The Least Expectation First (LEF) Media Caching Policy

Kyungwoon Cho† and Hyokyung Bahn††,

† Clunix R&D Center, Clunix Inc., 1-1206, Ace High Tech City, Seoul, 150-972 Korea
†† Department of Computer Science and Engineering, Ewha University, Seoul, 120-750 Korea

Summary
In this article, we present a novel media caching algorithm adequate for real-world streaming workload. Streaming workload is believed to exhibit very large and sequential access characteristics, which is the main concern of a legacy media caching policy. However, a legacy caching algorithm did not fully utilize access pattern and also tends to overlook streaming interactivity which can decrease the advantage of a caching technique utilizing sequential reference. As a replaced cache is determined based on the expected caching gain of the candidates, cache hit ratio can be significantly improved. We develop Least Expectation First technique which can manage very large number of block caches as two-level grouping. Experimental results show that the proposed media caching scheme yields better hit ratio and I/O smoothing than legacy buffer cache replacement schemes.
***Key words*:**
Media Caching Algorithm, Interval Caching, Streaming Server, VCR-like operation.

1. Introduction

In this paper, we focus our efforts on developing the buffer cache management scheme for multimedia streaming servers. Recent advances in a mobile device and communication technology enable users to enjoy on-line media streaming service. Deployment of fourth generation mobile service[1][15] and public broadband wireless further accelerates the proliferation of on-line multimedia service via enabling it on the mobile devices. With this growth in service volume, multimedia server is required to maintain larger and larger amount of data and is required to service increased number of concurrent service sessions. Particular care needs to be taken to elaborately capture the characteristics of the multimedia workload and to incorporate the findings in designing various parts of the system components.

The speed of CPU and the capacity of RAM have been doubling every 18 months for the last couple of decades as indicated by Moore's Law. However, this increase unfortunately has not been accompanied by the increase in the disk bandwidth. Thus, the performance of the application which requires frequent disk access, e.g. On-Line Transaction Processing, On-Line Analytical Processing, Web Server, and Streaming Server, greatly depends on the performance of I/O. It is important to minimize disk access. In this regard, the role of the buffer cache replacement scheme is becoming increasingly important.

Buffer cache management scheme for multimedia workload has been the subjects of numerous works during past several years. Most of these works assume that the streaming workload exhibits sequential access characteristics with bandwidth guarantee. Along with entertainment, education is the emerging area for multimedia application. In distance learning environment where the user accesses the lecture materials remotely, it is possible that the user accesses the particular segment of video repeatedly rather than simply scans the file from beginning to the end. Recently, a few studies examine the user access log obtained from streaming server in service and confirm that there are non-trivial amount of VCR-like operation, e.g. skip, reverse, or etc. This non-sequential access pattern may become more dominant when the user intends to briefly examine the video recordings particularly such as in bandwidth scarce mobile wireless environment.

We carefully believe that buffer cache management schemes which assume that the underlying access pattern is sequential leave much to be desired for mobile streaming environment mainly due to significant fraction of VCR-like operation. In this work, we present the media caching based on the expectation of a block cache. The main idea is that all the media block caches are two-level grouped and the cache block of the group which has the least caching gain is evicted as a replacement victim.

2. Related Works

There have been a lot of works[8][16][5][12][22] on continuous media caching in streaming system and most of caching schemes was based on an interval between two consecutive streams. Interval-based block caching schemes take advantage of sequential access characteristics of continuous media data and maintain information about intervals of consecutive streams accessing the same file. [8] proposed Interval Caching algorithm, which exploits temporal locality between streams accessing the same file by caching intervals between successive streams. The buffer requirement of

Manuscript received October 5, 2011
Manuscript revised October 20, 2011

successive streams is a function of the time-interval between the two streams and the compression method used. [16] presented DISTANCE method. It also caches the blocks in distance of successive clients accessing the same media file. Both algorithms select the victim for replacement with respect to the interval between candidate and its immediate successor. These interval-based replacement schemes are grounded at the assumption that the applications sequentially access the multimedia file. Indeed, interval-based caching schemes show good performance in the totally sequential workload.

