Intelligent Buffer Cache Management in Multimedia Data Retrieval

Yeonseung Ryu[1⋆], Kyoungwoon Cho[2], Youjip Won[3⋆⋆], and Kern Koh[2]

[1] Division of Information and Communication Engineering, Hallym University, Korea
ysryu@hallym.ac.kr
[2] School of Computer Engineering, Seoul National Univeristy, Korea
{cezanne, kernkoh}@oslab.snu.ac.kr
[3] Division of Electrical and Computer Engineering, Hanyang University, Korea
yjwon@ece.hanyang.ac.kr

Abstract. In this paper, we present an intelligent buffer cache management algorithm in multimedia data retrieval called *Adaptive Buffer cache Management (ABM)*. The proposed ABM scheme automatically detects the reference pattern of each file and intelligently switches between different buffer cache management schemes on per-file basis. According to our simulation based experiment, the ABM scheme yields better buffer cache miss ratio than the legacy buffer cache schemes such as LRU or interval based scheme. The ABM scheme manifests itself when the workload exhibits not only sequential but also looping reference patterns.

Keywords: Buffer Cache, Multimedia File System, Looping Reference

1 Introduction

The multimedia streaming workload is generally characterized as sequential access of the data. Multimedia technology is being applied in various fields, e.g. entertainment, education, tele-collaboration to list a few. Recently some studies showed that users may not sequentially access the files in on-line educational media server [8,1,7,5]. In on-line education system, students can access the particular segment of video clips repeatedly to review the materials they were not able to understand properly. Thus, it is possible that some users scan the file sequentially while the others repeatedly view the particular segment of the same file. In this case, aggregated file access workload may exhibit mixture of widely different access characteristics. The buffer cache replacement scheme should be able to change the policy dynamically depending on the workload characteristics.

In this paper, we propose a buffer cache replacement algorithm called *Adaptive Buffer cache Management (ABM)*, which automatically detects the reference pattern of each file and intelligently applies an appropriate policy per-file basis.

⋆ This work was supported by the Research Fund 2001 at Hallym University.
⋆⋆ The work of this author was supported by KOSEF through Statistical Research Center for Complex System at Seoul National University.

M.-S. Hacid et al. (Eds.): ISMIS 2002, LNAI 2366, pp. 462–471, 2002.

The proposed ABM 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 ABM scheme periodically monitors the reference pattern and applies different buffer replacement policies depending on whether the current workload exhibits sequential or looping reference characteristics. Simulation study shows that the ABM 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 this paper is organized as follows. In Sect. 2, we discuss the buffer replacement schemes for multimedia server. In Sect. 3, we describe *Adaptive Buffer Cache Management* algorithm. Section 4 presents results of performance analysis and verifies that the proposed algorithm behaves as expected. Finally, we conclude with a summary in Sect. 5.

2 Buffer Cache Schemes for Multimedia File System

When a stream accesses the file in sequential fashion, it is unlikely that recently accessed data block is to be referenced by the same stream in the near future. Rather, it will be referenced by different streams which access the same file with a certain time interval. Thus, as the age of the data block gets older, it is more likely that the block is to be accessed sooner. In this environment, the interval based caching policy looks promising way of minimizing the buffer cache miss ratio.

A number of articles proposed the interval based cache replacement schemes and showed that they deliver better buffer cache performance in multimedia streaming environment than the legacy LRU buffer cache replacement scheme [4, 3,6,9]. Dan *et al.* proposed *Interval Caching* algorithm [4,3]. It exploits temporal locality between streams accessing the same file, by caching *intervals* between successive streams. Özden *et al.* presented *DISTANCE* algorithm [6]. It also caches the blocks in *distance* of successive streams accessing the same media file.

