ABM: Looping Reference-Aware Cache Management Scheme for Media-on-Demand Server

K.W. Cho[1], Y.S. Ryu[2], Youjip Won[3], and Kern Koh[1]

[1] School of Computer Science & Engineering, Seoul National University, Korea
[2] Dept. of Computer Science, Hallym University, Korea
[3] Division of Electrical and Computer Engineering, Hanyang University, Korea

Abstract. Legacy buffer cache management schemes for multimedia server are grounded at the assumption that the application sequentially accesses the multimedia file. However, user access pattern may not be sequential in some circumstances, for example, in distance learning application, where the user may exploit the VCR-like function(rewind and play) of the system and accesses the particular segments of video repeatedly in the middle of sequential playback. Such a looping reference can cause a significant performance degradation of interval-based caching algorithms. And thus an appropriate buffer cache management scheme is required in order to deliver desirable performance even under the workload that exhibits looping reference behavior. We propose Adaptive Buffer cache Management(ABM) scheme which intelligently adapts to the file access characteristics. For each opened file, ABM applies either the LRU replacement or the interval-based caching depending on the *Looping Reference Indicator*, which indicates that how strong temporally localized access pattern is. According to our experiment, ABM exhibits better buffer cache miss ratio than interval-based caching or LRU, especially when the workload exhibits not only sequential but also looping reference property.

Keywords: Buffer Cache, Multimedia, File System, Interval Caching, LRU, Looping reference, ABM

1 Introduction

1.1 Motivation

In this paper, we focus our efforts on developing the buffer cache management scheme for multimedia streaming server. Recent advances in computer and communication technology enable the user to enjoy on-line multimedia data service anytime and anywhere. Deployment of third generation wireless service[12] further accelerates the proliferation of on-line multimedia service. With this growth in service volume, multimedia server is required to maintain larger and larger amount of data and is required to service more 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 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 avoid any disk accesses if possible and subsequently the role of the buffer cache replacement scheme is becoming increasingly important.

We argue that multimedia workload may exhibit the characteristics other than sequential access pattern. Further, if the access pattern is not sequential, the legacy interval-based caching strategy may not work properly. We carefully believe that non-trivial fraction of streaming workload actually does belong to this category, i.e. the one which does not exhibit sequential access pattern. 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.

In this paper, we propose novel buffer cache management scheme algorithm referred to as *Adaptive Buffer cache Management (ABM).* ABM periodically monitors the workload characteristics and system behavior and dynamically switches between interval-based caching or LRU replacement scheme. It intelligently applies an appropriate policy per-file basis.

1.2 Related Works

There have been a lot of works on the buffer management in continuous media file system [8,7,14,2,10]. *Interval caching policy* was proposed in [8] which caches intervals formed by pairs of consecutive streams accessing to the same movie object. This idea was extended in order to support caching short video clips as well as large video objects [7]. Özden et al. [14] presented two buffer replacement algorithms — BASIC and DISTANCE. DISTANCE scheme is an approximation of maintaining free buffers in separate MRU lists per client and replaces the caches in an order which depend on the distance of clients accessing the same media file. Recently, there have been several works on buffer cache management algorithm for streaming server for Internet environment. Hofmann et al. [10] proposes a solution for caching multimedia streams by integrating segmentation of streaming objects, dynamic caching and self-organizing cooperative caching. Matthew et al. [2] shows analytically that the interval caching scheme is optimal for caching multimedia streams in the Internet and that the maximum number of simultaneous cache misses is a more important factor rather than the total number of cache misses. Also buffer management schemes based on user level hints such as application controlled file caching [3] and informed prefetching and caching [16] have been proposed. Recently adaptive buffer management scheme is presented in [5] which automatically detects the block reference patterns of applications and applies different replacement policies to different applications based on the detected reference pattern.

Most existing buffer management policies mentioned above exploit only sequential file access pattern not considering looping references. The idea proposed in our work bears some similarity with the one which is recently proposed by Smaragdakis et al. [19]. They proposed *Early Eviction LRU* algorithm which evicts either least recently used page or *relatively* recently used pages depending on the system states. In EELRU algorithm, they maintain the information about the recently evicted pages. If it detects that larger fraction of recently fetched pages are evicted, it applies early eviction strategy instead of applying LRU. This algorithm behaves particularly well when there are large loops in the workload.

Recently, a number of research results have been released regarding the workload analysis of streaming and/or educational media server[17,1,4,15,9]. [15] analyzes the client access to MANIC system audio content. [1,4] analyze access logs of their educational media servers, eTeach and Classroom 2000, respectively. Rowe et al. [17] addressed that students access the video clips to review the materials they were not able to understand properly during the class. 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. Unfortunately, these works do not effectively address the access characteristics in small time scale, e.g. frame level or block offset. However, we carefully believe that a certain segment of the video clips can be accessed repeatedly by the *same* user exhibiting looping reference pattern.

