An integrated Scheme to Distribute Segmented Streaming Media over Hierarchical Caches

Zhou SU, Jiro KATTO, Takayuki NISHIKAWA, Munetugu MURAKAMI, Teruyoshi WASHIZAWA and Yasuhiko YASUDA

***Abstract*—The emergence of the Internet as a pervasive communication medium has led to the rise of many applications of streaming media. However, because of their distinct statistical properties and user viewing patterns, traditional delivery and caching schemes for web objects such as HTML files or images can not be efficiently applied to streaming media such as audio and video. In this paper, we therefore propose an integrated caching scheme for streaming media with segment-based caching and with hierarchically distributed proxies. Firstly, each stream is divided into segments and their caching algorithms are considered to determine how to distribute the segments into different level proxies efficiently. Then, by introducing two kinds of segment priorities, segments replacing algorithms are proposed to determine which stream and which segments should be replaced when the cache is full. Finally, a web friendly caching scheme is proposed to integrate the streaming caching with the conventional caching of normal web objects. Performance of the proposed algorithms is verified by carrying out simulations.**

***Index Terms*-- stream caching, web caching, content delivery networks, hierarchically distributed proxies**

1. INTRODUCTION

With the growth in popularity of the Internet and the wide availability of high-speed networks, an increasing number of streaming media objects are being distributed over the Internet.

However, web caching and delivery of web objects such as HTML files or images can't be efficiently used for streaming media such as video and audio. Steaming media has several inherent properties: Firstly, the size of streaming media is usually larger than non-streaming files by orders of magnitude [5]. Secondly, user access behavior shows different characteristics. For example, clients sometimes stop at one stream without watching all of the parts [6]. Thirdly, in contrast to other web objects, streaming media do not require to be delivered at once. Instead, the server usually pipelines the data to clients through the network.

Z.Su, T.Nishikawa, M.Murakami, J.Katto, Y.Yasuda are with the Graduate School of Science and Engineering, Waseda University, 3-4-1, Ohkubo, Shinjuku-ku, Tokyo 169-0072, Japan Contact E-mail: suzhou@yasuda.comm.waseda.ac.jp

T.Washizawa is with the Canon Research Center, Canon. Inc, 5-1, Morinosato-Wakamiya, Atsugi-shi, Kanagawa, 243-0193, Japan

ICITA2002 ISBN: 1-86467-114-9

In this paper, we therefore propose an integrated delivery and caching system for streaming media, where each stream is divided into segments and these segments are distributed among hierarchically distributed cache servers.

Firstly, because storing the entire stream in a single proxy cache is inefficient or even impossible due to its large size, different segment-based caching algorithms are proposed and compared. A part of the requested stream is cached in a local cache, and the remainder of the stream will be cached in an upper proxy cache.

Secondly, by convention, the same popularity was assigned to the whole stream when the classical replacing algorithm was carried out, however, each different stream has a different popularity and each segment has different access patterns. Therefore, in this paper, two priorities for each segment are introduced: one reflects its access property, and the other represents its position information in the stream. Two kinds of replacing algorithms are compared to decide which segments of which streams should be removed when the cache exceeds its limit. One of the two algorithms keeps the same relative length of each stream in the cache, while the other keeps the most accessed segments in the cache.

Finally, how to coordinate the streaming caching with the current caching scheme for normal web objects such as HTML files or images is considered. By introducing the EWMA (Exponential Weighted Moving Average) estimator, a web friendly caching scheme, which can improve the performance of the whole caching system, is proposed.

2. PREVIOUS WORK

There are numerous works on replacing algorithms for normal web objects such as HTML files and images [7]~[13]. However, detailed behaviors or simulation results of these algorithms has not been studied or shown for streaming media such as audio and video.

Content delivery networks (CDNs) appeared recently and are deployed quite rapidly [1]~[4]. Load balancing by request routing, efficient content delivery by locating edge servers near to clients and information exchange protocols among different CDN sites are developed. However, their concern is mainly placed on efficient delivery of static contents, i.e. HTML files and images.