The interval based replacement schemes maintain information about intervals (or distances) of consecutive streams accessing the same file. When there are only sequential references, intervals between consecutive streams do not change normally. However, if there exist looping references, intervals can be changed whenever loop begins. After intervals are changed, the interval based schemes are going to change contents of cache space gradually. That is, when new block needs to be read into the buffer cache, the interval based schemes must determine which blocks to cache and which blocks to replace using the newly created (or modified) interval sets. In order to maximize the number of streams accessing data from the buffer cache, the interval based schemes cache data blocks from the shortest intervals.

Recently, a number of research results have been published regarding the workload analysis of educational media servers [8,1,7,5]. In on-line education

system, the user watches lecture on-line and the lecture materials and instructor's annotation appears on the same screen synchronized what speaker is saying. We carefully suspect that the users may exhibit *more than* sequential behavior in this environment. Aimeida *et al.* showed that for short files, interactive requests like jump backwards are common [1]. Rowe *et al.* addressed that students access the video clips to review the materials they were not able to understand properly during the class [8]. 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*.

Looping reference can be thought as temporally localized or sequential depending on the time scale of interest. 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 [2].

The victim selection mechanism of the LRU and the interval based scheme is opposite to each other. The LRU selects the least recently used block as a victim while the interval based schemes select the block from the largest interval. The DISTANCE scheme, for example, actually chooses the youngest data block in the buffer cache which belongs to the largest interval. Thus, it is not possible to find the compromising solution between these two. If the buffer cache is large enough to hold all data blocks in the loop (the loop becomes *effective*), the LRU may be the right choice. Otherwise, the interval based schemes will be more appropriate. We will examine the relationship between the size of the loop and the size of buffer cache for each replacement scheme in more detail in Sect. 4.

3 Intelligenet Buffer Cache Management

3.1 Characterizing the Degree of Looping Reference

We propose a metric called *Looping Reference Indicator* to denote whether there exist looping reference patterns for a given file and how strong it is. The objective of this metric is to decide which of the two buffer replacement schemes to use: interval based caching or LRU. In this work, we choose the DISTANCE policy as interval based caching scheme.

Let u and $U_t(i)$ be the user id and the set of users who access the file i at time t. $|U_t(i)|$ is the number of members in $U_t(i)$. Let $N_i(R_t(u))$ be the logical block number of file i which is accessed by the user u at t. Let $SU_t(i)$ be the set of users in $U_t(i)$ who sequentially access the file i at t. That is, $SU_t(i) = \{ u \mid N_i(R_t(u)) = N_i(R_{t-1}(u)) + 1 \text{ and } u \in U_t(i) \}$. And the set of users who do not sequentially access the file i at t is denoted by $SU_t(i)^c$. Let $B_t(i)$ be the number of data blocks of file i in the buffer cache at time t. When u is accessing the file in looping reference at t, let $L_t(u)$ denote the length of the loop in terms of the number of data blocks. The length of the loop becomes available when

user accesses a certain segment of a file repeatedly. Also, it is not unreasonable to assume that user can mark the looping interval, i.e. segment of interest and pass the information of the looping reference to file system.

A loop is said to be *effective* if the number of buffer cache pages allocated to user u is greater than the length of the loop, $L_t(u)$. Let $EU_t(i)$ be a set of users in $SU_t(i)^c$ who have effective looping reference at t. These are the set of users whose subsequent I/O requests can be most likely serviced from the buffer cache.

$$EU_t(i) = \{u | L_t(u) \leq \frac{B_t(i)}{|U_t(i)|} \; and \; u \in SU_t(i)^c\} \tag{1}$$

Given all these, we define the *looping reference indicator* $\delta_t(i)$ of file i at time t as in Eq. 2.

$$\delta_t(i) = \frac{\int_{t-\theta}^{t} |EU_s(i)| ds}{\int_{t-\theta}^{t} |EU_s(i)| ds + \int_{t-\theta}^{t} |SU_s(i)| ds} \tag{2}$$

Large $\delta_t(i)$ means that there are many effective loop requests to the file i and thus data blocks in loop area can be served from buffer cache. On the other hand, small $\delta_t(i)$ implies that relatively larger fraction of the users are accessing the file i in sequential fashion or non-effective loop requests are often occurred.