The objective of media block caching is to exchange the block I/O bandwidth with main memory space. Won et al.[22] examine the buffer size trade-off between interval caching and disk retrieval and propose an algorithm which minimizes the overall buffer requirement in servicing a set of streams. Some of the works combines the application level caching with the traditional disk retrieval operation for multimedia[6][21].

A number of research results have been released regarding the workload analysis of streaming or educational media server[19][4][17][11]. [17] analyzes the client access to MANIC system audio content. [4][7] analyze access logs of their educational media servers, eTeach and Classroom 2000, respectively. These works deliver insightful information on the usage of the educational media server and the user behavior, e.g. access frequency distribution, file popularity, aging of file access popularity, etc. From these works we can easily see that the data access pattern is more than sequential. For example, [4] showed that for short media files, interactive requests like jump backwards are common.

3. The weakness of Interval-based Caching

In this section, we introduce *interval* which is used as a basic concept in many interval-based media caching algorithms [8][9][16] and supplement the interval idea for our work through a systematic approach. In Fig. 1, the small arrows marked by playback stream S_1, S_2 and S_3 denote the current block positions in the same file and I_{21} represents interval formed by two consecutive streams, S_2 and S_1. Intuitively, interval can be thought of as a group of block caches whose logical file positions are between a preceding and a following stream. In the figure, I_{32} holds block caches b_3 and b_4 which will be referenced by following stream S_2. If the following stream references block which has not been cached in the interval, cache miss is incurred and it should load the block into a cache from storage.

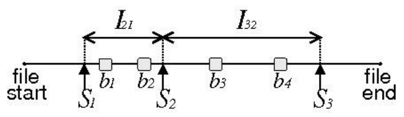

Fig. 1 Interval between two consecutive streams

In order to minimize cache misses, interval-based caching policy manages media block caches based on an interval size, which is a temporal distance between a preceding stream and a following stream. At cache replacement time, a cache in a longer interval is preferable to a cache in a shorter one as a replacement victim. The idea behind the interval caching well utilizes that the cache space of a longer interval is required more than shorter interval. Therefore, block I/O incurred by cache misses will be reduced more with the same cache space by caching media blocks in a shorter interval.

However, interval-based caching has a few drawbacks mentioned in many literatures. First, this caching policy cannot manage the region of a media not covered by an interval such as the head and the tail of the file. But block caches in these areas may have a significant impact on a caching performance. Media workload patterns show that users tend to access the head segment of a media[1].

Second, user interactivities such as VCR-like operations can give a bad effect to the performance of an interval-based caching. When the reference media block of a stream changes by the interactive playback of a user, existing intervals may disappear or a new interval can be introduced. Therefore effective interval management scheme should be devised.

Finally, interval notion cannot fully utilize a media access pattern characterized by a currently serviced stream. Interval-based replacement algorithm selects a victim among block caches in the longest interval believing that they have the least reference probability. But the case that the block cache in the longest interval has more likely to be accessed is possible. In Fig. 2, there are 3 playback streams, S_1, S_2 and S_3. Each small square block represents a media block for stream's continuous playback. Cached media block b_1 will be referenced earlier than block cache b_2 by S_1 but b_2 will be accessed two times by S_2 and S_3. If a replacement victim is being selected between the two block caches, b_1 and b_2, choosing b_2 as a victim cache is best with only 4 look-ahead references. However, evicting b_1 instead of b_2 is more desirable with additional future references. The more future references are utilized, the more caching gain will be achieved.