There have been some published studies on stream caching. Sen et al. [17] showed that caching a prefix (i.e., the initial part) of a stream at the proxy can hide the potentially large initial start-up delay of the work-ahead transmission schedule from

the client. Lee et al. [18] proposed a scheme that provides users with the video summary (a number of key-frame images) before they download the file. Recently, Kangasharju et al [16] studied distributing layered encoded video through caches. However, the first two are immature from the viewpoint of complete systems. The last one is also limited in a sense that a video frame has to be encoded into multiple layers instead of a group of video frames, i,e, segments, which characterizes our work.

We previously proposed a novel stream caching method using hierarchically distributed proxies [22] and a Web friendly stream caching [23]. This paper is therefore characterized as an extended version of these works after futher improving segment handling strategy (i.e. segment caching and segment replacement).

3. SYSTEM ARCHITECTURE

In this section, we will introduce an overview of our system architecture, which can be also applied to content delivery networks.

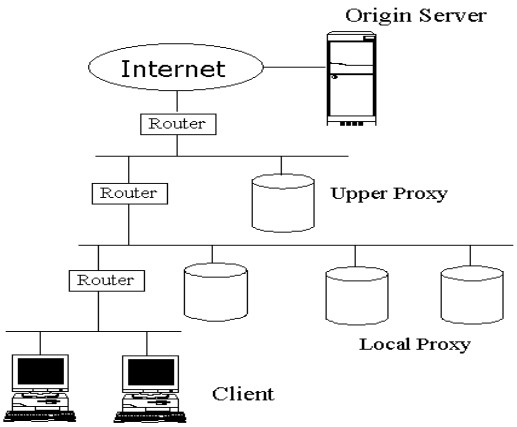

Fig.1 Distributed Cooperate Caching System

In our system depicted by Fig.1, we assume that each streaming media is divided into segments based on the result of shot boundary detection. Each segment corresponds to a continuously recorded sequence of frames such as multiple GOPs (group of pictures) and can be managed separately.

Recent studies [14][15] have shown that a number of proxies cooperating and sharing contents with one another can improve cache hit ratio. Therefore, distribution of the segments of streaming media among multiple proxies is also expected to result in performance improvement of stream caching as long as cache sharing and cooperative caching are done efficiently.

Our proposed caching system consists of an origin server, upper proxies and local proxies. As shown in Figure 1, local proxies are directly connected with clients. When a clients sends a request to a local proxy and the requested data can't be provided, the local proxy sends an ICP-like query to another local proxies to ask to see whether they have the requested contents. If the requested content can't be found in any local proxies, this request will be sent to upper proxies. If there is no requested content yet in the upper proxies, the request is sent to an origin server and it will send the requested data.

Note that our proposed architecture can be directly applied to content delivery networks. Local proxies are "edge servers" Some of upper proxies and local proxies can be grouped and a CDN site is composed. The original server may be expanded to multiple origin servers. Request routing methods, such as DNS redirection and URL rewriting, are also effective in our system. We want to emphasize our formulation below is effective to any types of CDNs.

4. PROPOSED ALGORITHMS

To improve the performance of stream caching, stream sequence segmentation and hierarchical distribution of cooperative cache servers are introduced in this paper. Problems to be solved are as follows.

1. How to distribute one stream into different level proxies?
2. As cache capacity is limited, which segment of which stream should be removed from cache when the cache is full?
3. How to integrate the streaming caching with the conventional caching of normal web objects?

The former is discussed in subsection 4.1. The latter two ones are discussed in subsection 4.2 and 4.3, respectively.

4.1 Segment Caching

In conventional methods, the entire data of one stream is either completely cached in a local cache or not. There are some disadvantages in these approaches. Firstly, the cache has to purge some currently cached data to make room for the incoming stream to satisfy current request. Because of its large size, many streams or normal web objects in the cache will be removed even some of them are frequently requested by many clients. Secondly, sometimes not all the requested streams will be completely observed by clients. Caching the entire stream is not efficient in this case.

Let M be the number of streams in an origin server, and L_i be the length of the stream i ($i=1, 2, \ldots, M$). S_{seg} denotes the size of one segment, and B_i is the number of segments of the stream i. For simplicity, every segment is assumed to have the same size.