In Eq. 2, θ is the update interval for δ. For every θ second, system collects $EU_s(i)$ and $SU_s(i)$ during past θ second and recomputes δ. To analyze the reference patterns more elaborately, we can distinguish the update interval θ and the window length $\mathcal{W}$. System can recompute the looping reference indicator for every θ seconds based on the past $\mathcal{W}$ sec's samples.

3.2 Buffer Management Method

The proposed ABM scheme applies buffer replacement policy per file basis. When the file is first referenced, the ABM applies the DISTANCE policy to that file. This is based on the assumption that streaming workload normally accesses the file in sequential fashion. The ABM scheme monitors the looping reference indicator of the files periodically. When the looping reference indicator of the file becomes large, the ABM switches to the LRU scheme for that file.

As a selection criteria to determine which of the two buffer cache replacement policies, the ABM uses the threshold value δ^*. If the looping reference indicator of file is smaller than δ^*, the DISTANCE policy is applied to that file. Otherwise, the LRU policy is applied.

Figure 1 shows the architecture of the ABM cache manager. The ABM cache manager consists of three components: system buffer manager, DISTANCE manager and LRU manager. The system buffer manager executes the ABM algorithm and controls the allocation of buffer space.

The system buffer manager computes the looping reference indicator of opened files and compares them with threshold value, δ^*. If the looping reference indicator of a file is greater than or equal to δ^*, then the file is allocated

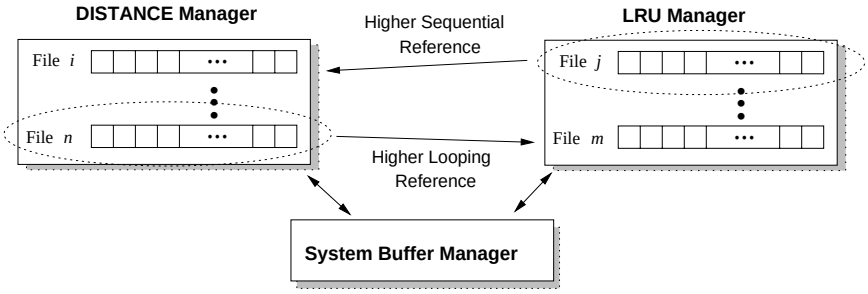


Fig. 1. Structure of Adaptive Buffer Cache Management Scheme

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 transferred to the DISTANCE manager.

It is very important to assign appropriate amount of buffers to each category, elaborate modelling of which is rather complicated. In this subsection, we propose a method of buffer allocation to each manager. The DISTANCE manager and the LRU manager periodically request for additional buffer pages to the system buffer manager. The system buffer manager examines whether it can acquire the buffer pages from the other manager. The system buffer manager maintains information about the total amount of buffer pages allocated to each manager and the respective number of users. The system buffer manager computes the amount of data blocks per user for each manager. The basic idea is to evenly allocate the data blocks to individual streaming sessions. When the DISTANCE manager requests for additional buffer pages, the system buffer manager examines if the users controlled by the DISTANCE manager is allocated smaller number of pages than the users in the LRU manager's control. If so, the system buffer manager requests for the buffer pages to the LRU manager. And then the LRU manager selects the buffer pages using its replacement policy and transfers them to the DISTANCE manager.

4 Simulation Study

In this section, we present the simulation results of legacy buffer cache replacement schemes and verifies the effectiveness of the ABM scheme. We use the synthetic workload. A loop is characterized by three attributes: interval between loops(IBL), length of loop, 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) length of single iteration is 50 blocks and (iii) loop is repeated

for 3 times. Interarrival times of clients are exponentially distributed and average interarrival time is 100 seconds. We assume that every client consumes one block per service round with same playback rate. 20 media files are serviced with an access skew parameter of 0.271 and each file has a length of 60 minutes. The prime performance metric is buffer cache miss ratio. Simulation took more than 8 hours to obtain various numbers: the number of cache hits, the number of block requests and etc.

