

A Generalized Interval Caching Policy for Mixed Interactive and Long Video Workloads

Asit Dan, Dinkar Sitaram
IBM T.J.Watson Research Center
Hawthorne, NY

e-mail:{[@watson.ibm.com](mailto:asit,sitaram)}

Abstract

In a video server environment, some video objects (e.g., movies) are very large and are read sequentially. Hence it is not economical to cache the entire object. However, caching random fractions of a multimedia object is not beneficial. Therefore, traditional cache management policies such as LRU are not effective. The sequential access of pages can be exploited by caching only the intervals between two successive streams on the same object, i.e., by retaining the pages brought in by a stream for reuse by a closely following stream and subsequently discarding them. In contrast to the movie-on-demand workload, an interactive workload is composed of many short video clips (e.g., shopping). Hence, concurrent access to the same video clip will be infrequent and interval caching policy will not be effective. In this paper, we propose a *Generalized Interval Caching policy* that caches both short video objects as well as intervals or fractions of large video objects. The efficacy of this technique for reducing disk overload and hence to increase the capacity of the video server is studied.

1 Introduction

In a video server environment, to guarantee continuous delivery of video streams to the clients sufficient resources need to be reserved on the server [15, 6]. For an environment with a large number of users, the required disk bandwidth can be very high. With falling memory prices, caching of video data can be cost-effective in reducing the disk bandwidth requirement. Two types of applications may co-exist in such environments: interactive and movie-on-demand. An interactive application consists of many small video

clips that are displayed in response to client commands. In contrast, under a movie-on-demand application a single long video may be watched for a long duration (say two hours). Hence it is necessary to develop caching techniques that handle both large as well as short video objects. In addition, such a policy should adapt quickly to a change in workload. Hence, fixed allocation of cache to different objects will not be effective.

Traditional cache management policies employed by various software systems are based upon the concept of a *hot set* of data which is much smaller in size than the total set of data [3, 14]. Generalizing the idea of a hot set to the video-on-demand environment (i.e. storing the most frequently accessed videos in the cache) may not be very useful, since multimedia objects (e.g. movie) may be very large. Hence, caching even a small number of popular movies will require a very large memory space. Caching policies which operate at the block level (e.g. single page) and caches unrelated set of blocks (e.g., LRU) can not guarantee continuous delivery of streams.

For movie-on-demand and other applications with large video objects, we proposed an *Interval Caching Policy* that caches only the small intervals between successive streams [5]. The pages brought in by a preceding stream are retained in the cache for reuse by a following stream being served from cache. By caching only the smallest intervals from a large sample of intervals, the policy exploits the statistical variation in interval size. Such a policy is shown to be superior, in terms of the cache hit ratio, to the policy of caching the hottest movies. In addition, the policy dynamically adapts to the changing frequency of access to different movies unlike the static caching policy. Interval caching policy, however, is not directly applicable for an *interactive workload* consisting of small videos. This is because concurrent accesses to small video objects are infrequent and hence, there are very few intervals that can be cached.

We extended the interval caching policy to handle any mix of interactive and long video workloads. The new policy is referred to as the *Generalized Interval Caching (GIC)* policy since it is based on the generalization of the concept of interval. For the short video objects where the concurrent accesses are unlikely, the interval is defined as the time between successive accesses to the same object, and is referred to as the *predictive interval*. The policy still caches the

shortest intervals whether the interval encompasses the entire video object or just a segment as in the interval caching policy. This uniform approach to caching allows the GIC policy to handle both interactive and movie-on-demand applications while retaining the benefits of interval caching such as superior performance and adaption to changing access frequency.

Earlier works in multimedia buffer management have studied the buffer requirements for various disk scheduling algorithms [15, 1]. In these works, the video data block is discarded once it has been transmitted to the clients and is not retained for subsequent re-use by other clients. The idea of using alternatives to LRU and CLOCK for sequential access patterns has been used in several contexts in database environments [18, 11, 2, 17, 16]. In a general database environment, various queries and transactions follow different access patterns [2, 17, 4]. The concept of query hot set is exploited in [2, 17, 12] for allocation of cache to various queries. Various prefetching strategies based on the dynamic identification of sequential access have been used in [16, 4]. The objective in all of the above work has been to minimize the overall response time and cache miss probability. Hence these algorithms may not provide continuous delivery of a stream from the cache. The algorithm that is closest to the current work is [16] which tries to retain the page that will be accessed earliest for concurrently executing queries in a batch environment. The objective of the algorithm is to minimize the total completion time of all jobs and hence to be fair in terms of cache miss to all jobs. Note that this does not translate into a continuous delivery guarantee.