We also introduce a vector of cache indicators $\boldsymbol{C}=(C_1, C_2, \ldots, C_M)$, with $0 \le C_i \le B_i$. Each C_i is set to n if n segments of the stream i is cached in a cache. Note that $C_i=0$ means that there is no segment of the stream i in a cache. With this notation, cache space occupied by cached segments, $G(\boldsymbol{C})$, is given by

$$G(C) = \sum_{i=1}^{M} C_i \times S_{seg} \qquad (1)$$

Three schemes are proposed as follows:

- *Last Watch*: Assume that stream i has been requested by Q times within a fixed period and each time the client stopped watching stream i after the playback of segment $S_{q,\mathrm{i}}$ ($q \in (1,Q)$, $i \in (1,M)$, $S_{q,\mathrm{i}} \in (1, B_i)$), where $S_{q,\mathrm{i}}$ means that the client finished watching total $S_{q,\mathrm{i}}$ segment of stream i when he requested this stream at the q-*th* time. In this approach, each local cache keeps the parameter $S_{Q,\mathrm{i}}$ for the Q-*th* access. When a new request for this stream i occurs next time ($(Q+1)$-*th* time), the Last Watch approach caches

only the segments before the position ($S_{Q,i}$) in a local cache. The remaining segments are not cached by the local cache and will stay in an upper cache.

- *Average Watch*: Different from the Last Watch, the Average Watch method keeps a series of the parameters $S_{q,i}$ for the past Q accesses. When a new request for the stream i occurs at the (Q+1)-*th* times, the average watch approach caches only the segments before the position ($\frac{1}{Q}\sum_{q=1}^{Q} S_{q,i}$) in a local cache. The remaining segments are cached in an upper cache.

- *Judging*: As the local cache has limited capacity, cached segments need to be removed from the local cache to make room for incoming segments when the cache is full. Therefore, a method to judge whether an incoming segment should be cached or not is introduced. We introduce two new parameters, $p_{request}$ and p_{remove}. $p_{request}$ represents a priority of the segment of a requested stream, and p_{remove} does a priority of the segment that needs to be removed from a local cache. If $p_{request}$ is higher than p_{remove}, the segment is cached. If not, the segment is removed on the contrary. The details of how to determine and assign these priority values will be discussed in Sect.*4.2.1.*

4.2 Segment Replacing

When a new object comes in and the cache exceeds its limit, one or more objects in the cache must be removed in order to make room for the newly accessed object. The replacing algorithm is to decide which object should be removed.

4.2.1 Segment Priority

Since we need to cache the stream segment by segment, one question is how to decide the priority of each segment. Well-known conventional method to decide object priority for replacing algorithms is *LRU*. However, researches showed that one of the main weaknesses of this *LRU* is that the cache would be flooded by objects referenced only once, flushing out documents with higher probability of being reused.

There are some other methods. However, most research concerns normal web objects. How to efficiently define priority for streaming media has not been proposed.

- 1 Priority Index of Access (*PIA*):

Through theoretical analysis, we firstly propose a priority index of access (*PIA*) for streaming media. Assume that stream i has been requested by Q times within a fixed period and each time the client stopped watching stream i after the playback of segment $S_{q,i}$ ($q \in (1,Q)$, $i \in (1,M)$, $S \in (1,B_i)$) as shown before. Priority Index of the j-th segment of stream i is then defined as follows:

$$PIA = \left| \{ S_{q,i} \mid j \le S_{q,i} \} \right| v_i / (T - T_{c,i}) \qquad (2)$$

Here |.| denotes the number of elements of a set. T and $T_{c,i}$ are respectively the present time and the time when stream i is cached in a cache. v_i denotes the request times between $T_{c,i}$ and T. For details, please see Appendix.

In the above equation, by using $S_{q,i}$, the system does not have to keep the numerous request records for all segments of all streams. And both access times (v_i) and the cached time ($T_{c,i}$), are considered.

- 2 Priority Index of Relative Length (*PIRL*):

As we mentioned before, not only do different streaming media have different popularities, but also access varies by position of each segment inside the stream. By taking this fact into account, another parallel priority for each segment is defined. We firstly define a relative length l_{rela}, which is a ratio of the serial number of this segment to the number of total segments in this stream. Then, assume that the total number of segments of stream i is B_i and the current vector of cache indicators $C=(C_1, C_2, \ldots, C_i, \ldots, C_M)$ as before, which means that there are C_i segments of stream i in a cache. Let the serial number of the beginning segment be one, then, the last segment of this stream has a relative length:

$$l_{rela} = C_i / B_i \qquad (3)$$

We call this l_{rela} the priority index of relative length (*PIRL*).

