A NOVEL BUFFER CACHE SCHEME FOR DISTRIBUTED MULTIMEDIA STREAMING

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

[1]School of Computer Engineering, Seoul National Univeristy, Korea
[2]Department of Computer Software, Myongji University, Korea
[3]Division of Electrical and Computer Engineering, Hanyang University, Korea

ABSTRACT

In this paper, we present a novel buffer management algorithm for multimedia streaming workload. We carefully examine the workload traces obtained from several streaming servers in service. The analysis results show that most users exhibit non-sequential access pattern using VCR-like operations such as jump backward and jump forward. Moreover, short jump accesses are shown to be common. We exploit the workload characteristics of the VCR-like operation and develop a buffer caching algorithm called *Virtual Interval Caching* scheme. Experimental results show that the proposed buffer management scheme yields better performance than the legacy schemes.

Key Words: Buffer Cache Replacement, Multimedia, VCR-like Operations

1. INTRODUCTION

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 applications which require frequent disk access greatly depends on the performance of I/O. 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 charateristics with bandwidth guarantee. There have been widely proposed interval-based buffer caching schemes for multimedia servers [1, 2, 3]. Interval-based buffer caching schemes take advantage of sequential access characteristics of continuous media data. They cache data blocks in the *intervals* of consecutive streams accessing the same file. In order to maximize the number of playbacks accessing data from buffer cache, interval based schemes sort intervals with increasing order and cache data blocks from the shortest intervals. [1] showed that interval-based caching schemes have optimal performance in sequential multimedia streaming workload.

A few studies recently examine the user access logs obtained from streaming servers in service and show that there are non trivial amount of VCR-like operations, e.g. jump forward, jump backward, or etc [4, 5, 6, 7, 8]. These works also deliver insightful information on the usage of the multimedia 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, [5] showed that for short media files, interactive requests like jump backward are common.

In this work, we analyze three multimedia streaming servers in Korea; *Hanmir Education Center* [9], *Ongamenet* [10] and *Myung Films* [11]. Like previous works, we can find the evidence that users access the multimedia files in non-sequential patterns using VCR-like operations and short jump operations are frequent. In order to manages the buffer space effectively under such a real workload that a non-sequential access patterns exist, we propose a novel buffer management scheme called *Virtual Interval Caching (VIC)*. The proposed VIC scheme maintains past intervals as virtual intervals and increases buffer cache hit ratio. Experimental results show that the proposed buffer management scheme yields better performance than the legacy schemes such as interval-based schemes, LRU, etc.

The remainder of this paper is organized as follows. In section 2, we show the access activity to media files by analyzing trace data obtained from three streaming servers in service. In section 3, we presents proposed buffer caching scheme called *VIC (Virtual Interval Caching)*. Section 4 presents performance results and finally section 5 summarizes our works.

2. ANALYSIS OF MULTIMEDIA STREAMING WORKLOAD

It has been commonly accepted that VOD streaming workload has sequential file access characteristics and the legacy buffer caching techniques utilize such a sequentiality that they optimize their buffer replacement algorithms. Most of popular media players, such as Window Media Player, have VCR control capabilities and they can change playback positions or stop the playing during normal playing. Such a non-sequential file access may degrade the performance of caching techniques which have been grounded at the assumption that the application sequentially accesses the media file.

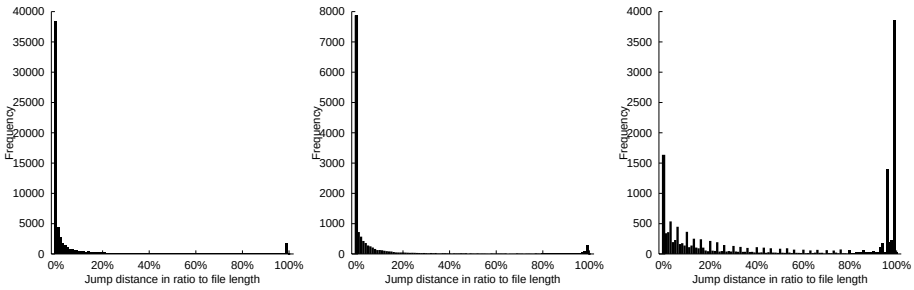
In order to provide well-founded evidence that there exist non-sequential accesses in real-world workload, we analyze Windows Media Server logs of 3 streaming service sites: *Hanmir Education Center* [9], *Ongamenet* [10] and *Myung Films* [11]. The client players have VCR control capabilities such as jump forward, jump backward, pause and stop. Each log entry is stored on the server not only when a new playback begins but also when jump operations are requested. So, the session of one streaming service consists of one or more separate playbacks. Playbacks within a single session can be easily collected from the logs.