Research Contributions: The contribution of this paper is twofold. First, we propose a model for *interactive workloads* that can be used to study system performance under interactive applications, e.g., medical, shopping, etc.

We next propose a stream dependent caching policy (as opposed to traditional block level caching policy) for both short video clips as well as long videos. In a mixed workload environment, where both long videos and short clips are present [8], we use the notion of predictive intervals to extend the basic interval caching policy that we had proposed in [5]. The basic policy was shown to be adaptable to load change in [5] and cost-effective in [10] for long video workload.

This extended policy is referred to as the *Generalized Interval Caching Policy*. Here, we also provide an overview of the GIC policy.

The remainder of this paper is organized as follows. In section 2, we discuss client behaviour, the types of applications studied and the server environment. We discuss the details of the GIC policy in Section 3. In section 4, we study the performance of this policy using a detailed simulation. Finally, we summarize our results in section 5.

A more detailed version of this paper [7] describes the implementation of the GIC policy and demonstrates its cost-effectiveness.

2 Video Applications

Two different types of applications may be possible in video-on-demand servers. In an interactive application, short video clips are displayed in response to frequent client commands. An example is a shopping application where a video catalog consisting of short product advertisements may be browsed by clients. Alternatively, in a movie-on-demand applications clients view long running movies (typically 1-2 hours long) and client commands are infrequent once the selected movie starts playing (e.g. occasional pause/resume). Since the performance of the GIC policy will depend on the type of application, both types of applications are considered in the paper.

2.1 Interactive Application

Under an interactive application, the server contains a video catalog consisting of short clips of varying length. The interactive application workload is characterized by the following three parameters:

Access skew across the clips. We will use the notation $x - y$ to represent that x fraction of the viewing requests go to the y fraction of the clips in the database.

Clip length distribution. The length of any clip is assumed to be distributed between L_{min} and L_{max} .

Viewing time per individual clip. Many interactive applications may use *video banners* consisting of a short clip that is displayed repeatedly (e.g. a logo that is shown until the client makes a selection). To model this kind of behaviour, each clip is assumed to be viewed for a random viewing time. Viewing time is assumed to be independent of the length of a clip, and can be smaller than the playback duration of an entire clip. Therefore, general viewing time, T_v , is modeled as a random variable.

Client session duration. Each client session is modeled as a set of clips selected using the above parameters and the session duration is assumed to be of length $T_{session}$. The client selects playing of a new clip until the session duration exceeds $T_{session}$.

2.2 Movie-on-demand application

In a pure movie-on-demand application, clients may choose movies according to an access distribution. The access distribution is based on the empirical data on video rentals in various video stores during a particular week [19]. The empirical data can be fitted using a Zipf distribution over 92 movies with the parameter 0.271 [5]. In our paper, the Zipf distribution is used to generate accesses to video files. Each client session is assumed to be the playback of a single movie.

A mixed workload environment is modeled by randomly classifying a client during the session open time as interactive or movie-on-demand clients. A control parameter called the *interactive probability* determines the ratio of interactive and movie-on-demand sessions. The client arrival process is modeled as a Poisson process with a mean inter-arrival duration of T seconds.

2.3 Server Environment

A video server may contain various software processes such as a disk manager that reads data from disk, a communication manager that transmits data to clients, and a cache manager that determines which blocks are to be retained in the cache [5]. The cache space is subdivided into a number of blocks, and the

blocks containing valid data are in the cache pool while the rest of the blocks are in the free pool. During normal operation, for each data stream the disk manager acquires a free block, inserts it into the cache pool and starts the disk I/O to read data into that block. The communication manager on the other hand sends previous data blocks of a stream stored in the cache to the corresponding client process.¹ In the absence of any long term caching, at the completion of the communication I/O for each data block, the data in that block is discarded by returning the block to the free pool. Hence, only two cache blocks per stream are required.

The data brought in by a stream however can be reused by other closely following streams, if sufficient cache space is available to retain the data blocks in the cache. Upon completion of the communication I/O for each data block the cache manager is invoked to decide whether the block should be returned to the free pool or retained in the cache pool for reuse by other streams. Note that the processes share blocks by the exchange of pointers and not by copying data. We next describe the GIC policy and detail the implementation issues.