4.2.2 Replacing Algorithms

Based on the two segment priorities defined before, two kinds of replacing algorithms are proposed:

- 1.Popularity Policy: Firstly, the cache selects the segments with the lowest *PIA*. Then, if more than one segment have the same lowest *PIA*s, the one with the highest *PIRL* will be removed.

- 2.Balance Policy: Firstly, the cache selects the segments with the highest *PIRL*. Then, if more than one segment have the same *PIRL*s, the one with the lowest *PIA* will be removed.

Here, we explain more about the Balance Policy algorithm in detail. Research [17] showed that keeping the first few segments of one stream in a cache could reduce start-up latency. The reason is: if the beginning part of one stream is available, the remainder of the stream can be sent to the client during the playback time of the beginning part. Because the Balance Policy tends to keep the same relative length of different streams in a cache, it will clearly be good for cache performance. Furthermore, if the end segments of two streams have the same l_{rela}, the one with the largest stream size will be removed from the cache. This makes more cache space available for an incoming object.

A disadvantage of the Balance Policy is a case where the beginning segment of one stream will remain in a cache forever even if it has not been requested for a long time. To solve this problem, we introduce a 'zero check' strategy, which means that streams whose recent request frequency is zero are removed from a cache.

4.3 Web-Friendly Streaming Caching Scheme

It can be expected that an increasing number of streaming media will be distributed over the Internet with its further development. Since more cache space is likely to be occupied by the streaming media, a cooperative caching method with normal web objects such as images and HTML files should be

discussed (otherwise, large streaming data causes removal of popular Web (static) contents). Large-scale CDNs may prepare multiple servers for Web objects (HTTP server) and streaming ones (e.g. RTSP server) separately, it costs much and a single server approach sharing a cache space is still preferable for small-scale CDNs and P2Ps. Therefore, we introduce here novel "Web-friendly" caching (WFC) algorithms.

In our proposal, 0the proxy cache firstly detects clients' access conditions and calculates a proportion of cache capacity being taken by streams (p_{stream}). Secondly, if the next request for a streaming object comes and p_{stream} exceeds a fixed threshold (*Th*), the cache will remove stream data instead of normal web objects in order to make room for the newly requested object.

In detail, the EWMA (Exponential Weighted Moving Average) estimator is introduced as follows:

$$p_{avr}(t) = \omega \times p_{stream} + (1-\omega) \times p_{avr}(t-1) \qquad (4)$$

Where p_{stream} is the proportion of the cache capacity being taken by streams and $p_{\text{avr}}(t)$ is the averaged proportion at time t, and ω is a smoothing constant, respectively. When a cache is full and the cached data need to be removed to make room for the newly requested object, if $p_{\text{avr}}(t)$ of Equa.(4) exceeds the threshold (*Th*), the cache will select streaming data to be removed according to the algorithm in Sect 4.2.

5. EVALUATION of ALGORITHMS

5.1 Simulation Conditions

We assume that there are 1000 different streams with the rate of 64kbps. The length of each stream is uniformly distributed between one minute and ten minutes [5]. The size of one segment is 48Kbyte. The position where a client stops watching a stream is decided by random. In the simulation experiments the request type and the stream requested are drawn from a Zipf distribution with a Zipf parameter of 0.8[19][20][21]. The distributed cooperative system is made up of one origin server, one upper proxy and four local caches. The capacity of each local cache is 3% of the total size of all streams in the origin server. The capacity of the upper cache is double the capacity of the local cache. The total request in the simulation is 20000 times and the parameter *T* in Equa.*2* is 5000 times. The buffer size of clients is 240Kbyte.

Hit-ratio and Byte hit-ratio [10] have been popularly used to evaluate the cache performance by most researchers. Here, we define a Segment Hit-ratio as the ratio of the data size of segments directly satisfied by a cache to the total size of segments requested by clients. Since the size of streaming objects is very large and researches showed that keeping the initial part of the stream in a cache is helpful to reduce delay, we define another hit ratio called Partial Hit-ratio. This Partial Hit-ratio is defined as the number of times when the size of the cached segments of the requested stream are more than client's buffer size divided by the number of total requests.