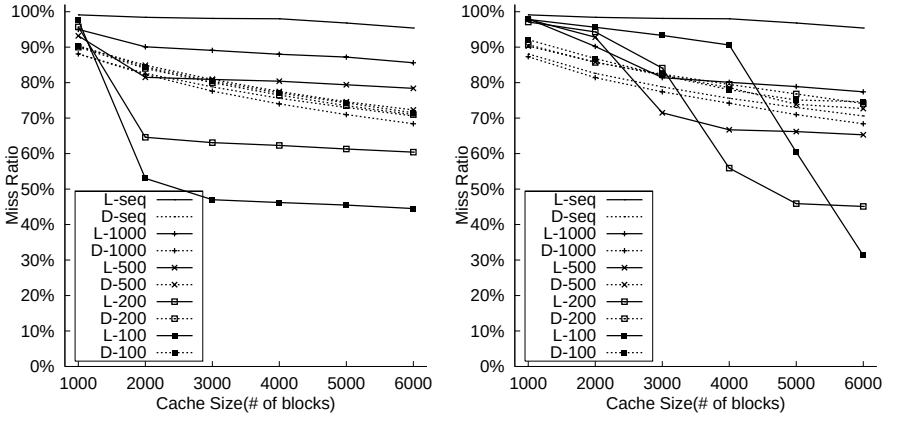
4.1 Performance Analysis: DISTANCE and LRU

We vary the average interval between the loops to examine the effect of frequency of looping accesses. Figure 2(a) and 2(b) illustrate the effects of varying IBL on the cache misses when the average length of loop are 20 seconds and 40 seconds, respectively. The loop count is set to 5. In the legend of figures, symbols L and D represent LRU and DISTANCE respectively. Numbers shown in the legend represent IBL. Let us look into the graph more closely. With sequential reference pattern, the miss ratios of LRU range from 95% to 99% and remain high even if the cache size increases. The miss rate of DISTANCE is much lower than LRU and decreases as the cache size increases. There is not much difficulty in finding that the DISTANCE is better choice for buffer replacement scheme under sequential reference pattern.

When sequential and looping reference co-exist, miss rate of LRU changes depending on IBL, the length of loop and the cache size. In Fig. 2(a), when IBL is greater than 500 and the cache size is 6000, the miss ratio of LRU is higher than that of DISTANCE. On the other hand, when IBL is 100, the LRU scheme exhibits lower miss ratio at 30% compared to the DISTANCE scheme. However, it is noted that when the cache size is relatively small(for instance, the cache size is 1000), the miss ratio of LRU is higher than that of DISTANCE. This is because the loops become *ineffective* when the cache size is small. As the cache space is big enough to accommodate the blocks accessed by looping, the miss ratios of LRU decrease rapidly and are lower than DISTANCE (See Fig. 2(b)). In summary, the LRU scheme outperforms the DISTANCE scheme when there exist effective looping reference patterns in continuous media streams.

Fig. 3 illustrates the effects of varying the length of loop from 20 blocks to 100 blocks. Average interval between loops is 100 seconds and loop count is 5. Numbers in legend of the figure represent the size of cache. The figure shows that varying the length of loop has little impact on the DISTANCE scheme, but has much impact on the LRU scheme. The LRU yields lower miss ratio as the length of loop decreases. This figure also shows that the cache size has significant impact on the miss ratio especially when the cache is managed by the LRU scheme. When the cache size is large enough to hold the entire loop, the LRU scheme exhibit better cache miss ratio.

Fig. 4 illustrates miss ratios with varying the loop count, 3, 5, and 10, respectively. The average length of loop is 20 blocks and the size of cache is 4000. Numbers appeared in the legend represent IBL. The miss ratios of LRU drop as the loop count increases.