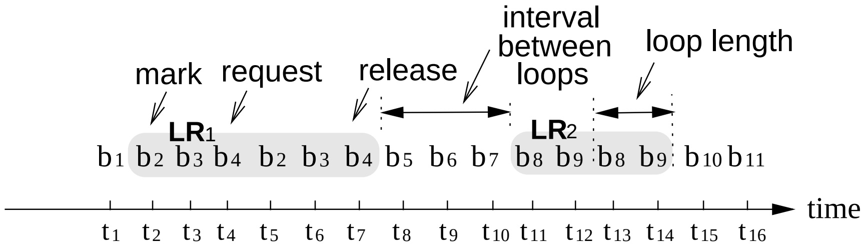
The remainder of this paper is organized as follows. Section 2 describes our modeling approach for multimedia workload. Section 3 provides the explanation of a interval-based replacement scheme and possible anomalies under looping reference workload. In section 4, we describe the proposed buffer caching algorithms. Section 5 presents performance results and validates our algorithm and finally section 6 summarizes our results.

2 Workload Modeling in Continuous Media Server

Data reference patterns can be categorized into four different categories: (i) sequential, (ii) looping, (iii) temporally localized and (iv) probabilistic reference. In *sequential* reference pattern, the application sequentially scans the file and once the data block has been accessed, it is not accessed again. In *looping reference*, application accesses a set of consecutive data blocks repeatedly. *Most Recently Used (MRU)* buffer cache replacement is usually preferred in sequential workload. It is known that *Least Recently Used (LRU)* replacement scheme is appropriate for handling temporally localized access pattern.

Multimedia technology is being applied in various fields, e.g. entertainment, education, medical study, tele-collaboration to list a few. Among them, entertainment and education are typical fields where video streaming technology proliferates at a tremendous rate. In entertainment arena, typical video clip is *news*, *movie clips*, etc. where data block access pattern is sequential. It is well known that *LRU* does not behave well under sequential workload. Under sequential access pattern, it was shown that interval-based caching delivers better cache performance than the legacy LRU based buffer cache replacement scheme[7,14, 2].

However, this commonly accepted assumption that data access pattern is sequential in streaming environment may not hold in a certain situation and we may overlook its implication on the system performance. In on-line education field, the user watches lecture on-line with the lecture materials and instructor's annotation appears on the same screen synchronized what the speaker is saying. We carefully suspect that the users may exhibit *more than* sequential behavior in this environment. According to the report by Rowe et al. [17], students use education media server mostly to review what instructor has said about a particular topic since the students have difficulty in understanding it in the class. In this situations, it is possible that the user accesses particular segment of the video repeatedly than sequentially scans the file from the beginning to the end.



interval between loops
loop length
mark
request
release
LR1
LR2
b1 b2 b3 b4 b2 b3 b4 b5 b6 b7 b8 b9 b8 b9 b10 b11
time
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16

Fig. 1. Looping Reference

When the workload bears looping reference characteristics, we need a systematic way of representing or characterizing the workload and the respective *degree of looping reference*. We introduce three metrics for this purpose: *loop length*, *loop count*, and *interval between loops*. *Loop length* is the number of data blocks within single scan of the loop. *Loop count* is the number of iterations. *Interval between loops* are the distance between the consecutive loops in terms of number of blocks. It would be better to explain each of these attributes via example. Fig. 1 illustrates the user reference pattern which may be generated by exploiting the VCR-like function or another designated facility for iterative playback. b_i denotes the i^{th} referenced data block. Consecutive data blocks $b_2b_3b_4$ and b_8b_9 are viewed repeatedly. When b_2 is referenced at t_2, that block is marked as the beginning position of the loop and then the offset of the playback will be adjusted in order to access blocks $b_2b_3b_4$ repeatedly just after loop request is issued at t_4. Block access patterns during two intervals – from t_2 to t_7 and from t_{11} to t_{14} build *looping references*, which may be represented such as $(b_k b_{k+1} \dots b_l)^m$ that *loop length* and *loop count* of the *looping reference* are $l - k + 1$ and m respectively. Sequential playback can be resumed by releasing *looping references*. In addition to these intra-loop attributes, logical distance between LR_1 and LR_2 can specify inter-loop relationships and will be defined as *interval between loops(IBL)*. The workload concerned in this paper is sequential user access pattern mixed with *looping references*. Such a model can be characterized with three loop parameters(*loop length*, *loop count*, *IBL*). In Fig. 1, *Loop length* and *loop count* of the *looping reference* LR_1 are 3 and 2. And *IBL* formed by two *looping references* LR_1 and LR_2 is 3.

3 Looping Reference

3.1 Interval Based Replacement vs. LRU Scheme

Interval-based scheme[8,7,14] maintains information about intervals of consecutive playbacks accessing the same file. In order to maximize the number of playbacks accessing data from buffer cache, interval-based schemes sort intervals with an increasing order and cache data blocks from the shortest intervals. When data blocks are accessed sequentially and playback rates are same, the event when a playback is newly launched or terminated can change the intervals formed between neighboring playbacks. However, if there exist looping references, intervals may be changed, merged, splitted, created and/or removed when loop request is issued.