5.2 Simulation Results

5.2.1 Caching Algorithms Evaluation

In this subsection, performance of caching algorithms discussed in Sect.*4.1*: *Last Watch*, *Average Watch*, *Judging* are compared with the conventional algorithm without any segmentation. In the *Judging* algorithm, whether or not an incoming segment should be cached in a local cache is decided by the priority of this segment. Here *PIA* defined in the Appendix is used to decide segment priority. To study these caching algorithms under the same conditions, the same replacing algorithm (Popularity Policy in Sect.*4.2.2*) is selected as the corresponding replacing algorithm in our simulation. The conventional one is the one that the stream is not segmented and the priority is given based on the unit of one stream instead of one segment, which means that the whole stream has the same priority regardless of segments. Also, the priority for one stream is decided by *LRU,* which is the most commonly used.

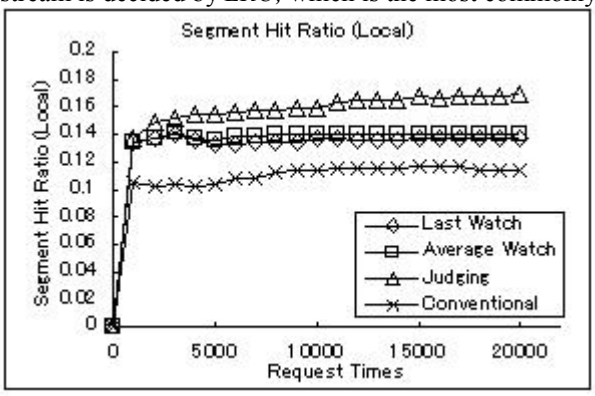

Fig 2: Comparison of Segment Hit-ratios of Local Proxy Caches with Different Caching Algorithms.

Fig.2 shows a comparison result of the Segment Hit-ratio of the local caches with respect to request times. All of the three proposed algorithms outperform the conventional one. The *Last Watch* and *Average Watch* provide a modest improvement while the *Judging* achieves the best result. The reason is that whether an incoming segment should be cached in a local cache is decided by the value of *PIA* based on the unit of segments according to the *Judging* policy. It can keep the clients' most accessed segments in a cache so that the Segment Hit-ratio is improved.

5.2.2 Replacing Algorithms Evaluation

In this second step, performances of different replacing algorithms discussed in Sect *4.2* are compared. There are four replacing policies we will study, which are *LRU Policy, LFU Policy*, *Popularity Policy* and *Balance Policy*

The former two are conventional ones, which we have introduced by assigning *LRU* and *LFU* to one whole stream (not segments). The last two are introduced in Sect.*4.2.2*. The proposed priorities, Priority Index of Access *(PIA)* and Priority Index of Relative Length (*PIRL*), are used in these two algorithms. A 'zero check' strategy in the Balance Policy is carried out every 2000 requests. Because the simulations in Fig.2 showed that the *Judging* policy performs best, we will compare the replacing algorithms under the same situation where the *Judging* policy is applied as a caching algorithm.

In Fig.3 we show a comparison of the results of the Segment Hit-ratio in local caches as a function of request times. Among four algorithms, It can be found that the Popularity policy gets

the best result. Let's refer to the Appendix for explanation. *PIA* is related to clients' access records, accumulative access times and the period of recent accesses. As a result, the cache can cache the objects according to recent client access patterns so that it can improve the Segment Hit-ratio. The Balance policy shows poorer performance than the Popularity policy . The reason is that it is likely to keep the initial parts of unpopular streams in the cache resulting in wasted cache resources.

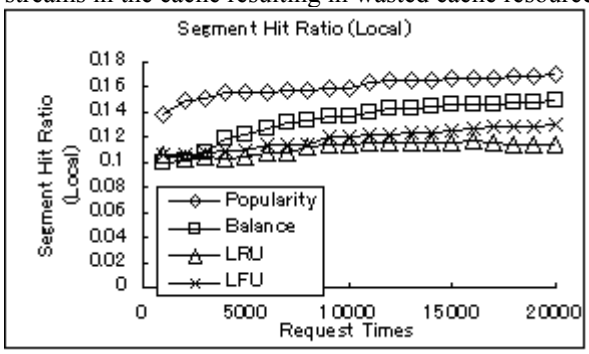

Fig 3: Comparison of Segment Hit-ratios of Local Proxy Caches with Different Replacing Algorithms.

