A HYBRID BUFFER CACHE MANAGEMENT SCHEME FOR VOD SERVER

Kyongwoon Cho[1] *Yeonseung Ryu*[2] *Youjip Won*[3] *Kern Koh*[1]

[1]School of Computer Engineering, Seoul National Univeristy, Korea
[2]Division of Information and Communication Engineering, Hallym University, Korea
[3]Division of Electrical and Computer Engineering, Hanyang University, Korea

ABSTRACT

In this paper, we present a novel buffer cache management scheme for Video-on-Demand Servers called *Hybrid Buffer cache Management (HBM)*. We carefully argue that multimedia workload may exhibit the characteristics other than sequential access pattern. If the access pattern is not sequential, the legacy buffer cache schemes that are grounded at the assumption that the applications sequentially access the multimedia file may not work properly. The HBM automatically detects the reference patterns of each file and intelligently switches between different buffer cache management schemes on per-file basis. According to our experiment, the HBM exhibits better buffer cache hit ratio than interval based caching or LRU, especially when the workload exhibits not only sequential but also looping reference property.

1. INTRODUCTION

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 the tremendous rate. In entertainment arena, user watches the video clip which can be *news*, *movie*, etc. Typical data block access pattern is sequential in these applications. In on-line education arena, user watches the lecture on-line with the lecture materials and instructor's annotation appears on the same screen synchronized what speaker is saying. Some studies showed that users may not access the files sequentially in on-line learning application[1, 2, 3, 4, 5].

A number of buffer cache replacement algorithms which are designed for multimedia storage servers exploit sequential data access characteristics[6, 7, 8, 9]. [6, 7] proposed Interval Caching algorithm. It exploits temporal locality between streams accessing the same file, by caching *intervals* between successive streams. [8] 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 distance 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, to be shown in detail in section 4, they show good performance in the sequential streaming workload.

We argue that multimedia workload may exhibit the characteristics other than sequential access pattern. In distance learning environment, for example, the user may exploit the VCR-like functions (rewind and play) of the system and accesses the particular segments repeatedly while scanning the video file. Looping reference can be thought as either sequential access or temporal localized access depending on the size of the loop. Thus the appropriate buffer cache management scheme needs to be applied. We argue that the interval based caching may not deliver desirable performance when the workload exhibits looping reference behavior.

In this paper, we propose a novel buffer cache management scheme called *Hybrid Buffer cache Management (HBM)*. The proposed HBM scheme maintains a metric called *Looping Reference Indicator* for each opened files which denotes whether there exist looping reference patterns and how *effective* they are. The loop is said to be *effective* if the data blocks in the loop can be entirely loaded in the buffer cache. The HBM periodically updates the Looping Reference Indicator and dynamically switches between Interval based caching or LRU replacement scheme. The HBM is applied in per file basis. Simulation study shows that the HBM scheme can significantly improve the buffer cache miss ratio, especially when the underlying workload exhibits not only sequential but also looping reference property.

The remainder of the paper is organized as follows. In section 2, we discuss the workload characteristics in multimedia files. In Section 3, we describe *Hybrid Buffer cache Management* scheme. Section 4 presents results of performance analysis and verifies that the proposed algorithm behaves as expected. Section 5 concludes the paper.

2. REFERENCE PATTERNS IN MULTIMEDIA FILES

Recenlty, a number of research results have been published regarding the workload analysis of streaming and/or educational media servers[1, 2, 3, 4, 5]. Padhye et al. [4] analyzed the client access to MANIC system audio content. Aimeida et al. [2] and Harel et al. [5] analyzed the access log of their educational media server, eTeach and Classroom 2000, respectively. For example, Aimeida et al. [2] showed that for short files, interactive requests like jump backwards are common. Rowe et al. [1] addressed that students access the video clips to review the materials they were not able to understand properly during the class. These works present insightful information about the workload of media servers, e.g. access frequency distribution, file popularity, aging of file access popularity, session characteristics, client interactivity.

The legacy interval based buffer cache management schemes assume that typical access behavior of streaming workload is *sequential*. However, this commonly accepted assumption may not hold in a certain situation and we may overlook its implication on the system performance. In on-line education field, the students may want to review what instructor has said about a particular topic since they have difficulty in understanding it. In this situation, it is possible that the user accesses particular segment of the video repeatedly rather than sequentially scans the file from the beginning to the end. We call the reference pattern where there exists repetitive sequence of reference pattern as *looping reference*. Fig. 1 illustrates the looping reference.