Fig. 2 illustrates data block access pattern for individual sessions. S_2 exhibits looping reference. Let $I_{i,j}$ be the distance between S_i and S_j in terms of the data blocks. During the first phase, P_1, the distance between S_1 and S_2, i.e. $I_{1,2}$ are 2 and $I_{2,3}$ is 6. Thus, to minimize the disk bandwidth utilization with minimum overhead, the system maintains the data blocks used by S_1 so that they can be used by S_2 later. When the data blocks are viewed by S_2, they can be replaced out from the buffer cache. During the second phase, P_2, S_2 views the block 10 to block 12 repeatedly(looping reference) and thus $I_{1,2}$ and $I_{2,3}$ changes to 5 and 3, respectively. During P_2, it is better to maintain the data blocks viewed by S_2 rather than the data blocks used by S_1. This is because the playback distance between S_1 and S_2 becomes 5 while the playback distance between S_2 and S_3 is 3. After intervals changes due to occurrence of loop, the interval-based cache management scheme is going to change contents of cache space gradually. That is, when new block needs to read into the buffer cache, the interval-based scheme must determine which blocks to cache and which blocks to replace using the newly created (or modified) interval sets.

Depending on the time scale of interest, looping reference pattern can be thought as temporally localized workload or sequential workload. The time scale of interest depends on the buffer cache size. If the buffer cache is large enough to hold the entire data blocks in the loop, we can think that the workload exhibits temporal locality. Otherwise, it is regarded as sequential workload. The LRU policy is known optimal in temporal localized reference pattern [6].

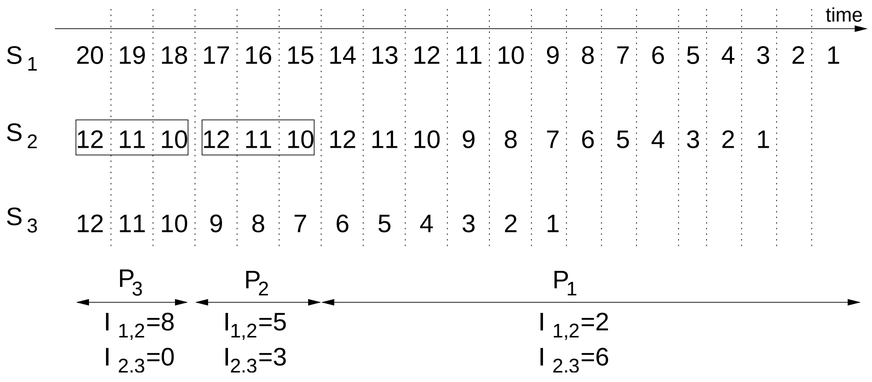
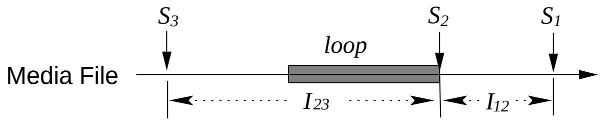


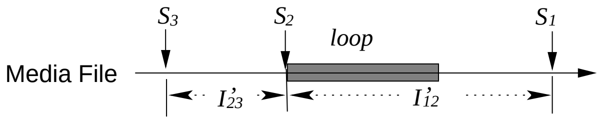
Fig. 2. Interval based replacement scheme with looping reference

3.2 Interval-Based Caching Anomaly

Fig. 3 illustrates the situation that intervals between the playbacks dynamically changes due to *looping reference*. From the aspect of interval-based replacement algorithms, data blocks belonging to I_{12} have higher priorities than those belonging to I_{23}. After S_2 goes backward to the beginning point of the loop, the playback distances between S_1 and S_2 and between S_2 and S_3 changes to I'_{12} and I'_{23}. More interesting thing is shown in Fig. 4. Looping reference may reverse the orders of streams so that S_2 follows S_3 after loop request was issued by S_2.