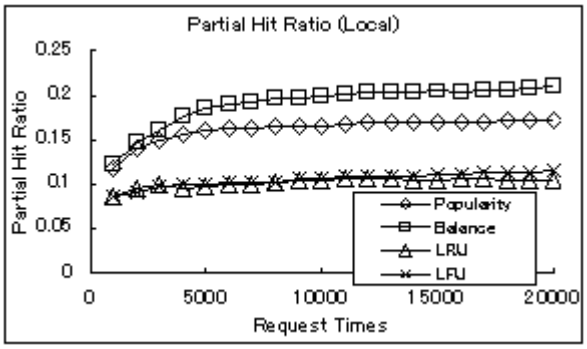

Fig 4: Comparison of Partial Hit-ratios of Local Proxy Caches with Different Replacing Algorithm

In Fig.4 we plot the Partial Hit-ratio for all algorithms mentioned previously. If the first few segments of one stream are available in the client's buffer, the remainder of the stream can be sent to the client during the playback period of the beginning part in buffer. In Fig.4 we find that the Balance policy obtains the highest Partial Hit-ratio while the Popularity policy showed the best performance in Segment Hit-ratio before. The reason is that the Balance policy tends to keep the same relative length of different streams in a cache. That is to say, large-size streams are likely to be removed to make room for other streams. As a result, more kinds of streams can be stored in a cache at the same time and the Partial Hit-ratio becomes higher.

5.2.3 Web Friendly Caching.

The scheme introduced in Sect.4.3 will be carried out with parameters ω =0.002 and *Th* =0.7. We assume that a server contains 1000 different streams and 10000 normal web objects. Streams have the same characteristics as mentioned before. The size of a normal object is 60Kbyte. At the beginning of the simulation, the request arrival rate of streaming objects is 5% of normal web objects'. Then it is continually increased during the time of the first 50000 requests until both streaming data and normal web objects have the same request arrival rate. The total request time is 100000.

It can be found that the Segment Hit-ratio of the streaming media decrease about 7% while the Segment Hit-ratio of normal web objects can still be increased to around 23% in Fig.5. In one word, our WFC algorithm leads to "balanced" Segment Hit-ratio between web caching and stream caching. Therefore, it is confirmed that performance of the whole caching system is improved by using WFC.

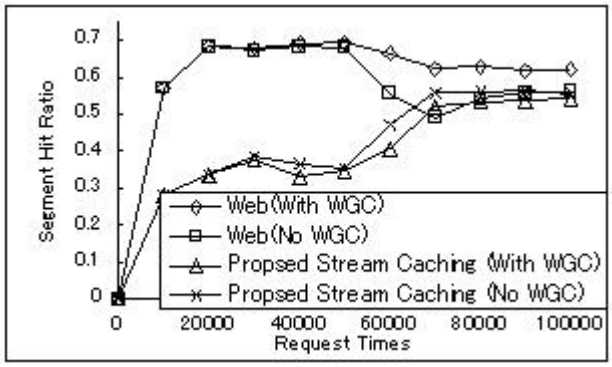

Fig 5: Comparison of Segment Hit-ratios (of an Upper Proxy Cache and Local Proxy Caches) with/without carrying out WFC into the proposed stream caching scheme

6. CONCLUSION

In this paper we design a system to spread the segmented streaming media into a cooperative distributed system. Three different caching algorithms were then proposed in order to cache a part of one stream into a local cache in an efficient manner. Among the proposed algorithms, simulation results showed that the *Judging* achieved the best performance. Next, two priorities for streaming (priority index of access (*PIA*) and priority index of relative length (*PIRL*)) were proposed. By combining them, two replacing algorithms were proposed and both the proposed algorithms outperform the conventional ones. Results show that the Popularity policy is better to deal with network congestion while the Balance policy is helpful to achieve shorter response time. Finally, we apply a web friendly caching scheme into our proposed system. Simulation results showed it can efficiently integrate the streaming caching with the conventional caching of normal web objects.