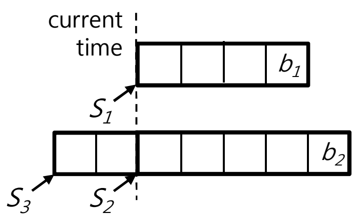

Fig. 2 choosing a replacement victim

Legacy buffer caching algorithms such as interval caching[8][9], DISTANCE[16] and dynamic grouping[20] did not utilize information about a relationship among caches in the same interval. But the caches within even a single interval may have different access probabilities and if so, taking more accurate probability into consideration may help to improve the performance of a media block caching. Also client does not exhibit strictly sequential access pattern and may change the position of the playback. This means that the later a cache is accessed, the less likely it is actually referenced. Therefore optimal cache replacement algorithm should choose the cache which contains the block that would not be accessed for the longest period of time. For example if a playback stream will reference block b_2 after accessing block b_1, and no other streams use b_1 and b_2 in the meantime, choosing b_2 as a replacement victim instead of b_1, until cache block b_1 is referenced, would be a better replacement policy.

But this could lead to large overheads of maintaining an access probability per block cache. To reduce such overheads without losing the benefit of classifying caches according to access probability, *cache set* is devised which is extended from interval caching. An interval is formed only between two consecutive streams but *cache set* may come into being with only preceding stream or following. Besides, there can exist a *cache set* even with neither preceding nor following stream.

4. LEF Caching

4.1 Cache set and cache run

Cache run is a group of consecutive block caches which are locally sorted by the increasing block position in a media file. The head of a cache run is the cache which has the smallest block position in a media file and the tail has the largest one. If any block cache in the middle of a cache run is replaced, the cache run will be split into two smaller ones. All cache runs are included to exactly one *cache set*, so *cache set* means a group of cache runs and they are globally ordered by the smallest position of each owned block caches. The boundary of cache set is determined by the smallest block position and the largest one of holding caches. Every stream has a matching cache set whose head block position is same with its referencing position. If a stream accesses a block cache in the middle of cache set, then it will be split into two smaller cache sets and possibly cache runs will be also split. Therefore the boundary of cache set is determined by a stream and the head position of a cache set progress gradually as a stream accesses sequentially. But there may exist cache set which has no stream such as a cache set on the starting media area.

Fig. 3 shows relationship between cache set and cache run. CS_1 and CS_2 are cache sets and each includes two cache runs. The matching cache set of stream S_1 is CS_1 and S_1 is about to reference b_1, the head of cache run CR_1. Note b_3 and b_9 are not cached. So CR_1 and CR_2, or CR_3 and CR_4 cannot be merged.

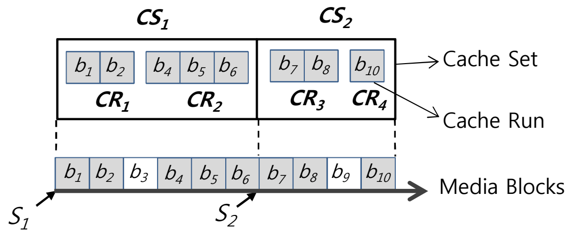
CS_1
CS_2
b_1 b_2 b_4 b_5 b_6 b_7 b_8 b_{10}
CR_1 CR_2 CR_3 CR_4
Cache Set
Cache Run
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10}
Media Blocks
S_1
S_2

Fig. 3 Cache Set and Cache Run

Cache set can be thought of as an interval but it is not the same in several respects. Interval has a preceding stream and a following stream, whereas cache set does not have a preceding stream. To be more exact, cache set is not an interval or a distance between two streams but just a group of block caches as a cache management unit. Moreover, the head cache set at the beginning of a media can have no following stream.

4.2 Least Expectation

To achieve maximum cache hit ratio, caching policy should maintain most accessed block caches. To this end, our method introduces the *expectation* of a block cache. *Expectation* is an expected benefit to improve a cache hit ratio. It depends on how many times each block cache will be accessed within a certain future duration. Even though the position of a stream may change via VCR-like operations, when each block cache will be accessed and what times it will be, can be acquired by tracking the positions of all streams. Let $R(T, b_i)$ be the referenced count of b_i within a duration T. Then the expectation $E(T, b_i)$ of a block cache b_i is

$$E(T, b_i) = \frac{R(T, b_i)}{T} \quad (1)$$

The larger T is used, the more accurate expectation will be obtained. But the cost of finding $R(T, b_i)$ depends on T, so it will be confined to a reasonable extent. Also calculating the expectation of every block cache is impossible with an online algorithm.