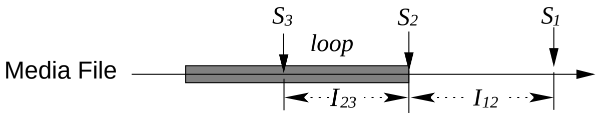


(a) Before S_2 issues loop request

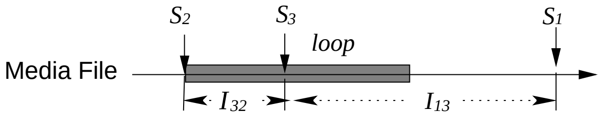


(b) I_{12},I_{23} are modified

Fig. 3. Intervals are changed due to looping reference



(a) Before S_2 issues loop request



(b) I_{12},I_{23} are merged into I_{13} and I_{32} is created

Fig. 4. Intervals are merged and a new interval is created

Interval-based buffer cache replacement scheme mainly exploits the sequentiality of data reference pattern. Buffer cache replacement scheme first estimates the future access probability of the data blocks and performs replacement based on it. In interval-based buffer cache replacement scheme, the future access probability is estimated by the logical distance to the most recently used preceding block in the same file. This way of computing the future access probability does not perform well when the playback distance dynamically changes especially via looping reference. Let us visit Fig. 3 again. Comparing Fig. 3(a) and Fig. 3(b), I_{23} becomes smaller than I_{12} in Fig. 3(b). Thus, at some point between the transition from Fig. 3(a) to Fig. 3(b), it become better to select the data block in I_{12} than in I_{23} as a victim for replacement. Given all these characteristics of looping reference and interval caching, we carefully conjecture the interval-based caching for multimedia streaming may not work properly when the application exhibits *looping reference*. To verify our conjecture, we examine the buffer cache miss ratio under various different legacy buffer cache replacement algorithms: OPT, DISTANCE[14], LRU, LRU-k[13] and MRU. Throughout simulations in Section 5, DISTANCE exhibits the lowest miss ratio while LRU and MRU miss ratios is larger than 90% with sequential reference only. Interestingly, however, when the workload bears some degree of *looping reference* characteristics, LRU is better than DISTANCE. This result confirms our conjecture that *looping reference* may be handled properly using LRU in some cases. On the other hand, if the buffer cache is large enough to hold the entire data blocks referenced in the looping area, workload can be said to exhibit temporal locality characteristics. Henceforth, LRU based scheme may be the right choice in this situation.

Modern server system is required to handle a number of concurrent sessions. Even though the type of requested data may be homogenous, e.g. video file, individual users may exhibit widely different access characteristics which is either *sequential* or *looping*. In this article, we propose novel buffer cache management algorithm which effectively incorporate the various access characteristics in single framework and selects the right victim for replacement.

4 Adaptive Buffer Management (ABM)

4.1 Looping Reference Indicator

In this work, we propose a metric called *Looping Reference Indicator*, δ, to denote whether there exists *looping reference* for a given file and how strong it is. δ is maintained for each opened file.

Let s and $S_t(i)$ be the user id and the set of users who accesses the file i at time t. Let $|S_t(i)|$ be the number of members in $S_t(i)$. Let $N_i(R_t(s))$ be the logical block number of file i which is accessed by the user s at t. Let $SR_t(i)$ be the set of users in $S_t(i)$ who sequentially accesses the file i at t. That is, $SR_t(i) = \{ s \mid N_i(R_t(s)) = N_i(R_{t-1}(s)) + 1 \text{ and } s \in S_t(i) \}$. And the set of users who do not sequentially access the file i at t is denoted by $SR_t(i)^c$. Let $B_t(i)$ be the number of data blocks of file i in the buffer cache at time t. When s is accessing the file in looping reference at t, let $L_t(s)$ denote the *loop length* in terms of the number of data blocks. If the looping reference pattern is represented as $(b_k b_{k+1} \ldots b_l)^m$, the *loop length* corresponds to $l - k + 1$. When the loop request is issued, it is called as *effective* if the number of buffer cache blocks allocated to user s is greater than the *loop length*, $L_t(s)$. Otherwise, that loop request is regarded as *non-effective*.

$$ER_t(i) = \{s|\ L_t(s) \leq \frac{B_t(i)}{|S_t(i)|}\ \textit{and}\ s \in SR_t(i)^c\} \tag{1}$$

$$NER_t(i) = \{s|\ L_t(s) > \frac{B_t(i)}{|S_t(i)|}\ \textit{and}\ s \in SR_t(i)^c\} \tag{2}$$

If it is possible to maintain data blocks of the loop area in the buffer cache, LRU scheme can achieve higher cache hit rate. Finally, $ER_t(i)$ is a set of users in $S_t(i)$ whose loop request is effective at t and their subsequent I/O requests can be most likely serviced from the buffer cache.

Given all these, we can define the *looping reference indicator(LRI)* $\delta_t(i)$ as in Eq. 3. θ in the equation is the *update window* for δ. *LRI* is calculated with only past θ samples to limit overheads to maintain $ER_t(i)$, $NER_t(i)$ and $SR_t(i)$. But too small θ may cause *LRI* too sensitive to workload fluctuation.

$$\delta_t(i) = \frac{\sum_{t-\theta}^{t} |ER_t(i)|}{\sum_{t-\theta}^{t} |ER_t(i)| + \sum_{t-\theta}^{t} |NER_t(i)| + \sum_{t-\theta}^{t} |SR_t(i)|} \tag{3}$$

Large $\delta_t(i)$ means that there are many effective loop requests and thus blocks in loop area can be served from buffer cache if temporal locality is exploited such as *LRU*. On the other hand, smaller *LRI* implies that relatively larger fraction of the users are accessing the file in sequential fashion or non-effective loop requests are often occurred. The objective is to determine which of the buffer cache management algorithm is to be used for file i: *DISTANCE* or *LRU*. We use the threshold value δ^* as a selection criteria for buffer cache management algorithm. If the *LRI* of a file is smaller than δ^*, DISTANCE is applied to that file. Otherwise, LRU scheme is applied.