Analyzed results are shown in Table 1. *Hanmir* and *Ongamenet* media server are accessed frequently but *Myung Films* server has only 1.1 concurrent streams. One of the interesting things about the results is that there are the significant number of VCR-like operations such as jump forward or jump backward. In case of *Hanmir*, the number of jump operations per each session is averaged 1.62.

Jump distance is a distance from the playback position before jump to the current position, which tends to be short as shown in Fig. 1. But there are a few of jump distances running close to file length. This tendency is very clear in Fig. 1(c). It is owing to that

	Hanmir	Ongamenet	Myung Films
Log Duration	10 days	3 days	43 days
Number of log entries	192457	98640	69330
Number of file	3432	953	468
File Length	1009 secs	809 secs	141 secs
Number of sessions	73655	55423	42119
jump operations	1.62	0.78	0.64
backward distance	281.28 secs	254.32 secs	77.93 secs
forward distance	311.68 secs	153.63 secs	26.31 secs
Concurrent sessions	38.78	166.28	1.1

Table 1: Analyzed results of 3 Windows Media Servers



(a) *Hanmir* (b) *Ongamenet* (c) *Myung Films*

Figure 1: Jump distance distribution

stream is jumped forward from start to the end by a hasty user or a whole file(especially a small file) is accessed repeatedly.

As stated above, there are a significant of jump operations in streaming workload and jump distance has a bias towards being short. Thus buffer caching algorithm in streaming server should take into account it and more elaborated caching technique is required when client caching is not affordable such as in mobile environments.

3. VIRTUAL INTERVAL CACHING

3.1. Concept of Interval

In this subsection, we introduce the concept of *interval* and the interval-based buffer caching algorithms [1, 12, 2].

In Fig. 2, each S_i denotes the stream accessing the file and the small arrows marked by stream S_i denote the current positions in a file. In the figure, I_{21} and I_{32} represent intervals formed by two consecutive streams. The two streams of an interval are referred to as the preceding and the following streams.

The basic idea of interval-based caching algorithms is that by caching the blocks brought in by the preceding stream, it is possible to serve the following stream from the cache. The cache requirement of an interval is defined to be the number of blocks needed to store the interval and is a function of the time-interval between the two streams and the compression method used. The interval caching scheme selects the intervals to be cached so as to maximize the number of streams served from cache. It orders the intervals in terms of their cache requirements and caches blocks first from the shortest intervals.

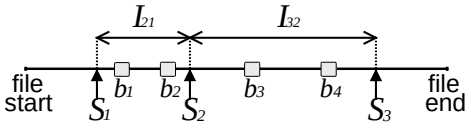


Figure 2: Interval between two consecutive streams

In the circumstance of only sequential accesses, changes to the interval list occur due to the arrival of a new stream or termination of a stream. When a new client arrives, a new interval is formed with the preceding stream. If there exist VCR operations, intervals may be changed, merged, split, created and/or removed when VCR request is issued.

3.2. Virtual Interval Caching

We define a *virtual interval* as an interval which is associated with only one stream or is not associated with any stream. A *real interval* is defined as an interval which has both a preceding stream and a following stream. We also define a *head interval* as an interval whose immediate following stream is located at the largest position in the file. The head interval can have no preceding stream. Likewise, an interval whose immediate preceding stream is at the smallest position

in the file is a *tail interval*.

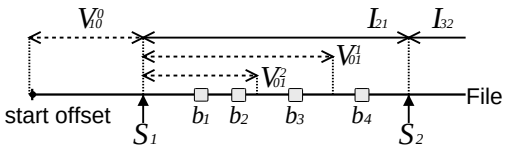


Figure 3: Virtual Interval

Fig. 3 shows relationship between real interval and virtual interval. I_{21} represents a real interval which is generated by two streams S_2 and S_1. In contrary, V_{01}^1, V_{01}^2 and V_{10}^0 are virtual intervals which have only a following stream or neither of streams. We can see that I_{21}, V_{01}^1, V_{01}^2 have the same following stream but only I_{21} has a preceding stream.