Our proposed scheme, *LEF*(*Least Expectation First*) utilizes a expectation per cache set, not block cache. The expectation of cache set is the expectation of the largest block cache among holding block caches. *T* in expectation will be used as double the cache set length, which is the distance from the head to the tail. Let $Pos(b_i)$ denote the temporal position of b_i in a media file, then the cache set size, $Len(CS)$ can be defined as

$$Len(CS) = Pos(b_{tail}) - Pos(b_{head}) \qquad (2)$$

where b_{tail} and b_{head} is each the head block cache and the tail block cache of *CS*. The expectation of cache set *CS* is defined as

$$Exp(CS) = E(2 \cdot Len(CS), b_{tail}) \qquad (3)$$

LEF caching policy orders all cache sets in terms of increasing their expectations and replaces the tail of the cache set with the lowest expectation as a victim cache. The durations of the cache set expectations are different from each other. In viewpoint of LEF, an interval-based caching can be regarded as a policy just using the largest interval size as the expectation duration. LEF utilizes 2 times more knowledge about block reference than interval caching. As the expectation of each cache set is calculated based on its own duration, LEF may commit a misleading replacement that the block cache of a smaller cache set is evicted even though its expectation with same duration may be greater than that of a larger cache set. But such a wrong decision by LEF is always below 50% if the inter-arrival time of a stream obeys Poisson distribution.

Theorem: *If* $Exp(CS_s) < Exp(CS_l)$*, the probability that the tail block cache of* CS_s *is referenced more than reference count of* CS_l *within the double duration of a larger cache set is less than 50%.*

Proof: The case that $Len(CS_s) \geq Len(CS_l)$ is trivial, so $Len(CS_s) < Len(CS_l)$ is assumed. Let b_s and b_l be the tail block caches of CS_s and CS_l each. In order to prove this theorem, show that a following inequation is valid.

$$Prob\left(\frac{R(2 \cdot Len(CS_l), b_s)}{2 \cdot Len(CS_l)} < \frac{R(2 \cdot Len(CS_l), b_l)}{2 \cdot Len(CS_l)}\right) < 0.5 \qquad (4)$$

If a stream arrives at λ rate, the probability that following two equations are valid is over 50%.

$$R(2 \cdot Len(CS_l), b_s) = 2 \cdot Len(CS_l) \cdot \lambda \qquad (5)$$

$$R(2 \cdot Len(CS_s), b_s) = 2 \cdot Len(CS_s) \cdot \lambda \qquad (6)$$

From equation (5) and (6), we get

$$\frac{R(2 \cdot Len(CS_l), b_s)}{Len(CS_l)} = \frac{R(2 \cdot Len(CS_s), b_s)}{Len(CS_s)} \qquad (7)$$

Given equation (7), showing that inequality (8) is always true would be sufficient to prove inequation (4).

$$\frac{R(2 \cdot Len(CS_l), b_s)}{2 \cdot Len(CS_l)} < \frac{R(2 \cdot Len(CS_l), b_l)}{2 \cdot Len(CS_l)} \qquad (8)$$

From a precondition, we can get

$$\frac{R(2 \cdot Len(CS_s), b_s)}{2 \cdot Len(CS_s)} < \frac{R(2 \cdot Len(CS_l), b_l)}{2 \cdot Len(CS_l)} \qquad (9)$$

and by applying equation (7) to inequation (9), we have same result with inequation (8). ■

5. Experiment Result

In this chapter, we present performance analysis of proposed LEF algorithm through the comparison with legacy buffer cache management schemes such as Interval Caching(IC) and LRU. In addition, we also present the performance a modified interval caching(ICC). This scheme collects block caches belonging to the newly created interval from a free block cache list and assigns them to the interval. To facilitate our simulation study, we implemented *ms2sim* simulator[24] conducting various algorithms. We use a fixed size of retrieval block as a unit of I/O and buffer caching. The performance metric we use is cache hit ratio. We examine the effects of varying the size of block cache on cache misses under various caching algorithms. In our experiments, we use the Zipf distribution to generate accesses to 100 media files with a 0.271 skew factor. Also we simulate the session length and the inter-seek time of each stream[3].