(a) loop length= 20 (b) loop length=40

Fig. 2. Effects of varying the frequency of looping reference via Interval Between the Loops factor

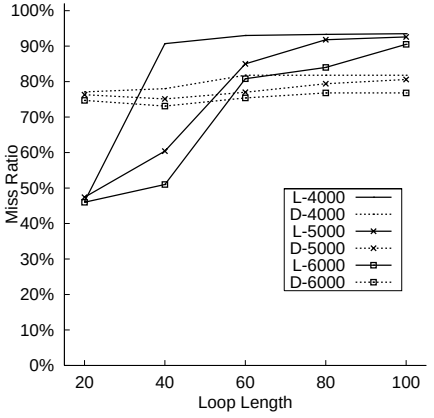


Fig. 3. Effects of the loop length

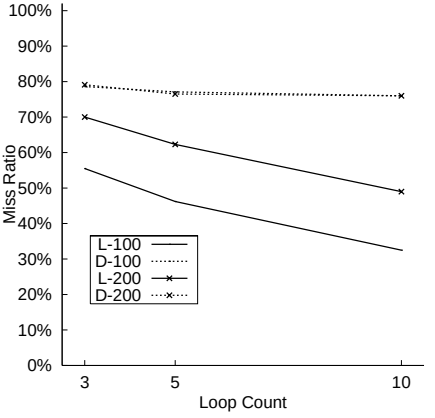
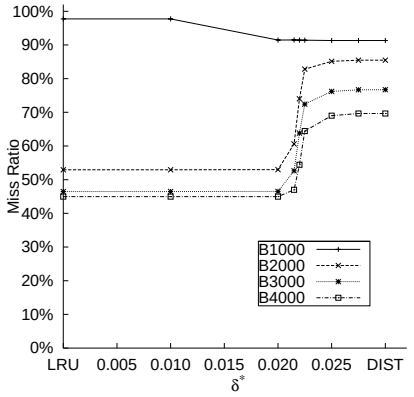


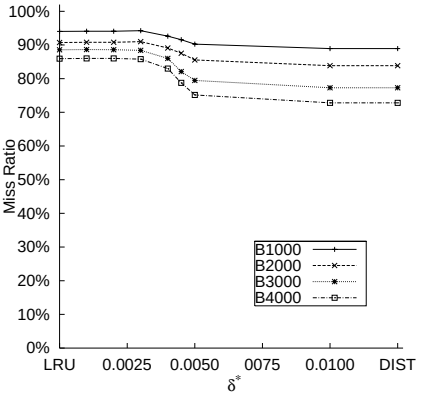
Fig. 4. Effects of the loop count

4.2 Performance of Adaptive Buffer Cache Management Scheme

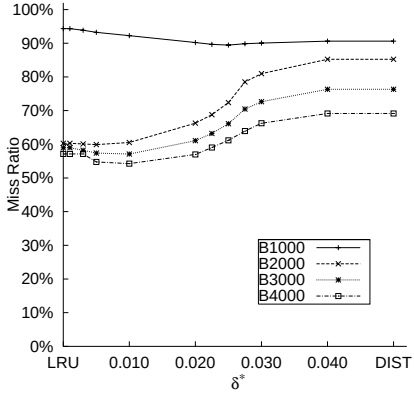
In order to investigate the performance of the ABM scheme, we generate a number of different looping reference patterns for each file. There are 20 files in total. Figures in Fig. 5 describes the various workload used in our experiments and illustrate the respective simulation results. **IBL**(i) denotes the average interval between loops of clients accessing file i. N and $L(i)$ denotes the total number of