4.2 Buffer Management Method

ABM manages two separate buffer pools and different replacement policies are applied to each pool: LRU pool and DISTANCE pool. Assigning a cache block of each file to LRU pool or DISTANCE pool is determined based on per-file reference characteristic. *Looping reference indicator*, δ of the each accessed file is maintained and appropriate buffer pool is selected with comparison between δ and δ^*. If δ of a file is greater than or equal to δ^* which means that there are sufficient effective loop requests, whole buffers of the file belong to LRU pool. Otherwise, they become parts of DISTANCE pool. When the file access behavior changes according to workload pattern change such as effective loop request is increased, buffers of the file may be transferred to other buffer pool.

As buffer replacement is required, ABM firstly checks per-stream amount of the each buffer pool. Selected buffer pool which has higher buffer usage should choose a victim buffer in its pool, which is is determined by its own replacement policy. The number of buffers allocated to each pool has highly impacted the cache performance. The basic idea of the proposed buffer allocation to each pool is to evenly allocate the data blocks to individual streaming sessions.

5 Simulation Results

In this section, we present the results from simulation of existing buffer replacement schemes, such as DISTANCE, LRU, LRU-k and MRU, and also show the effectiveness of the proposed ABM scheme. In our experiments, synthetic data reference pattern of clients is used because it is hard to get real workload traces.

In all the experiments, clients arrive randomly. Inter-arrival times of clients are exponentially distributed and loop parameters are generated from gaussian distribution. The number of media files is 20 and each file has a length of 3600 blocks. Every stream consumes one block per service round with same playback rate. The performance metric considered is the cache miss ratio and is measured through 43200 service rounds simulation.

5.1 Comparison of Legacy Algorithms

We examine the performance of existing schemes such as DISTANCE, LRU, LRU-k and MRU when clients generate the looping access pattern. In these experiments, LRU-k takes into account knowledge of the last two references. We vary the average *IBL* of each client to assess the effect of frequency of looping accesses. System parameters used in this simulation are listed in Table 1.

Table 1. System Parameters used in IBL performance comparison

System Parameter	**Value**
Cache size	6000 blocks
Interval between loops	No looping references, 1000, 500, 200, 100
Loop length	40 blocks
Loop count	5

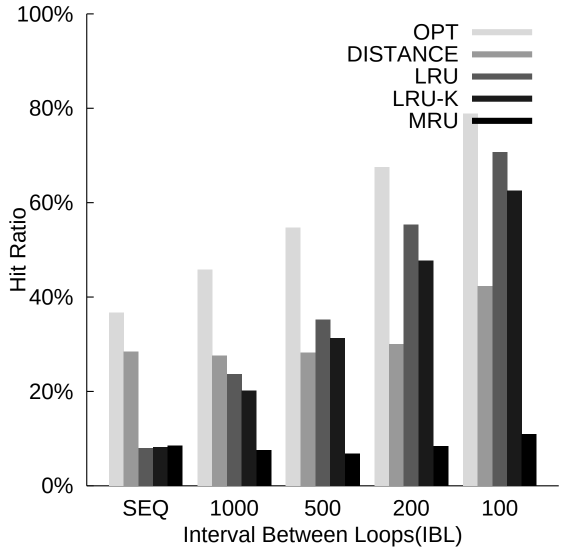


Fig. 5. Effects of varying IBL on the cache hits

Fig. 5 illustrates the effects of varying *IBL* on the cache hits under different buffer replacement schemes. OPT in the figure just gives explanatory information for optimal algorithm. With only sequential reference pattern, LRU and MRU yield much lower hit ratio while DISTANCE gets close to optimal buffer replacement. DISTANCE's hit rate is much higher than other policies as the cache size increases. Hence, DISTANCE is a suitable candidate for buffer replacement scheme under only sequential reference pattern in multimedia data retrieval. When sequential and looping references co-exist, LRU's hit rate is highly dependent on *IBL*. The shorter *IBL* is, the higher the hit ratio of LRU is. When *IBL* is long, for example *IBL* is 500 and 1000 in Fig. 5, the hit ratio of LRU is yet lower than that of DISTANCE. But, when *IBL* is 100 and *loop length* is 40, LRU shows higher hit ratio at about 30% compared to DISTANCE and almost reaches OPT. From this figure, LRU policy is best among other replacement algorithms for looping reference pattern in continuous media streams. Hit ratios of DISTANCE and MRU tend to remain regularily regardless of *IBL*. On the contrary, performances of the LRU and LRU-k are largely concerned with *IBL*.