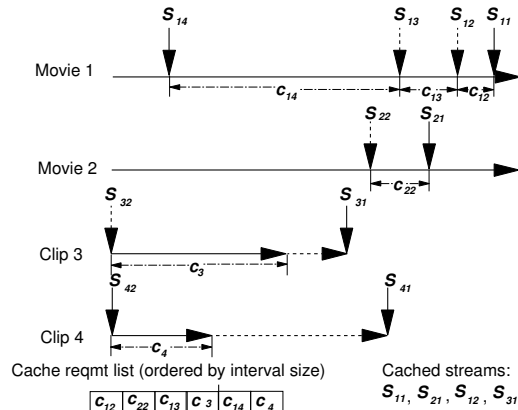


Figure 1: Generalized Interval Caching policy

3 Overview of the Proposed Policy

The main idea behind the proposed policy is illustrated using Figure 1. The small arrows marked by

¹The same data may be sent to multiple client processes if playback of the video by multiple clients are synchronized [6].

S_{11} through S_{42} represent the pointers corresponding to the various playback streams on the large movies 1 and 2, and video clips 3 and 4. In the interval caching policy [5], two streams, S_{ij} and S_{ik} accessing the same video object are defined as *consecutive* if S_{ik} is the stream that next reads the data blocks that have just been read by S_{ij} ; and a pair of consecutive streams is referred to as an *interval*. The two streams of an interval are referred to as the *preceding* and the *following* streams. 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 policy 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 the shortest intervals.

Figure 1 illustrates how the definition of an interval is extended to small video objects that are not being accessed concurrently. Arrows S_{32} and S_{42} represent two streams presently reading small video objects 3 and 4. Arrows S_{31} and S_{41} represent the position that would have been reached by the previous streams accessing objects 3 and 4 if the objects had been larger. (S_{31}, S_{32}) , and (S_{41}, S_{42}) are defined to form two intervals for objects 3 and 4 even though streams S_{31} and S_{41} have already terminated. The *interval size* in this case is defined to be the time-interval between two successive accesses on the same object. However, the cache requirement is smaller than the interval size times the data rate and is equal to the size of the object. Therefore, if the interval is selected for caching, the entire video object will be cached. Note that at the time of allocating cache to a stream there is no concurrent following stream on that object. The sizes of the previous intervals provide an indication of the anticipated interval size for that object. In this paper, we will assume the size of the last interval as the anticipated interval size.

The GIC policy thus caches intervals which may be either video segments or entire video objects. The policy orders all intervals (current or anticipated) in terms of increasing interval size (see, Figure 1). As before, it then allocates cache to as many of the intervals as possible. Hence, with a larger number of video segments and video objects, the GIC policy still maximizes the number of streams served from the cache.

The ability to choose a small set of intervals from a large set of samples allows the policy to exploit the variation in cache requirements across all intervals due to size, the variation in inter-arrival times and/or different compression rates for different videos.

Changes to the interval list may occur due to the arrival of a new stream or termination of a current stream. Therefore, the algorithm for reordering of the interval list and the allocation and deallocation of cache to intervals is executed only at the time of arrival or ending of a playback stream (the finer details of selecting the intervals to be cached and switching of a stream from cache to disk and vice versa are discussed in later subsections). Since only simple modifications are required when opening or closing a video file, the policy is simple to implement.

OPEN:

```

Form new interval with previous stream;
Compute cache requirement
    and interval size;
Reorder cache requirement list;
If not already cached
    If space available,
        Cache this new interval;
    else if this interval is smaller
        than existing cached intervals
        and sufficient cache space
        can be released
        Release cache space from
            larger intervals;
        Cache this new interval;

```

Figure 2: Details of OPEN in generalized interval caching policy

3.1 Details of implementation

Figure 2 shows the actions taken when an open request is received from an user. As described earlier, a new interval is formed by the new request and the preceding stream. The cache requirement and interval size of this new interval are computed. Successive requests to a small video object may result in all the blocks of the object being retained in the cache. Hence, if the open request is for a small video object, the interval may already be in the cache.