REFERENCES

[1] J.Kangasharju and K.W. Ross," *Performance Evaluation of Redirection Schemes in Content Distribution Networks"*, The 5th International Web Caching and Content Delivery Workshop, May 2000.

[2] A.Beck and M. Hofmann, "Enabling the Internet to Delivery Content-Oriented Services," The 6th International Web Caching and Content Distribution, Boston, USA Jun 2001.

[3] Adero, <URL: http: www. adero. com>

[4] Akamai, <URL: http: www. akamai. com>

[5] M. Chesire, A. Wolman, G.M.Voelker, and H.M.Levy, "Measurement and Analysis of a Stream Media Workload", USITIS'01, San Francisco, CA, Mar.2001.

[6] S. Acharya, B. Smith and P. Parnes, "Characterizing User Access To Videos on the Videos On the World Wide Web'' SPIE/ACM MMCN2000, San Jose, CA, Jan 2000.

[7] M. Arlitt, R. Friedrich, and T. Jin: "Performance Evaluation of Web Proxy Cache Replacement Policies'', Lect. Notes Computer Sience, Vol.1469, pp.193-206 1998

[8] Y. Yasuda, T. Yasuno, F. Katayama, T. Toida, and H. Sakata, ''Image Database System Featuring Graceful Oblivion", IEICE Trans., Commun., Vol.E79-B, No.8, pp.1015-1021, August 1996

[9] Z. Su, T. Washizawa, J. Katto, Y. Yasuda; "A New and Robust Replacement Algorithm for Proxy Caching with Hierarchical Image Coding" 2001 Image Media Processing Symposium (IMPS2001), Nov 2001

[10] A. Ortega, F. Carignano, S. Ayer, and M. Vetterli, "Soft Caching: Web Cache Management Techniques for Images", IEEE Signal Proc. Society Workshop on Multimedia Signal Processing, (Princeton, NJ), June 1997.

[11] T. Ishikawa, W. B. Hui, H. Ohsawa, and Y. Yasuda "A Method of Improving Response Time in Still Database based on CD-ROM Changers by Graceful Caching'' IIEEJ, 1999,Vol 28,No 5

[12] Z. Su, T. Washizawa, J. Katto, Y. Yasuda, "A New Prefetching Algorithm for Graceful Caching System", 2001 General Conference of IEICE, March 2001.

[13] Z. Su, T. Washizawa, J. Katto, Y. Yasuda, "Performance Improvement of Graceful Caching by Using Request Frequency Based Prefetching Algorithm", IEEE TENCON, Singapore, Aug, 2001.

[14] P. Rodriguez, C. Spanner, E. W. Biersack, " Web Caching Architectures: Hierarchical and Distributed Caching" The 4th International Web Caching Workshop, Mar.1999

[15] S. Paul, Z.Fei, " Distributed Caching with Centralized Control" The 5th International Web Caching and Delivery Workshop, May.2000

[16] J.Kangasharju, F.Hartanto, M.Reisslein, K.W. Ross, "Distributing Layered Encoded Video through Caches", IEEE INFOCOM01, N.Y, April.2001.

[17] S.Sen, J.Rexford,and D.Towsley, " Proxy Prefix Caching for Multimedia Streams," IEEE INFOCOM99, N.Y, Mar.1999.

[18] Sung-Ju Lee, Wei-Ying Ma, and Bo Shen" An Interactive Video Delivery and Caching System Using Video Summarization", WCW*2001,* Boston, MA, June 2001

[19] V. A. F. Almeida, M. A. G. Gesrio et al" Analyzing the Behavior of a Proxy Server in the Light of Regional and Cultural Issues" 1998.

[20] L. Breslao, P. Cao, L. Fan, G. Phillips, and S. Shenker "Web Caching and Zip-like Distributions: Evidence and Implications" Proc. IEEE INFOCOM'99

[21] J. Gwertzman. "Autonomous Replication in Wide-Area Networks". Technical Report 17-95, Harvard University.

[22] K.Ebisawa, J.Katto and Y.Yasuda, "Distributed Caching for Multimedia Streams," IEICE Autumn Conference, B-7-84, Sep.2001 (Letter in Japanese).