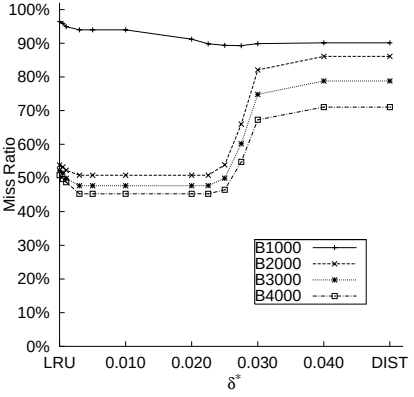
(a) LDT1: **IBL**(i) = 100, for all files (1 ≤i≤ 20)



(b) LDT2: **IBL**(i) = 1000, for all file (1 ≤i≤ 20)



(c) LDT3: **IBL**$(1) = 50$, **IBL**$(i) =$ **IBL**$(1) * 1.2^{(i-1)}$



(d) LDT4: **IBL**$(1) = 50$ and $L(i) = 3600$. If $i \leq \frac{N}{2}$, **IBL**$(i) =$ **IBL**$(1) * 1.2^{(i-1)}$. Otherwise, **IBL**$(i) = \frac{L(i)}{1.2^{(N-i)}}$

Fig. 5. Performance of ABM

files and the length of file i, respectively. In all LDT, we set the average length of loop and the loop count to 20 blocks and 5 times, respectively.

Fig. 5(a) summarizes the result of experiment with LDT1. Be reminded that when IBL is 100(Fig. 2(a)), the LRU scheme exhibits better performance than DISTANCE scheme. Let us examine the cache size of 4000 blocks. If δ^* is smaller

than 0.02, The ABM applies the LRU policy to all files. Hence, the miss ratio of ABM becomes the same as that of LRU. If δ^* is greater than 0.025, the ABM applies the 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 0.025.

In LDT2, $\mathbf{IBL}(i)$ for all file is set to 1000. In this case, the ABM should select the DISTANCE policy to all files because the DISTANCE outperforms the LRU as shown in Fig. 2(a). However, if we use very small threshold value, the ABM may execute the LRU policy to the files even though the files have few looping references. In Fig. 5(b), when δ^* is smaller than 0.01 and the cache size is 4000, the miss ratio of ABM 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, the ABM applies the LRU policy to files whose looping reference indicator is greater than δ^* and applies the DISTANCE policy to other files. In fig. 5(c), consider the size of cache is 4000. the ABM can outperform than both LRU and DISTANCE if it uses 0.01 as δ^*.

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, the ABM tries to apply the LRU policy to the first group's files and the DISTANCE policy to the second group's files. In fig. 5(d), the ABM 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 Adaptive Buffer cache Management(ABM) that detects the *effective* looping reference pattern and adaptively applies appropriate replacement policy. It is very important to have the right metric which effectively quantifies the characteristics of workload. We develop a metric called *looping reference indicator* to denote the *degree* of looping reference. The ABM 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 reveals a number of interesting phenomenon. We observed that the LRU policy can yield lower cache miss rate than the DISTANCE policy if effective looping reference constitutes 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 ABM 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 ABM scheme changes the buffer cache management algorithm dynamically depending on the ongoing workload characteristics and henceforth is possible to deliver superior buffer cache miss ratio. The dynamic and intelligent nature of the ABM manifests itself when the underlying workload exhibits both sequential and looping reference access pattern.

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. E. G. Coffman, Jr. and P. J. Denning. *Operating Systems Theory.* Prentice–Hall, Englewood Cliffs, New Jersey, 1973.
3. 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.
4. Asit Dan and Dinkar Sitaram. Buffer Management Policy for an On-Demand Video Server. Technical Report IBM Research Report RC 19347, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598, 1993.
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. 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.
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. Lawrence A. Rowe, Diane Harley, and Peter Pletcher. Bibs: A lecture webcasting system. Technical report, Berkeley Multimedia Research Center, UC Berkeley, June 2001.
9. Youjip Won and Jaideep Srivastava. "smdp: Minimizing buffer requirements for continuous media servers". *ACM/Springer Multimedia Systems Journal*, 8(2):pp. 105–117, 2000.