILRU evicts an element that is a ChaD, it checks to see if it must victimize more elements to accommodate the incoming data. If this is the case then it deletes the ChaD from memory. Otherwise, it produces its ChaSm and inserts it at the head of the queue, i.e., ChaSm inherits the recency time stamp of its ChaD. This ChaSm must be referenced prior to the next eviction in order to move to the tail of the queue.

Current-time LRU, *CLRU*, is similar to ILRU with one difference: the newly created ChaSm is inserted at the tail of the queue. In essence, CLRU behaves as if the ChaSm encountered a cache reference, initializing its recent reference to the current CCM clock. This increases the likelihood of a ChaSm incurring a cache hit by a future reference.

DA-CLRU further increases the likelihood of cache hits for ChaSms by evicting ChaD chunks first, populating the cache with ChaSms. It manages ChaSms using their recency of references. We implement DA-CLRU using two queues: The first maintains recency of references for ChaDs while the second maintains recency of references for ChaSms. DA-CLRU evicts chunks corresponding to the head of the first queue as long as it is not empty. Once this queue is empty, it starts to evict ChaSms from the head of the second queue.

When compared with one another, DA-CLRU minimizes the likelihood of a chase display reference observing a cache miss, see Section 5.

4.2 Partitioned CCM

With a partitioned CCM, the available cache space is partitioned into two, data cache (DCache) and summary cache (SCache). ChaDs reside in DCache and ChaSms occupy SCache. This partitioning enables the memory to be managed and accessed at finer granularity. All devices produce data that is inserted into DCache. When DCache is full and a new ChaD is inserted, the system evicts the least recently referenced ChaD and summarize it to produce a ChaSm. This ChaSm is inserted into SCache which may evict a ChaSm from SCache. Below, we describe three replacement techniques.

The first technique, LRU-CurrentTimeLRU (LRU-CLRU), employs LRU to manage DCache entries and CLRU to manage ChaSms occupying SCache. When a ChaD is evicted from DCache, it is summarized and the resulting ChaSm is inserted into SCache with its recency time stamp set to current time. This is identical to the CLRU discussion of Section 4.1 with one difference: ChaSms do not evict ChaDs.

The second technique, LRU-AGE, is designed for those applications that access the most recently generated summary data more frequently than those generated earlier. It differs from LRU-CLRU as follows: References to ChaSms do not update their recency time stamp. In other words, the queue is static and does not change when user requests reference ChaSms. Note that LRU-AGE continues to use LRU to manages DCache.

The last technique, LRU-InheritAge (LRU-IAGE), is designed for applications whose viewers reference the most re-

$\mathcal{N}$	Non-partitioned				Partitioned
	ILRU	Random	CLRU	DA-CLRU	LRU-LRU/ AGE/IAGE
10	100%	100%	100%	100%	100%
20	67%	89%	100%	100%	100%
30	45%	68%	74%	100%	80%
40	34%	54%	56%	100%	61%
50	27%	44%	45%	100%	48%
60	23%	37%	38%	100%	41%
70	20%	32%	32%	95%	35%
80	17%	28%	29%	84%	31%
90	15%	25%	26%	74%	27%
100	14%	23%	23%	67%	25%

Table 3: Cache hit rate of alternative techniques.

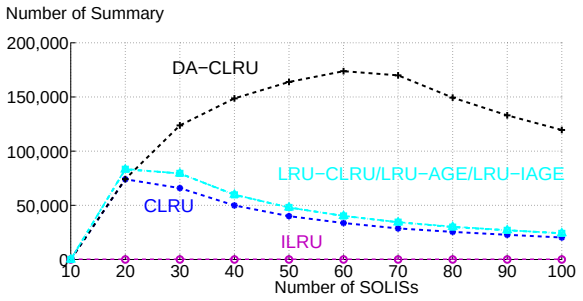


Figure 4: Number of ChaSms served by the alternative techniques.

cently generated data more frequently. It is different than LRU-AGE because it assigns the creation time of a ChaD as the recency time stamp of its ChaSm. It does an insertion sort of this entry in the queue of SCache. A SCache hit does not change either the time stamp of its entries or the order of entries in the queue. Similar to LRU-AGE, DCache is managed using LRU.