If the interval is not already cached, the cache manager determines if it is desirable to cache the interval as follows. Since the cache manager attempts to retain the blocks with the lowest interval size, it determines the total cache space available in intervals with a larger interval size and the free pool. If this space is larger than the cache requirement of the new interval, the new interval is allocated cache space from the free pool as well as the intervals with the largest interval sizes. In the case where the free space by itself is sufficient, the cache manager need not release cache from intervals with larger sizes. The decision to cache an interval has different results depending on whether the interval is formed by two concurrent streams or not. In the former case, blocks brought in by the preceding stream are retained in the cache and used to serve the new request. Caching an anticipated interval results in retaining the blocks brought in by the new request for use by a later stream.

CLOSE:

```

If following stream of an interval
Delete the interval;
Free allocated cache;
If next largest interval can
    be cached
    Cache next largest interval;

```

Figure 3: Details of CLOSE in generalized interval caching policy

Figure 3 summarizes the steps needed during when the client finishes viewing a video. When a close request is received, the interval in which the client was the following stream is deleted. If the interval is a segment of a video object, it is unlikely that there will be any benefit to retaining it in the cache. Hence the cache management process releases the space being held by the interval. The cache management process then considers the interval with the smallest interval size to determine if it should be allocated cache using the same algorithm used during the arrival of a new stream.

3.2 VCR control

Video-on-demand systems may allow clients to pause and restart viewing at arbitrary times. At such times, the stream dependencies will change. The problem of

providing VCR control capabilities exists even in the absence of a caching policy, since the resources released during a pause may not be available in general at the time of a resume request. In [6] it is proposed that pause and resume requests be handled by setting aside a small pool of channels called *contingency channels*. The method provides a *statistical guarantee* that with high probability (e.g. 99%) a resume request can be serviced within a small pre-specified delay. It is shown in [9] that this method is more efficient than reserving resources for all paused streams. Details of an integrated GIC policy under VCR control can be found in [7].

4 Simulation and Results

In [5], it is shown that the analysis of the interval caching policy can be related to the general problem of obtaining order statistics and that obtaining explicit expressions may be quite a difficult task. Hence, simulation is used to study the effectiveness of the GIC policy. In the simulation, clients made random requests to a server. The server disks were assumed to be striped together into a single striping group with a pre-specified bandwidth. The simulation modelled the change in disk load due to arrival and ending of client requests, or switching between cache and disk. Requests for videos that would result in overloading of disk bandwidth are rejected. However, individual block I/Os are not modelled. The cache is modelled as a collection of 1MB blocks. While computing the amount of cache space required for a cachable segment, the computation is rounded up to the nearest megabyte. In order to estimate the length of the simulation, trial runs are made using the method of batch means. It is found that with a simulation duration of 8 hours, after ignoring the initial transient (1.5 hours), 99% confidence intervals of less than 10% are obtained for important quantities such as the average number of streams reading from the cache. Thus the simulations are run, in general, for 8 hours.

4.1 Workload Parameters

The client inter-arrival time, T , is assumed to be 6.25 seconds corresponding to a server simultaneously serving on an average 400 streams. This value is typical of the range of active clients expected for video-on-

demand servers [13]. The mixed workload is modeled by varying the control parameter *interactive probability*. Unless mentioned otherwise, this parameter is taken to be 0.8. The duration of an interactive session, T_s , is assumed to be 30 minutes. Therefore, 50% of all I/O requests are made by the interactive sessions. All videos are assumed to be MPEG-1 with a delivery rate of 192 KB/s. The length of the movies is assumed to be approximately ninety minutes long so that the total amount of storage required per movie is one GB. The movie library consists of 92 movies with an access skew parameter of 0.271.

For interactive applications, in the base case, a video catalog of 500 MPEG-1 video clips with viewing time uniformly distributed up to 30 seconds was stored on the server. Thus in the base case, the total size of the video catalog was 125 minutes. The number of video clips was varied in the simulation to study its impact on the server capacity. A clip access distribution of 80-20 was studied. The length of a clip is uniformly distributed between 1 second and 30 seconds. For modelling video banners, each clip was viewed for an exponentially distributed viewing time with an average of 30 seconds. However, the minimum viewing time of a clip is set to 5 seconds. Experiments varying other parameters are reported in [7].