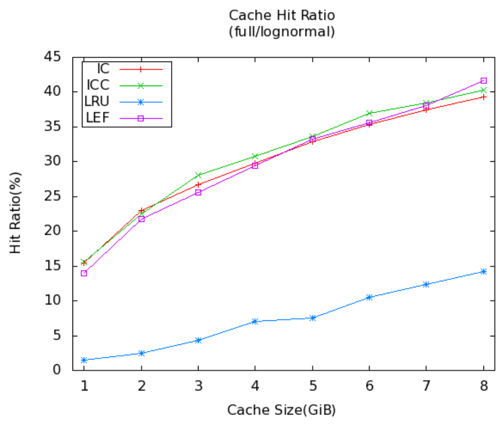

Fig. 4 Hit Ratios with full session length and inter-seek time based on lognormal distribution

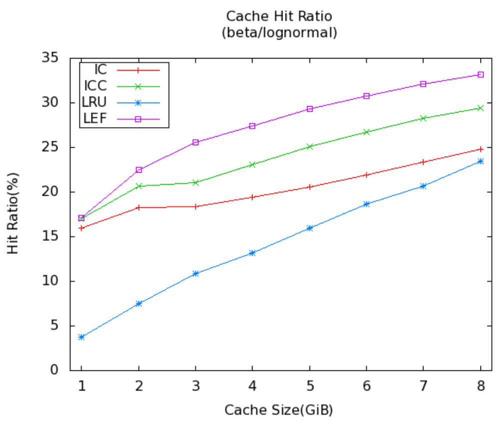

Fig. 5 Hit Ratios with session length based on beta distribution and inter-seek time based on lognormal distribution

Fig. 4 and Fig. 5 illustrate the simulation results for cache hit ratios. When full session length is used, IC, ICC and LEF have shown almost the same performance. However, it is found that when a session length is based on beta distribution, the hit ratios of IC and ICC drop. The hit ratio of LRU is improved compared to the result in full session length.

In Fig. 6 and Fig. 7, the cache space requirements of each caching policy are presented as the number of concurrent stream sessions increases. It is observed that LEF requires much less cache space compared to other policies when full session length is used. But we can see that when a session length obeys beta distribution, LEF is slightly better than LRU. Note IC and ICC require generally more cache space than LEF and LRU even though they have better hit ratio. This means that an interval caching has a media block I/O fluctuated more.

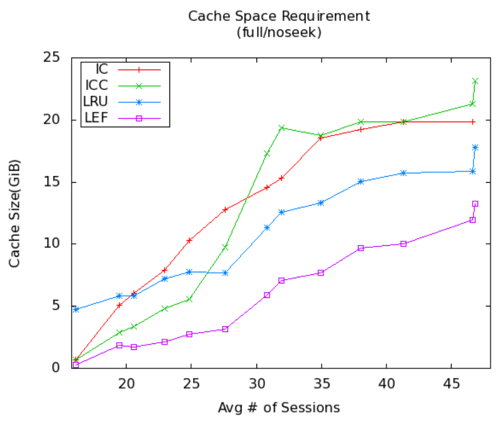

Fig. 6 Required cache size with full session length

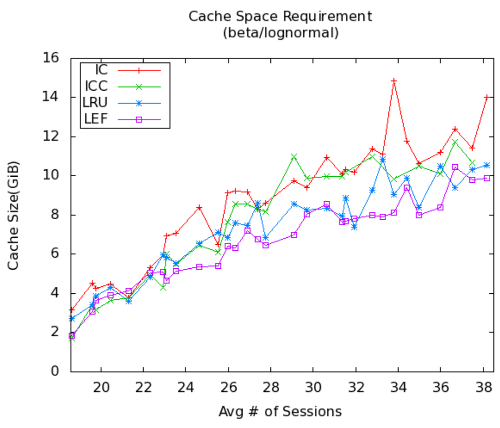

Fig. 7 Required cache size with session length based on beta distribution and inter-seek time based on lognormal distribution