A virtual interval is created when a stream disappears due to playback position change of a stream such as a stop or jump. And virtual interval can be divided into two smaller ones or reduced to a smaller one when a new stream arrives or an existing stream resumes playing from a new position. Fig. 4 shows that a new virtual interval is created. In this figure, as the preceding stream of I_{ji} goes away, it is simply converted into V_{0i}^1. Its interval size is not changed. But the interval size of V_{0j}^0 enlarge because its immediate following stream becomes I_{ih}.

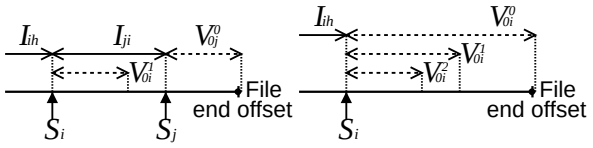
Fig. 5 explains that a virtual interval is reduced to a smaller one. After S_j appears, I_{ki} is split into I_{ji} and I_{kj}. At the same time V_{0i}^1 in Fig. 5(a) also get split but is regarded as being reduced into V_{0j}^1 because its immediate following stream becomes S_j.

3.3. Replacement Algorithms

For approximation to maintaining access probabilities per each buffer, *virtual interval caching* conducts replacement policy based on averaged buffer access probabilities per each interval which are proportional to its interval size except tail intervals. In the case of a tail interval, interval size should be estimated from a average number of streams within a file as follows:

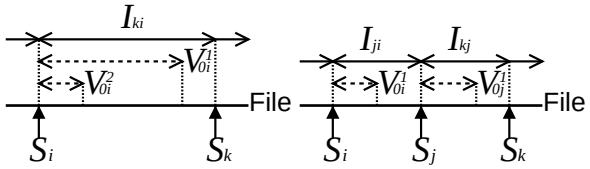
$$max\{interval\ size\ ,\ \frac{file\ length}{average\ number\ of\ streams}\}$$

By letting the size of virtual intervals same as their interval sizes, access probabilities of intervals can be obtained seamlessly with *interval size*. As such, *VIC* selects a buffer in the interval with largest virtual size as a victim at replacement time.



(a) Before S_j disappears (b) After S_j disappears

Figure 4: A real interval is converted into a virtual interval due to the disappearance of an existing stream



(a) Before S_j appears (b) After S_j appears

Figure 5: A virtual interval is reduced to a smaller one due to the advent of a new stream

4. EXPERIMENT RESULTS

In this section, we present performance analysis of proposed VIC algorithms through the comparison with existing buffer cache management schemes such as Interval Caching(IC), LRU, LRU-k[13], MRU and Optimal algorithm(OPT). Throughout experiments LRU-k takes into account knowledge of the last two references (i.e. k = 2). To facilitate our simulation study, we implemented a simulator conducting various algorithms with real-world workload traces not synthetic one. Windows media server logs mentioned in Section 2 are cooked into that traces as a input to the simulator. In the process of trace generation, it is assumed that playback rates of all stream are same. So, every stream needs same amounts of data per each round. We use a fixed size of retrieval block as a unit of I/O and buffer caching. The performance metric we use is cache miss ratio. We examine the effects of varying the size of buffer cache on cache misses under various buffer replacement algorithms.

Fig. 6 illustrates the simulation results using *Hanmir* trace data. The size of the buffer cache is changed from 2000 to 16000 (in terms of the number of buffer blocks). In the figure, OPT(optimal algorithm) is not online buffer caching algorithm and just gives explanatory information. It is observed that VIC delivers lower miss ratio than other algorithms. VIC is average 6% better than IC or LRU-k. The miss ratio of LRU-k is comparable to the those of IC but it slightly outperforms IC from buffer size 10000.

Figure 7 shows results using *Ongamenet* trace. We observed many concurrent streams from *Ongamenet* trace data. Thus, it is likely that there exist not a few intervals. This leads to good performance of interval caching and virtual interval caching, which is very close to OPT. With small buffer size (i.e., less than 8000), difference of miss ratio between IC and VIC is not so much. However, VIC shows a better miss ratio as buffer size increases. The miss ratio of the VIC is 6.7% lower than IC or LRU-k and 12.9% lower than LRU at buffer size 20000.