0-7803-7304-9/02/$17.00 C2002 IEEE

f_i denotes the i^{th} frame. In Fig. 1, we can find that the consecutive frames $f_2 f_3 f_4$ and $f_6 f_7$ are viewed repeatedly. The length of the loop is three and two frames, respectively.

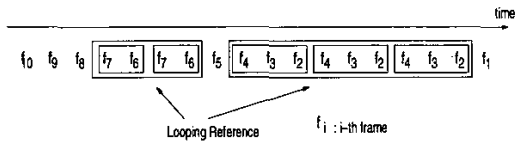


Figure 1: Looping References.

Depending on the time scale of interest, looping reference pattern can be thought as temporally local workload or sequential workload. The time scale of interest depends on the amount of buffer cache available. 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 [10].

When the workload exhibits looping reference, the system should decide whether the ongoing workload is sequential or not. Based on the decision, either the interval based algorithm or the LRU is to be applied. The victim selection mechanism of the LRU and the interval based caching is opposite to each other. The LRU selects the least recently used page as a victim while interval based algorithm actually chooses the youngest data block in the buffer cache. Thus, it is not possible to find the compromizing solution between these two. It is worth noting that whether the given looping reference(if there is any) is sequential or not is subject to the size of buffer cache. If the buffer cache is large enough to hold all data blocks in the loop(the loop becomes *effective*), the LRU may be the better choice. Othewise, interval based caching will be more appropriate.

3. HYBRID BUFFER CACHE MANAGEMENT

3.1. Effective Loop

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

Let s and $S_t(i)$ be the user id and the set of users who access the file i at time t. Let $|S_t(i)|$ be the number of users 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 access the file i at t. That is, $SR_t(i) = \{\ s \mid N_i(R_t(s)) = N_i(R_{t-1}(s)) + 1$ 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*, $L_t(s)$. 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.

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

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

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. $\delta(i)$ is based on the most recent θ samples. θ value enables us to control the computational overhead of maintaining $ER_t(i)$, $NER_t(i)$ and $SR_t(i)$. To reduce the overhead, it may be beneficial to make θ small. However, small value of θ may make *LRI* overly sensitive to workload fluctuation.

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

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*[8] 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^*, the DISTANCE scheme is applied to that file. Otherwise, the LRU scheme is applied.

3.2. Buffer Management Method

The HBM cache manager consists of three modules: system buffer manager, DISTANCE manager and LRU manager. The system buffer manager executes the HBM algorithm and controls the allocation of buffer space. When the application first opens the file, it is initially allocated to the DISTANCE manager. This is based on the assumption that streaming workload normally accesses the file in sequential fashion. The system buffer manager computes the looping reference indicator of opened files and compares them with threshold value, δ^*. If the indicator value of a file is greater than or equal to δ^*, then the file is allocated to the LRU manager. Otherwise, the file remains under the DISTANCE manager's control. When the file is determined to transfer to the LRU manager, the data blocks allocated to that file become controlled by the LRU manager. On the contrary, if the file is transferred from the LRU manager to the DISTANCE manager, the data blocks are also trasnferred to the DISTANCE manager.

4. SIMULATION STUDY

In this section, we present the results of simulation. We first examine the cache miss rate of legacy buffer cache replacement algorithms, e.g. OPT, DISTANCE, LRU, LRU-k and MRU. Then, we compare the cache miss rate of the HBM scheme with those of legacy buffer cache management schemes and show the effectiveness of the *Hybrid Buffer Cache Management* scheme. In this work, synthetic workload is used in simulation. A workload is characterized by loop parameters which consists of three attributes: *interval between loops*, *loop length* and *loop count*. For example, loop parameter of (100, 50, 3) means that (i) the distance between the completion of the one looping reference and the beginning of the following looping reference is 100 second on the average, (ii) loop length is 50 blocks and (iii) loop count is 3.

In all the experiments, inter-arrival times of clients are exponentially distributed and average interarrival time is 100 seconds. 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 hit/miss ratio and is measured through 43200 service rounds simulation.

First, we examine the performance of existing schemes, e.g. DISTANCE, LRU, LRU-k and MRU. We assume that LRU-k scheme reflects the most recent two references. We present the result of experiment that shows the effect of frequency of looping accesses by varying the average interval between loops (*IBL*) of each client. In this experiment, we set the loop length and the loop count to 40 blocks and 5 times, respectively. Buffer cache size is set to 6000 blocks.