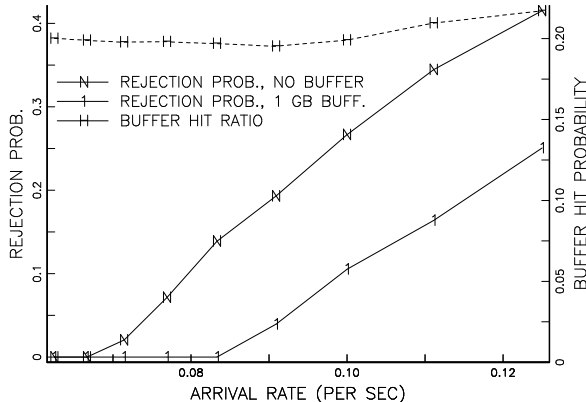


Figure 4: Rejection probability; 400 stream disk capacity

4.2 Admission Control

In the presence of caching, the number of streams that can be served by a video server depends on the number of streams that can be served from the cache. The capacity of a system can be measured as the request rate that can be supported with no or a small

rejection probability. Figure 4 shows the rejection probability as a function of the arrival rate for a system with a disk capacity of 400 streams. The two curves are for a system without any cache and a system with 1 GB cache. In both cases, the rejection probability is 0 initially and then rises approximately linearly. Considering the arrival rates where the rejection probabilities become non-zero (.067/sec and .085/sec) it can be seen that the effect of caching is to increase the system capacity by 25%. Figure 4 also shows the corresponding overall cache hit ratios, defined as the number of reads served from the cache as a fraction of the total number of reads. From the graph it can be seen that the hit ratio is 0.22 which translates to a 25% increase in system capacity. Hence, in the following results, only the cache hit ratio is studied since the increase in system capacity due to caching is proportional to the cache hit ratio.

4.3 Cache hit ratios

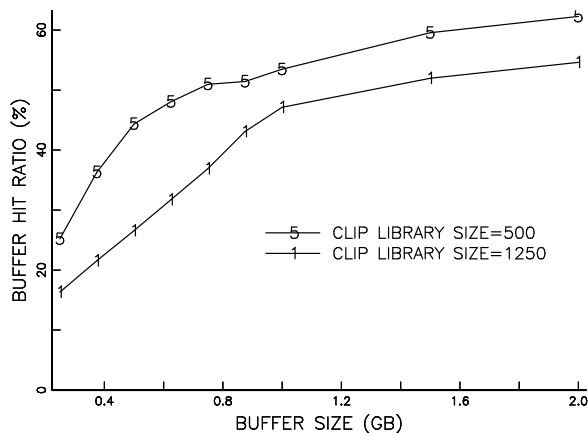


Figure 5: Overall cache hit ratio for mixed workload

In the following, we show that high hit ratios are possible with the generalized interval caching policy under a wide range of variation in the workload parameters. Figure 5 shows the variation of the overall cache hit ratio as a function of the cache size for the base mixed workload. The two curves are for an interactive library size of 500 clips and 1250 clips. For a given cache size, the overall hit ratio with 500 clips is higher than that with 1250 clips since with a smaller number of clips, a larger fraction of the accesses falls on the clips retained in the cache. Figure 6 shows the hit ratios of the individual components of the

workload. The clip hit ratio is defined as the fraction of clip reads satisfied from the cache. Similarly, the movie hit ratio is the fraction of movie reads serviced from the cache. The access to the short videos are more likely to be found in the cache, and only the shortest intervals of long videos are satisfied from cache. However, both clip and movie hit ratios are smaller for the workload with 1250 clips.