Fig. 6 illustrates the effects of varying the *loop length* on the cache hit ratio. The *loop length* is changed from 20 blocks to 100 blocks. Detailed system parameters used in this simulation are described in Table 2. The figure shows that loop length has little impact on the hit ratios of MRU and DISTANCE, but has much effects on LRU and LRU-K. This is because the longer *loop length* incurs more sequential references and buffer space to hold data blocks in loop area exceeds the number of the buffer. Shorter *loop length* such as 20 may lead to under-utilization of buffers.

Table 2. System Parameters used in loop length comparison

System Parameter	**Value**
Cache size	6000 blocks
Interval between loops(IBL)	100
Loop length	20, 40, 60, 80, 100 blocks
Loop count	5

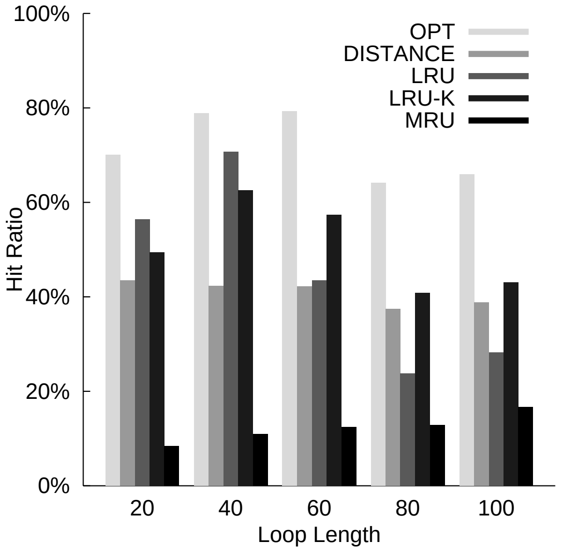


Fig. 6. Effects of varying the loop length on the cache hits

Both Fig. 5 and Fig. 6 also show that the size of cache has much impact on LRU. Given a *loop length*, LRU has better hit rate if the size of cache is so large that we can have more effective loop requests. That is, when the cache space is sufficient to hold the blocks in loop, LRU's hit rate can be decreased dramatically.

Fig. 7 represents hit ratios with varying the *loop count* such as 3, 5, 7, 9 and 11 times. As the *loop count* increases, hit ratios of the all policies except MRU get higher. But DISTANCE ratio grows a little smoothly as compared to LRU and LRU-K. LRU algorithm shows best hit ratio in all experiments with various *loop counts.*

Table 3. System Parameters used in loop count comparison

System Parameter	**Value**
Cache size	6000 blocks
Average interval between loops(IBL)	100
Loop length	20 blocks
Loop count	3, 5, 7, 9, 11

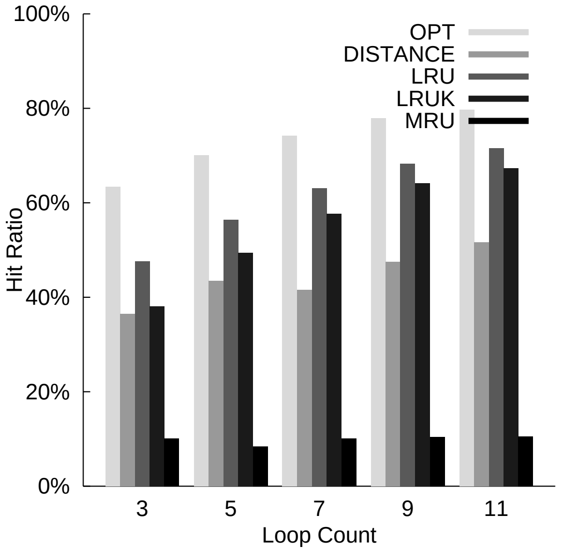


Fig. 7. Effects of varying the loop count on the cache misses

5.2 Performance of ABM

In order to investigate the performance of ABM, we need to define looping reference pattern of each file differently. We use 20 files and assign them integer number from 1 to 20. And we define four *Loop Distribution Type (LDTs)* described in Table 4. In the Table 4, $\mathbf{IBL}(i)$ is the average *IBL* of clients accessing file i. In all LDTs, the average *loop length* and the *loop count* are fixed at 20 blocks and 5 times, respectively.