Fig. 2 illustrates the effects of interval between the loops on the cache hits under different buffer replacement schemes. OPT in Fig. 2 gives illustrative information for optimal algorithm. When the workload has only *sequential* access pattern, DISTANCE based replacement scheme behaves almost as good as optimal buffer replacement. LRU and MRU yield much lower hit ratio than DISTANCE. Our simulation results shows that benefit of DISTANCE based buffer cache replacement scheme becomes dominant as the cache size increases. This result suggest that DISTANCE based buffer cache management scheme is the right choice when the workload exhibits only sequential reference pattern. When sequential and looping references co-exist, LRU's hit rate is highly dependent on frequency of the loop. As the loop occurs more frequently, i.e. interval between the loops(*IBL*) becomes shorter, LRU based buffer cache management scheme yields higher the hit ratio. When the looping reference does not occur frequently, i.e. interval between the loop(*IBL*) is long, the hit ratio of LRU is yet lower than that of DISTANCE. This phenomenon can be seen when *IBL* is 500 and 1000 in Fig. 2, for example. When *IBL* is 100, LRU yields higher hit ratio than DISTANCE(by 30%) and is as good as OPT.

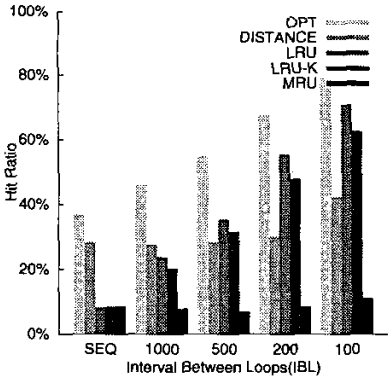


Figure 2: Effects varying IBL

To investigate the effectiveness of the HBM scheme, we ran experiments with a number of different looping reference patterns for each file. There are 20 files in total. Table 1 illustrates the four looping distribution types (LDT) used in our experiment. In the Table 1, $\mathbf{IBL}(i)$ is the average interval between loops of clients accessing file i. In all LDT, we set the average length of loop and the loop count to 20 blocks and 5 times, respectively. Fig. 3 illustrates that the miss rate of HBM ranges between LRU and DISTANCE with varying the

Table 1: Description of LDT used in experiments

LDT	Description
LDT1	$\mathbf{IBL}(i) = 100$, for all file ($1 \leq i \leq 20$)
LDT2	$\mathbf{IBL}(i) = 1000$, for all file ($1 \leq i \leq 20$)
LDT3	$\mathbf{IBL}(1) = 50$ $\mathbf{IBL}(i) = \mathbf{IBL}(1) * 1.2^{(i-1)}$
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 $i \leq \frac{N}{2}$, $\mathbf{IBL}(i) = \mathbf{IBL}(1) * 1.2^{(i-1)}$ Otherwise, $\mathbf{IBL}(i) = \frac{L(i)}{1.2^{(N-i)}}$

threshold of looping reference indicator(δ^*). δ^* is the threshold value to determine whether to apply either DISTANCE or LRU. If δ^* is 0, *LRI* is always greater than δ^* and henceforth LRU algorithm is used to replace buffer cache. If δ^* is large enough such that *LRI* in practice is always less than δ^*, HBM becomes equivalent to DISTANCE. As can be seen in figures in Fig. 3, the miss rate of hybrid cache management scheme(HBM) lies between the LRU and DISTANCE and varies with δ^*. The values in each legend represent the size of cache.

Fig. 3(a) summarizes the result of experiment with LDT1. Be reminded that when IBL is 100 in Fig. 2, LRU performs better than DISTANCE. Let us examine the cache size of 4000 blocks. If δ^* is smaller than 0.02, HBM applies LRU policy to all files. Hence, the miss ratio of HBM becomes the same as that of LRU. If δ^* is greater than 0.025, HBM applies DISTANCE policy to all files and thus, the miss ratio of HBM equals that of DISTANCE. It is worth noting that the miss rate of HBM changes rapidly when δ^* is changing between 0.02 and 0.025.

Fig. 3(b) shows the results of experiment with LDT2. In LDT2, $\mathbf{IBL}(i)$ for all file is set to 1000. In this case, HBM should select DISTANCE policy to all files because DISTANCE outperforms LRU as shown in Fig. 2. However, if we use very small threshold value, HBM may execute LRU policy to the files even though the files have few looping references. When δ^* is smaller than 0.01 and the cache size is 4000, the miss ratio of HBM is worse than that of DISTANCE.