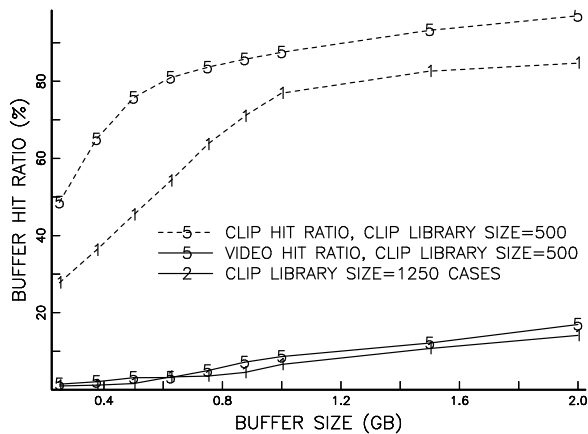


Figure 6: Interactive and movie cache hit ratios for mixed workload

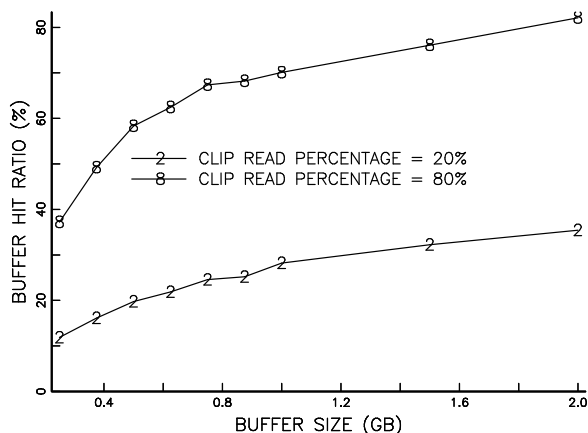


Figure 7: Effect of varying interactive fraction on overall cache hit ratio

The variation of the overall cache hit ratio with the cache size for different fractions of short video clips is shown in Figure 7. The interactive library size contains 500 clips as in the base workload. The two curves shown are for workloads where the fraction of accesses to the short clips are 20% and 80%, respectively. The cache hit ratio is much larger for the workload with the read percentage to interactive

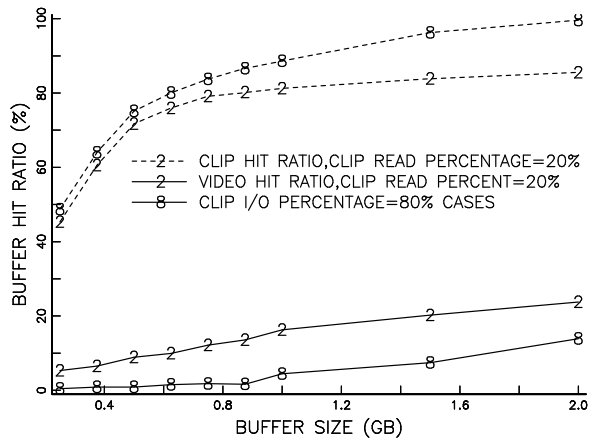


Figure 8: Effect of varying interactive fraction on interactive and movie cache hit ratios

clips as 80%, since by caching the hot clips in a small amount of cache, a larger fraction of the accesses can be served from the cache. Figure 8 shows the hit ratios for the interactive clips and the movies separately for the two different values of read percentage to interactive clips. The dotted lines are for the interactive clips while the solid lines are for the movies. In both cases, the cache hit ratio for the clips is higher than that for the movies. However, the clip hit ratio for the 80% read case is higher than that for the 20% read case while the relationship is reversed for the movie hit ratios. This is because with decreasing clip access frequency, a smaller number of hot clips are retained in the cache, leading to a lower clip cache hit ratio and a higher movie cache hit ratio.

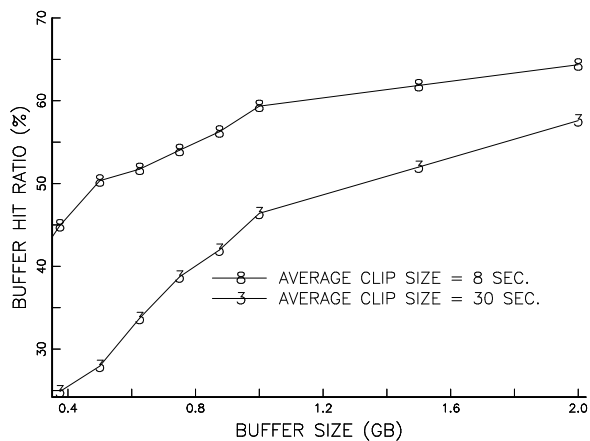


Figure 9: Effect of varying average clip size on overall cache hit ratio

Figure 9 plots the cache hit ratio against the cache size for two different values of average clip size (8 seconds and 30 seconds). The cache hit ratio with an average clip size of 8 seconds is higher than that for an average clip size of 30 seconds, since less cache is required when the clip size is smaller. At large cache sizes, however, the difference in hit ratio is reduced, since in both cases most of the hot clips can be stored.