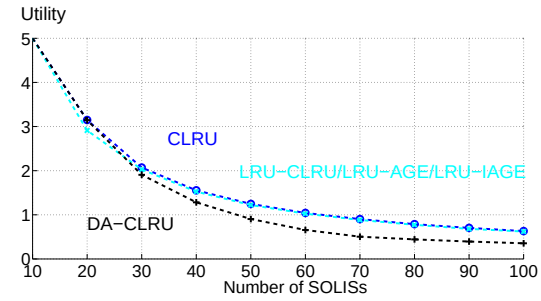
5. EVALUATION

We used an event driven simulation study to evaluate the alternative cache management techniques of Section 4. Below, we provide a brief overview of the simulator and its parameters. Subsequently, we present the obtained results.

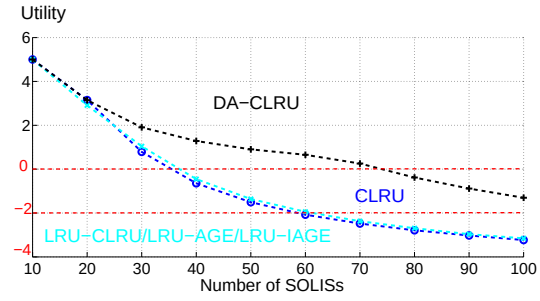
5.1 Event driven simulator

Our simulator models $\mathcal{N}$ SOLISs producing ChaDs simultaneously. Different SOLISs may produce data at different rates, competing for the available cache space. The simulator implements the alternative partitioned and non-partitioned cache management techniques of Section 4 and measures the observed cache hit rate. It distinguishes between ChaDs and ChaSms that observe a cache hit, computing the utility function of Equation 1. α, β, and ψ are input parameter values to the simulator.

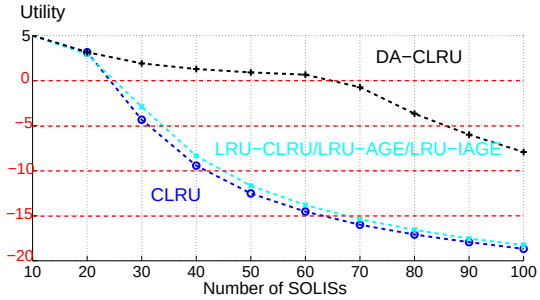
In addition, the simulator models a user i viewing a SOLIS j with a pre-specified chase duration $\Delta_{i,j}$. A user may reference one of $\mathcal{N}$ SOLISs using either a uniform or a Zipfian distribution of access. The simulator implements three different patterns of access for a user referencing chase data: Oldest chase chunk first, Most recent chunk first, and Random. The first two are modeled using a Zipfian distribution to the chunks that constitute a chase display.



5.a) ψ=0



5.b) ψ=-1



5.c) ψ=-5

Figure 5: Utility of data with different values for ψ.

5.2 Obtained results

We conducted many experiments using the different settings of the simulator. In general, DA-CLRU is superior to the other alternatives (both partitioned and non-partitioned). Below, we present results based on one setting of the simulator parameters. Subsequently, we describe how the obtained results change with other parameter settings.

Table 3 shows the cache hit rate of the alternative techniques for one setting of the simulator, see Table 2. For comparison purposes, we include numbers from a simple technique that evicts chunks using a random number generator, named Random. Key findings are as follows. First,DA-CLRU enhances cache hit rate while ILRU provides the worst hit rate. The alternative partitioned cache replacement techniques provide almost the same hit ratio when the users reference chase data using a uniform distribution of access. While Random out performs ILRU, it is not as effective as the other techniques with 20 to 30 SOLISs. Beyond 50 SOLISs, Random approximates CLRU and the partitioned techniques closely because the simulator is configured with the uniform pattern of access to the chunks.

DA-CLRU enhances cache hit rate because it summarizes

ChaDs aggressively to service as many requests as possible. ILRU cannot do the same because the most recently produced ChaSm is very likely to be evicted as its recency time reference is set to its corresponding ChaD that was just evicted. Figure 4 shows this discrepancy. The y-axis of this figure is the number of requests serviced using summary data. The x-axis is the number of SOLISs in the system. With fewer than 10 SOLISs, there is no need for summary data and all chase references are serviced using cache resident ChaDs. 20 SOLISs exhaust the cache space and different techniques start to summarize ChaDs into ChaSms. The number of serviced requests using ChaSms is only a few hundred with ILRU where as it is in the order of tens of thousands with the other approaches. DA-CLRU produces the highest number of summary data, maximizing the number of requests serviced using ChaSms.

Figure 5 shows the utility of data with α=1, β=0.1, and three different ψ values. When ψ equals zero, DA-CLRU produces a slightly lower utility than CLRU. There are two reasons for this. First, while CLRU allows ChaDs and ChaSms to compete with one another based on their recency of reference, DA-CLRU produces ChaSms aggressively by evicting ChaDs. Second, utility of ChaDs is ten times higher than ChaSms. With negative values of ψ, DA-CLRU minimizes the likelihood of a cache miss and outperforms other techniques. This gap widens when the user's level of dissatisfaction with the system's failure to service chase data is higher, i.e., larger negative values for ψ.