In LDT3, we assign different $\mathbf{IBL}(i)$'s to each file. $\mathbf{IBL}(i)$ decreases with the increase of i. In this case, HBM applies LRU policy to files whose looping reference indicator is greater than δ^* and applies DISTANCE policy to files whose looping reference indicator is smaller than δ^*. Fig. 3(c) shows the result of LDT3. Consider the size of cache is 4000. HBM can outperform than both LRU and DISTANCE if it uses 0.01 as δ^*.

Fig. 3(d) illustrates the result when using LDT4 as input workload. In LDT4, we partition files into two groups. Files in the first group are assigned small IBL and files in the second group are assigned large IBL. In this case, HBM tries to apply LRU policy to the first group's files and DISTANCE policy to the second group's files. As in the case of using LDT3, HBM performs better than both LRU and DISTANCE when δ^* is 0.01 or 0.02.

5. CONCLUSION

In this paper, we propose a novel buffer replacement scheme called Hybrid Buffer cache Management(HBM) that detects the *effective*

looping reference pattern and adaptively applies appropriate replacement policy. We develop a metric called *looping reference indicator* to denote the *degree* of looping reference. The HBM scheme periodically monitors the looping reference indicator of opened files and dynamically switches between the DISTANCE and the LRU on per-file basis.

Our simulation study revealed a number of interesting phenomenon. We observed that the LRU policy can yield lower cache miss rate than the DISTANCE policy if effective looping references constitute dominant fraction of workload. Even though there exist looping references, the LRU policy may not work properly if the buffer cache can not accommodate the entire data blocks in the loop. We also observed that the HBM scheme exhibits better cache hit rate than both the LRU and the DISTANCE when the threshold value of looping reference indicator is properly established. The dynamic and intelligent nature of the HBM manifests itself when the underlying workload exhibits both sequential and looping reference access pattern.

6. REFERENCES

[1] Lawrence A. Rowe, Diane Harley, and Peter Pletcher, "Bibs: A lecture webcasting system," Tech. Rep., Berkeley Multimedia Research Center, UC Berkeley, June 2001.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

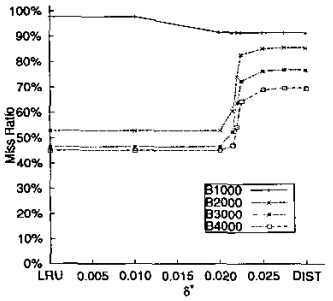
[6] A. Dan and D. Sitaram, "Buffer management policy for a on-demand video server," Technical Report RC 19347, IBM.

[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.

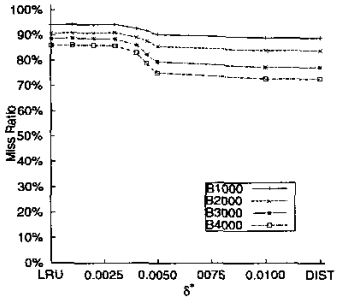
[8] Banu Özden, Rajeev Rastogi, and Abraham Silberschatz, "Buffer replacement algorithms for multimedia storage systems," in *International Conference on Multimedia Computing and Systems*, 1996, pp. 172–180.

[9] Youjip Won and Jaideep Srivastava, ""smdp: Minimizing buffer requirements for continuous media servers"," *ACM/Springer Multimedia Systems Journal*, vol. 8, no. 2, pp. pp. 105–117, 2000.

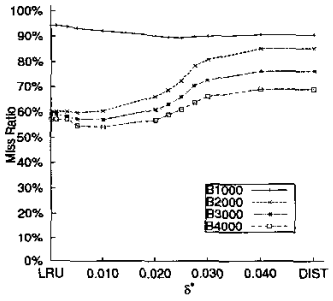
[10] E. G. Coffman, Jr. and P. J. Denning, *Operating Systems Theory*, Prentice–Hall, Englewood Cliffs, New Jersey, 1973.



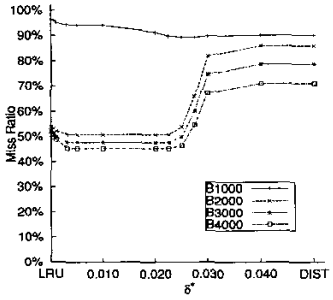
(a) LDT1



(b) LDT2



(b) LDT3



(b) LDT4

Figure 3: Performance of HBM.