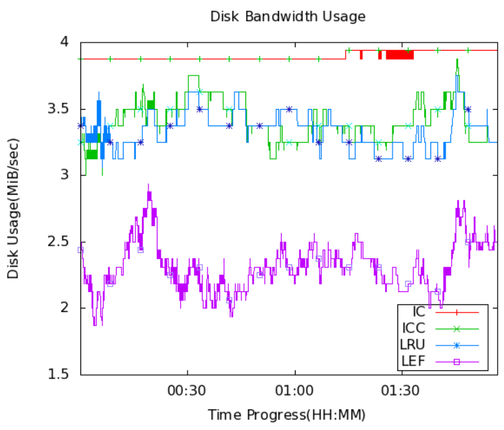

Fig. 8 Disk bandwidth usage trace

In Fig. 8, we trace a disk usage for each caching policy with 4MiB/sec at maximum disk bandwidth. In the figure, the usage of IC is nearly 4MiB, so it cannot support continuous playback. From disk usage trace, we notice that an ICC and LEF have more bandwidth peaks than LRU. But LEF has much lower disk usage than other policies and shows smoothed disk usage.

6. Conclusion

Our proposed media caching policy called *Least Expectation First*(*LEF*) can manage block caches effectively under a real-world workload that a non-sequential access pattern exists. Unlike previous interval caching scheme, LEF keeps orphan block caches in cache runs, which interval caching had put on free cache list when a stream disappears. As all block caches are two-level grouped by cache set and cache run and a block cache is replaced based on its caching gain, LEF can increase cache hit ratio. We showed through simulations that LEF algorithm can perform better and yield more smoothed I/O than well-known legacy algorithms.

Acknowledgement

This work was partially supported by the research fund of Electronics and Telecommunications Research Institute (ETRI).

References

[1] Tomazic, S. & Jakus, G. Long term evolution: Towards 4th generation of mobile telephony and beyond Proc. 9th Int. Conf. Telecommunication in Modern Satellite, Cable, and Broadcasting Services TELSIKS '09, 2009, 91-96

[2] Yu, H.; Zheng, D.; Zhao, B. Y. & Zheng, W. Understanding user behavior in large-scale video-on-demand systems SIGOPS Oper. Syst. Rev., ACM, 2006, 40, 333-344

[3] Brampton, A.; MacQuire, A.; Fry, M.; Rai, I.; Race, N. & Mathy, L. Characterising and exploiting workloads of highly interactive video-on-demand Multimedia Systems, Springer Berlin / Heidelberg, 2009, 15, 3-17

[4] Jussara M. Aimeida, Jeffrey Krueger, Derek L. Eager, and Mary K. Vernon. Analysis of educational media server workloads. In Proceedings of International Workshop on Network and Operating System Support for Digital Audio and Video, Port Jefferson, NY, USA, June 2001.

[5] Matthew Andrews and Kameshwar Munagala. Online algorithms for caching multimedia streams. In European Symposium on Algorithms, pages 64–75, 2000.

[6] Bradshaw, M. K.; Wang, B.; Sen, S.; Gao, L.; Kurose, J.; Shenoy, P. & Towsley, D. Periodic broadcast and patching services - implementation, measurement and analysis in an internet streaming video testbed Multimedia Systems, Springer Berlin / Heidelberg, 2003, 9, 78-93.

[7] M. Chesire, A. Wolman, G. Voelker, and H. Levy. Measurement and analysis of a streaming media workload. In Proceedings of 3rd USENIX Symp. on Internet Technologies and Systems, San Francisco, CA, USA, March 2001.