5 Conclusions

In this paper, we proposed a model for interactive workload consisting of short video clips. We then used the notion of predictive intervals to extend the basic stream dependent caching policy (*Interval Caching* policy [5]) for handling mixed workloads consisting of long movies and short interactive clips. The extended policy is referred to as the *Generalized Interval Caching (GIC)* policy. The implementation of GIC policy requires only a small modification to the video server access path during start, resume, pause or stop of videos. Hence, it is easy to implement in any existing multimedia server. Since analytical modelling of the policy is intractable, the performance of the generalized interval caching (GIC) policy is studied by simulation. A mixed workload consisting of an interactive application and a movie-on-demand application was used in the simulation. It is shown that the GIC policy has high cache hit ratios with small size of cache with varying mixes of the two application types and with varying parameters of the workloads.

References

- [1] Chen, M., D. Kandlur, and P. Yu, "Optimization of the Grouped Sweeping Scheduling (GSS) with Heterogenous Multimedia Streams", *Proc. ACM Multimedia 93*, Anaheim, CA, Aug. 1993, pp. 235-242.
- [2] Chou, H. T., and D. J. Dewitt, "An Evaluation of Buffer Management Strategies for Relational Database Systems," *VLDB Conf.*, Stockholm, Sweden, 1985.
- [3] Dan, A., and D. Towsley, "An Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes," *ACM SIGMETRICS*, Denver, CO, May 1990, pp. 143-152.
- [4] Dan, A., P. Yu, and J. Y. Chung, "Characterization of Database Access Patterns for Analytic Prediction of Buffer Hit Probability", *The VLDB Journal*, Vol. 4, No. 1, January 1995, pp. 127-154.
- [5] Dan, A., and D. Sitaram, "Buffer Management Policy for an On-Demand Video Server", *IBM Research Report, RC 19347*, Yorktown Heights, NY, 1993.
- [6] Dan, A., D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching", *Proc. ACM Multimedia 94*, San Francisco, CA, October, 1994. An extended version to appear in *ACM Multimedia Systems*.
- [7] Dan, A., and D. Sitaram, "A Generalized Interval Caching Policy for Mixed Interactive and Long Video Workloads", *IBM Research Report, RC 20206*, Yorktown Heights, NY, 1995.
- [8] Dan, A., and D. Sitaram, "An Online Video Placement Policy based on Bandwidth to Space Ratio (BSR)", *Proc. SIGMOD'95*, San Jose, May 1995.
- [9] Dan, A., P. Shahabuddin, D. Sitaram, and D. Towsley, "Channel Allocation under Batching and VCR Control in Video-On-Demand Servers", *Journal of Parallel and Distributed Computing*, Vol. 30, No. 2, November 1995, pp. 168-179.
- [10] Dan, A., D. M. Dias, R. Mukherjee, D. Sitaram and R. Tewari, "Buffering and Caching in Large-Scale Video Servers", *IEEE CompCon'95*.
- [11] Effelsberg, W. and T. Haerder, "Principles of Database Buffer Management", *ACM Trans. Database Systems*, Vol. 9, No. 4, Dec. 1984, pp.560-595.
- [12] Faloutsos, C., R. Ng, and T. Sellis, "Predictive Load Control for Flexible Buffer Allocation," *VLDB Conf* Barcelona, Spain, 1991, pp. 265-274.
- [13] *Electronic Engineering Times*, March 15, 1993, pp 72.
- [14] Nicola, V. F., A. Dan, and D. M. Dias, "Analysis of the Generalized Clock Buffer Replacement Scheme for Database Transaction Processing", *ACM SIGMETRICS*, 1992, pp. 35-46.
- [15] Rangan, P. V., H. M. Vin, and S. Ramanathan, "Designing an On-Demand Multimedia Service," *IEEE Communication Magazine*, Vol. 30, July 1992, pp. 56-65.
- [16] Rahm, E., and D. Ferguson, "Cache Management Algorithms for Sequential Data Access" *IBM Research Report, RC15486*, Yorktown Heights, NY, 1993.
- [17] Sacco, G. M., and M. Schkolnick, "Buffer Management in Relational Database Systems", *ACM Trans. Database Systems*, Vol. 11, No. 4, Dec. 1986, pp. 473-498.
- [18] Teng, J. Z., and R. A. Gumaer, "Managing IBM Database 2 Buffers to Maximize Performance", *IBM Systems Journal*, Vol.23, No.2, 1984, pp.211-218.
- [19] *Video Store Magazine*, Dec. 13, 1992.