Results with *Myung Films* trace is provided in Fig. 8. When buffer size is 500, VIC is 2.8% and 8.6% better than LRU-k and IC, respectively. However, we can see that VIC does not provide a lower miss ratio any more as the buffer size increases. This is because there are too few concurrent streams and thus the number of intervals is not so sufficient that VIC runs effectively.

5. CONCLUSION

In our work, it is shown that there exist a significant number of non-sequential access in real multimedia workload through analysis of real-world media server's log. Most of the non-sequential file ac-

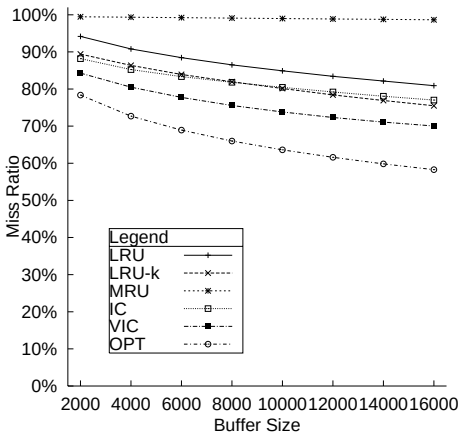


Figure 6: Miss ratio comparison(using *Hanmir* trace)

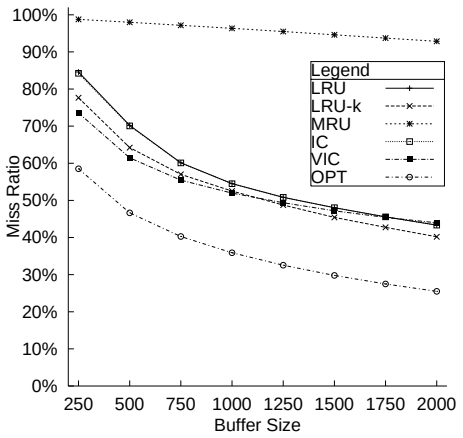


Figure 8: Miss ratio comparison(using *Myung Films* trace)

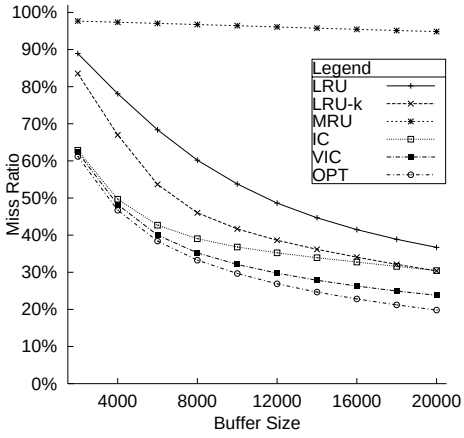


Figure 7: Miss ratio comparison(using *Ongamenet* trace)

cesses are jump forward or backward operations, which are triggered by VCR control capable client player. Moreover, the jump distances have a tendency towards being short. Therefore, we need new buffer caching algorithms that exploit various file access workload in multimedia streaming servers.

Our proposed buffer replacement algorithm called *Virtual Interval Caching (VIC)* can handle buffer cache effectively under such a real workload that a non-sequential access pattern exists. Unlike previous interval caching scheme, VIC keeps intervals that were removed by VCR operations as virtual intervals. By keeping data blocks in the buffer cache that frequently accessed by VCR operations, VIC can increases cache hit probability. We showed through trace-driven simulations that VIC algorithm can perform over 5% better than well-known legacy algorithms.

6. REFERENCES

[1] 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.

[2] 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.

[3] Matthew Andrews and Kameshwar Munagala, "Online algorithms for caching multimedia streams," in *European Symposium on Algorithms*, 2000, pp. 64–75.

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

[5] 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.

[6] M. Chesire, A. Wolman, G. Voelker, and H. Levy, "Measurement and analysis of a streaming media workload," in *Proceedings of 3^{rd} USENIX Symp. on Internet Technologies and Systems*, San Francisco, CA, USA, March 2001.

[7] 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.

[8] 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.

[9] "Hanmir education center," `http://edu.hanmir.com`.

[10] "Ongamenet," `http://www.ongamenet.com`.

[11] "Myung films," `http://www.myungfilm.co.kr`.

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

[13] E. O'Neil, P. O'Neil, and G. Weikum, "Page replacement algorithm for database disk buffering," SIGMOD Conf., 1993.