[8] A. Dan, Y. Heights, and D. Sitaram. Generalized interval caching policy for mixed interactive and long video workloads. In Proc. of SPIE's Conf. on Multimedia Computing and Networking, 1996.

[9] A. Dan and D. Sitaram. Buffer management policy for a on-demand video server. Technical Report RC 19347, IBM.

[10] Kevin W. Froese and Richard B. Bunt. Cache management for mobile file service. The Computer Journal, 42(6):442–454, 1999.

[11] N. Harel, V. Vellanki, A. Chervenak, G. Abowd, and U. Ramachandran. Workload of a media-enhanced classroom server. In Proceedings of IEEE Workshop on Workload Characterization, Oct. 1999.

[12] M. Hofmann, E. Ng, K. Guo, S. Paul, and H. Zhang. Caching techniques for streaming multimedia over the internet. Technical Report BL011345-990409-04TM, Bell Laboratories, 1999.

[13] Rainer Koster and Thorsten Kramp. Structuring qoS-supporting services with smart proxies. In Proceedings of the IFIP/ACM Middleware Conference (Middlware), volume 1795, Berlin, Heidelberg, New York, Tokyo, 2000. Springer-Verlag.

[14] Geoffrey H. Kuenning, Gerald J. Popek, and Peter L. Reiher. An analysis of trace data for predictive file caching in mobile computing. In USENIX Summer, pages 291–303, 1994.

[15] Nobuo Nakajima. The path to 4g mobile. IEEE Communications, 39(3):38–41, March 2001.

[16] Banu Ozden, Rajeev Rastogi, and Abraham Silberschatz. Buffer replacement algorithms for multimedia storage systems. In International Conference on Multimedia Computing and Systems, pages 172–180, 1996

[17] J. Padhye and J. Kurose. An empirical study of client interactions with a continuous-media courseware server. In Proceedings of International Workshop on Network and Operating System Support for Digital Audio and Video, July 1998.

[18] R. Rejaie, M. Handley, H. Yu, and D. Estrin. Proxy caching mechanism for multimedia playback streams in the internet, 1999.

[19] Lawrence A. Rowe, Diane Harley, and Peter Pletcher. Bibs: A lecture webcasting system. Technical report, Berkeley Multimedia Research Center, UC Berkeley, June 2001.

[20] S. Sheu, K. Hua, and W. Tavanapong. Dynamic grouping: An efficient buffer management scheme for video-on-demand servers. Technical Report CSTR-97-02, University of Central Florida, Orlando, Florida, Feb 1997.

[21] Shi, W. & Ghandeharizadeh, S. Trading memory for disk bandwidth in video-on-demand servers Proceedings of the 1998 ACM symposium on Applied Computing, ACM, 1998, 505-512

[22] Youjip Won and Jaideep Srivastava. "smdp: Minimizing buffer requirements for continuous media servers". ACM/Springer Multimedia Systems Journal, 8(2):pp. 105–117, 2000.

[23] Kun-Lung Wu, Philip S. Yu, and Joel L. Wolf. Segment-based proxy caching of multimedia streams. In World Wide Web, pages 36–44, 2001.

[24] https://code.google.com/p/ms2sim/

Kyungwoon Cho received the BS and MS degrees in computer science from Seoul National University, Korea, in 1995 and 1997, respectively. He is currently a chief officer in the Clunix R&D Center, Seoul, Korea. His research interests include multimedia systems and operating systems.

Hyokyung Bahn received the BS, MS, and PhD degrees in computer science from Seoul National University, Korea, in 1997, 1999, and 2002, respectively. He is currently an associate professor in the department of computer science and engineering, Ewha University, Seoul, Korea. His research interests include operating systems, caching algorithms, Web technologies, reference behavior modeling, genetic algorithms, and distributed systems. Dr. Bahn is a member of the IEEE Computer Society, the IEICE, and the Korea Information Science Society.