Obtained results show the partitioned techniques to provide either the same or better utility (cache hit rate) than CLRU. These techniques raise the following challenging question: How should memory be partitioned across DCache and SCache? The presented results assume DCache is 4 times larger than SCache, see Table 2. This approximates the assumed factor of compression (5) once a ChaD is summarized. Assuming the compression rate is fixed, when we reduce the size of DCache and increase the size of SCache, the partitioned techniques start to approximate the behavior of DA-CLRU. This is because they start to maintain more ChaSms by evicting ChaDs and summarizing them. Similarly, as we increase the size of DCache and take away memory from SCache, the partitioned techniques start to approximate ILRU.

With other simulator settings (such as pattern of access to data), the value of ψ has a significant impact on utility of data. Obtained results are identical to those presented in this section. With negative ψ values, DA-CLRU provides a better utility when compared with the other alternatives. When ψ equals zero, both CLRU and LRU-CLRU outperforms DA-CLRU. The margin of difference depends on the values of α and β. For example, when β is one tenth of α, the margin of difference is small, see discussions of Figure 5.a. However, if β is one hundredth of α, LRU-CLRU and CLRU outperform DA-CLRU by a wider margin.

With a skewed distribution of access to chase data, CLRU, LRU-CLRU and DA-CLRU outperform Random by a wider margin. These techniques use recency of references to identify frequently accessed chunks and maintain them in the cache longer. Random is unable to do the same.

6. FUTURE RESEARCH

While summarizing ChaDs is CPU intensive, one may parallelize the task and utilize multiple cores to summarize different ChaDs independently. With thousand core processors on the horizon, a system may support multiple summary formats for a ChaD. One may extend DA-CLRU to this environment by supporting one queue per summary format. The highest resolution formatted data is evicted first, producing the next lower resolution format. We intend to investigate this design and others as our next step.

Another near term research direction is to complete an implementation of DA-CLRU using COSAR and to provide it both as a public domain software and a component of RAYS, i.e., Stream Manager of Figure 3.

7. ACKNOWLEDGMENTS

We thank Jason Yap and Dieter Gawlick for their valuable feedback on the ideas presented in this paper.

8. REFERENCES

[1] H. Aoki, S. Shimotsuji, and O. Hori. A shot classification method of selecting effective key-frames for video browsing. In *Proceedings of the 6th ACM International Conference on Multimedia*, 1994.

[2] S. Apart. Memcached Specification, http://code.sixapart.com/svn/memcached/trunk/server/doc/protocol.txt.

[3] Bambuser. Live From Your Mobile, http://bambuser.com.

[4] S. Barahmand, S. Ghandeharizadeh, A. Ojha, and J. Yap. Three highly available data streaming techniques and their tradeoffs. In *Proceedings of the ACM Workshop on Advanced Video Streaming Techniques for Peer-to-Peer Network and Social Networking*, October 2010.

[5] C. Barnes, D. B. Goldman, E. Shechtman, and A. Finkelstein. Video tapestries with continuous temporal zoom. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 29(3), August 2010.

[6] M. Christel, M. Smith, C. Taylor, and D. Winkler. Evolving video skims into useful multimedia abstractions. In *CHI*, April 1998.

[7] E. T. Freeman and S. J. Fertig. Lifestreams: Organizing your electronic life. In *AAAI Fall Symposium: AI Applications in Knowledge Navigation and Retrieval*, November 1995.

[8] S. Ghandeharizadeh, S. Barahmand, A. Ojha, and J. Yap. Recall All You See, http://rays.shorturl.com.

[9] S. Ghandeharizadeh, J. Yap, S. Kalra, J. Gonzalez, E. Keshavarzian, I. Shvager, F. Carino, and E. Alwagait. COSAR: A Précis Cache Manager. Technical report, University of Southern California, Database Laborary Technical Report 2009-03, http://dblab.usc.edu/users/papers/cosar09.pdf.

[10] qik. Record and Share Video Live From Your Mobile Phone, http://qik.com.

[11] K. Raijkse. Video coding for low-bit-rate communication. *IEEE Comm. Magazine*, 34(12):42–45, 1996.

[12] A. Sellen and S. Whittaker. Beyond total capture: A constructive critique of lifelogging. *Communications of the ACM*, 53(5):70–77, May 2010.