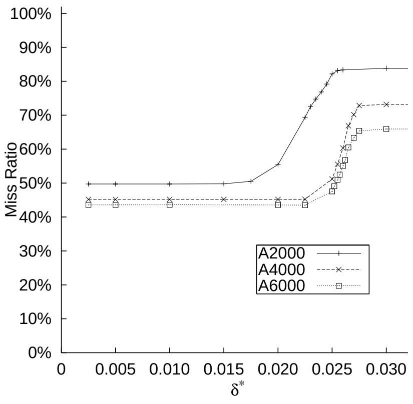
Table 4. Description of LDTs used in experiments

LDT	**Description**
LDT1	$\mathbf{IBL}(i) = 100$, for all file $(1 \le i \le 20)$
LDT2	$\mathbf{IBL}(i) = 1000$, for all file $(1 \le i \le 20)$
LDT3	$\mathbf{IBL}(1) = 50$ $\mathbf{IBL}(i) = \mathbf{IBL}(1) * 1.1^{(i-1)}$
LDT4	Let N be the number of files and $L(i)$ be the length of file i in terms of the number of blocks, respectively. $\mathbf{IBL}(1) = 50$ and $L(i) = 3600$ If $i \le \frac{N}{2}$, $\mathbf{IBL}(i) = \mathbf{IBL}(1) * 1.1^{(i-1)}$ Otherwise, $\mathbf{IBL}(i) = \frac{L(i)}{1.1^{(N-i)}}$

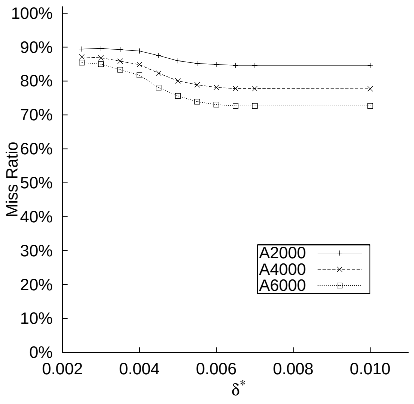
In LDT1, all files has many looping references. Note that if *IBL* is 100, LRU performs better than DISTANCE. Fig. 8(a) summarizes the result of experiment with LDT1. This figure shows that the miss rate of ABM moves between LRU and DISTANCE with varying the threshold of *looping reference indicator*, δ^*. Consider the size of cache is 4000. If δ^* is smaller than 0.02, ABM applies LRU policy to all files. Hence, the miss ratio of ABM equals that of LRU. If δ^* is greater than 0.03, ABM applies DISTANCE policy to all files and thus, the miss ratio of ABM equals that of DISTANCE. It is also noted that the miss rate of ABM changes rapidly when δ^* is changing between 0.02 and 0.03. This phenomenon appears in other buffer sizes.

Fig. 8(b) shows the results of experiment with LDT2. In LDT2, $\mathbf{IBL}(i)$ for all file is set to 1000. Hence, clients access files in sequential fashion and sometimes access some blocks repeatedly. In this case, ABM should apply DISTANCE policy to all files because DISTANCE outperforms LRU as shown in Fig. 5.

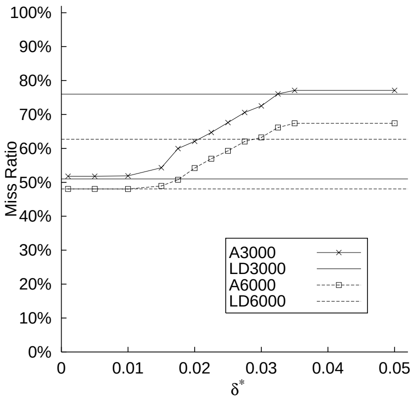
In LDT3, we assign $\mathbf{IBL}(i)$ to file i decreasing value as file number i increases. In this case, ABM applies LRU to files whose *looping reference indicator* is greater than threshold value δ^* and DISTANCE to files whose *looping reference indicator* is smaller than δ^*. Fig. 8(c) shows the result of LDT3. In this figure and 8(d), LD3000 and LD6000 in legend means the miss ratio of better policy between LRU and DISTANCE. Consider the size of cache is 6000. ABM can slightly outperform than both LRU and DISTANCE if it uses 0.01 as δ^*.



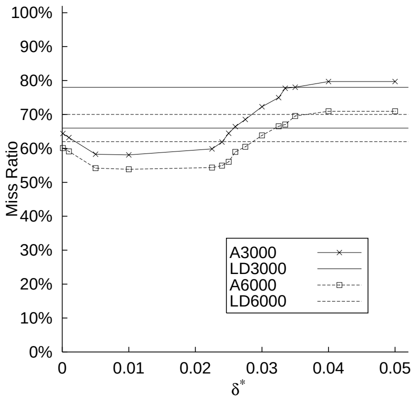
(a) Miss ratios of ABM with LDT1



(b) Miss ratios of ABM with LDT2



(c) Miss ratios of ABM with LDT3



(d) Miss ratios of ABM with LDT4

Fig. 8. Miss Ratios of ABM

Finally, we investigate the case when LDT4 is used. In LDT4, media files are partitioned into two groups. Files in the first group are assigned small *IBL* and second group files are assigned large *IBL*. In this case, ABM tries to apply LRU policy to the first group's files and DISTANCE policy to the second group's files. ABM performs better than both LRU and DISTANCE when δ^* is 0.01 or 0.02.