[23] K.Ebisawa, J.Katto and Y.Yasuda, "A Web-Friendly Streaming Caching Scheme," IEICE Spring Conference, B-7-206, Mar.2001 (Letter in Japanese).

APPENDIX

In the case when the cache keeps storing the first k segments of stream i, if a user requested this stream and stopped watching it after the playback of j segments, the probabilistic form of the Segment Hit-ratio is given by the following equation:

$$p_{s\_hit} = \sum_{i,j,k} p(j \le k \,|\, j,i)\, p(j,i) \quad \text{(A-1)}$$

Where $P(j \le k|\, j, i)$ is the conditional probability distribution, or cpdf for short, that means the user is satisfied after watching the playback of total j segments already stored in the cache. $P(j, i)$ is the pdf that a user requests the j segments of stream i. $P(j, i)$ is decomposed into $P(j|i)$ and $P(i)$ and Eq.(1) is rewritten as

$$p_{s\_hit} = \sum_{i,j,k} p(j \le k \,|\, j,i)\, p(j\,|\,i)\, p(i) \quad \text{(A-2)}$$

Let the indices of the removed segments of the streams from the cache be $\Lambda 3$. If the k-th segment of the stream a is removed, $\Lambda 3=\{(k,a)\}$.Besides, the new coming segments are $k+1$ to l segments of the stream m. We denote the set of indices $\{(k+1,m), (k+2,m),..., (l,m)\}$ as $\Lambda 2$.

Then the Segment Hit-ratio after the replacement can be written as follows:

$$p_{s\_hit} = \sum p(j \le k / j,i)\, p(j,i) \quad \text{(A-3)}$$
$$+ \sum p(r \le k / l,m)\, p(r,m)$$
$$- \sum p(s \le k / r,n)\, p(s,n)$$

Where (r,m) belongs to $\Lambda 2$ and (s,n) belongs to $\Lambda 3$. Since only the third term is controllable by the cache server in the above equation, the maximization of $P_{s\text{-}hit}$ is equivalent to the minimization of the third term. Therefore, $\Lambda 3$ is the set of indices of the layers of the images with the lowest pdf $P(s,n)$.

Therefore, the segment of one stream with the lowest $P(j|i)P(i)$ should be removed first. This is the rule of the replacement.

How to efficiently calculate $P(j|i)P(i)$ is also a problem considered by us. As there are numberless segments of different streams, keeping the request records for all segments of all streams is not realizable. Here, we introduce a method to obtain $P(j|i)$ by just keeping the stopping viewing point of stream i within several past requests.

Assume that stream i has been requested by Q times within a fixed period and each time the client stopped watching stream i after the playback of segment $S_{q,i}$ ($q \in (1,Q)$, $i \in (1,M)$, $S \in (1, B_i)$). $P(j|i)$ can be then computed as follows:

$$p(j|i) = \left|\{S_{q,i} \,|\, j \le S_{q,i}\}\right| / Q \quad \text{(A-4)}$$

Where $|.|$ denotes the number of elements of a set.

On the other hand, $P(i)$ is the request frequency of stream i. It can be obtained by calculating the accumulative request times within one testing period. However, as the popularity may change over time, a stream with high access frequency in the last period may not be popular currently. Therefore, how to decide the testing period is very important.

We define T and $T_{c,i}$ respectively as the present time and the time when stream i is cached in the cache. Let v_i denote the request times between $T_{c,i}$ and T. Then, we can get:

$$p(i) = g(i) / (\textstyle\sum_j g(j)) \quad \text{(A-5)}$$

$$g(i) = v_i / (T - T_{c,i}) \quad \text{(A-6)}$$

In the above equation, both access times (v_i) and access period ($T_{c,i}$) are considered. We think it is better than just adopting a constant period to calculate request frequency.

Hence, the priority index of segment can be decided by

$$p(j|i)p(i) = (\left|\{S_{q,i} \,|\, j \le S_{q,i}\}\right| / Q) g(i) / (\textstyle\sum_j (g(j)) \quad \text{(A-7)}$$

As Q and the summation of $g(j)$ over j are identical for all streams in the above Quation, for convenience, we define PIA (priority index of access) as follows:

$$PIA = \left|\{S_{q,i} \,|\, j \le S_{q,i}\}\right| v_i / (T - T_{c,i}) \quad \text{(A-8)}$$