5.3 Adjusting δ^* Value

In practical system, we need for a mechanism that dynamically adjusts δ^* according to the workload and a given system configuration to get the best performance results. To address this issue we propose a method adjusting δ^* periodically depending on whether the hit rate has improved during the last θ period.[1] For example, if the hit rate of period p is better than that of period p-1 and the δ^* value for period p is larger than the δ^* value for period p-1, the δ^* value is incremented. On the other hand, if the hit rate of period p is worse than that of period p-1 and the δ^* value for period p is larger than the δ^* value for period p-1, the δ^* value is decremented.

6 Conclusion

We observed that LRU policy yields lower cache miss rate than DISTANCE policy when the workload exhibits looping reference pattern. This brought us some evidence that legacy interval-based caching schemes like DISTANCE algorithm may not work properly when the workload carries looping behavior. On the other hand, since LRU selects victim based on the interval from the last usage of the block, it may not exploit the sequential characteristics of the workload. The server is required to handle a number of concurrent sessions whose behaviors are widely different. Neither DISTANCE nor LRU policy alone is not able to deliver desirable system performance.

In this work, we develop the novel buffer cache replacement algorithm which effectively incorporates the dynamically changing workload characteristics and which adaptively applies the proper replacement algorithm based on the system states. We carefully argue that multimedia workload may exhibit the characteristics other than sequential access pattern. In distance learning application, for example, the user may exploit the VCR-like function(rewind and play) of the system and access the particular segments of video repeatedly while scanning a file. If the access pattern is not sequential, the legacy interval-based caching strategy does not work properly.

We propose buffer replacement scheme called *Adaptive Buffer Management (ABM)* adaptively applies the appropriate replacement policy. The objective of ABM is to apply DISTANCE policy to files whose reference pattern is mainly sequential and LRU policy to files that have many looping references. In order to properly characterize the workload behavior, e.g. *if there exists the looping reference for a given file* and how *strong* it is, we propose a metric called *looping reference indicator*. In ABM, the server regularly monitors the system and updates the looping reference indicator for individual files. This mechanism enables the server to adaptively change the buffer cache replacement algorithm based on the dynamically changing workload characteristics. The results of the simulation based experiments show that ABM algorithm exhibits superior cache hit rate than both LRU and DISTANCE depending on the threshold value of looping

[1] Similar problem and its solution was discussed in [11], too.

reference indicator. We strongly believe that ABM(Adaptive Buffer Cache Management) algorithm proposed in this article is very suitable candidate for buffer replacement scheme in next generation streaming system. ABM will manifest itself particularly when the workload exhibits not only sequential but also looping access characteristics.

References

1. 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.
2. Matthew Andrews and Kameshwar Munagala. Online algorithms for caching multimedia streams. In *European Symposium on Algorithms*, pages 64–75, 2000.
3. P. Cao, E. Felten, and K. Li. Implementation and performance of application-controlled file caching. In Proceedings of the First Symposium on Operating Systems Design and Implementation, 1994.
4. 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.
5. J. Choi, S. Noh, S. Min, and Y. Cho. An implementation study of a detection-based adaptive block replacement scheme. In *USENIX Annual Technical Conference*, pages 239–252, 1999.
6. E. G. Coffman, Jr. and P. J. Denning. *Operating Systems Theory.* Prentice–Hall, Englewood Cliffs, New Jersey, 1973.
7. 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.
8. A. Dan and D. Sitaram. Buffer management policy for a on-demand video server. Technical Report RC 19347, IBM.
9. 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.
10. 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.
11. D. Lee, J. Choi, J. Kim, S. Noh, S. Min, Y. Cho, and C. Kim. On the existence of a spectrum of policies that subsumes the least recently used (lru) and least frequently used (lfu) policies. In *ACM SIGMETRICS Conference*, 1999.
12. Nobuo Nakajima. The path to 4g mobile. *IEEE Communications*, 39(3):38–41, March 2001.
13. E. O'Neil, P. O'Neil, and G. Weikum. Page replacement algorithm for database disk buffering. SIGMOD Conf., 1993.
14. Banu Özden, Rajeev Rastogi, and Abraham Silberschatz. Buffer replacement algorithms for multimedia storage systems. In *International Conference on Multimedia Computing and Systems*, pages 172–180, 1996.
15. 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.

16. R. Patterson, G. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In Proc. 15th Symposium on Operating Systems Principles, 1995.
17. Lawrence A. Rowe, Diane Harley, and Peter Pletcher. Bibs: A lecture webcasting system. Technical report, Berkeley Multimedia Research Center, UC Berkeley, June 2001.
18. Youjip Won and Jaideep Srivastava. "smdp: Minimizing buffer requirements for continuous media servers". *ACM/Springer Multimedia Systems Journal*, 8(2):pp. 105–117, 2000.
19. S. Kaplan Y. Smaragdakis and P. Wilson. Eelru: Simple and effective adaptive page replacement. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 122–133, 1999.