Buffer Management Policy for an On-Demand Video Server

Asit Dan
Dinkar Sitaram

IBM Research Division, T. J. Watson Research Center
Yorktown Heights, NY 10598

ABSTRACT

In an on-demand video server environment, multimedia 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. This is due to the stringent response time requirements where continuous availability of a stream has to be guaranteed; whereas caching random fractions will result in unpredictable load on the disks. Therefore, traditional buffer management policies such as LRU are not effective. In addition, the sequential access implies pages brought in by a stream can be reused by a closely following stream and subsequently discarded, thus buffering only a fraction of the entire object. In this paper, we propose a buffer management policy called the *interval caching policy* based on the above idea that identifies certain streams and temporarily buffers the pages brought in by those streams. We study the efficacy of this technique for reducing disk overload and hence to increase the capacity of the video server. We also show that the interval caching policy makes it cost-effective to use memory to buffer video streams even with a uniform access pattern to all movies.

IBM Research Report RC 19347

1 Introduction

With the advent of digital video technology, it is feasible to provide video-on-demand services to a large number of clients over a geographically distributed network [12, 17, 14]. A commonly proposed architecture for such applications is the client-server environment [7, 11] where clients request playback of the movie of their choice from a set of centralized servers. The servers starts a new playback stream corresponding to each new client request for a movie by retrieving the stored movie from the set of disks attached to the server. Because of the hard real-time constraints on response time, sufficient resources are reserved on the server to guarantee continuous delivery of a video stream to the client [14, 2]. With a large number of users, the required disk bandwidth can be very high. With falling memory prices, buffering of video data can be cost-effective in reducing the disk bandwidth requirement.

Traditional buffer management policies employed by various software systems (e.g. database buffer managers, operating system memory management) are based upon the concept of a *hot set* of data which is much smaller in size than the total address space. Various buffer management policies such as LRU, clock etc. use different mechanisms to identify the hot set and to retain it in the buffer [5, 13]. Note that generalizing the idea of a hot set to the video-on-demand environment may not be very useful. This is because each multimedia object (e.g. movie) is very large. Some movies may be more popular than the others, however buffering even a small number of popular movies will require a very large memory space. Buffering policies which operate at the block level (e.g. single page) do not make sense for the following reasons. First, the unpredictable nature of the buffer hits under an algorithm such as LRU makes it difficult for the server to guarantee continuous delivery of the stream as required for multimedia applications. Additionally, as in the object-level buffering, the buffer hit probability will be very small unless a very large buffer size is used. In this paper, we propose a new buffer management policy called the *interval caching* policy that overcomes both of these constraints.

The interval caching policy is based on the following observation. For any two consecutive requests for the same movie, the later stream can read the data brought into the buffer by the earlier stream if the data is retained in the buffer until it is read by the later stream. Understanding such dependencies makes it possible to guarantee continuous delivery of the later stream with a small amount of buffer. This is possible since with a small amount of dedicated buffer, it can be ensured that the later stream always reads from storage. With a larger number of concurrent streams reading the same movie, it is possible to choose the streams to be retained so as to maximize the number of streams read from the buffer. Note that inter-arrival times are not uniform due to statistical fluctuations. The ability to choose a small set of inter-arrival times from a large set of samples allows the interval caching policy to exploit the variation in the inter-arrival times. In this paper, we will demonstrate that this characteristic allows buffering under the interval caching policy to be cost-effective even when frequency of access to the movies is uniform.

Earlier works in multimedia buffer management have studied the buffer requirements for various disk scheduling algorithms [1, 14, 3, 20]. 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, 9, 4, 16, 15]. In a general database environment, various queries and transactions follow different access patterns [4, 16, 6]. The concept of query hot set is exploited in [4, 16, 10] for allocation of buffer to various queries. Various prefetching strategies based on the dynamic identification of sequential access have been used in [18, 15, 6]. The objective in all of the above work has been to minimize the overall response time and buffer miss probability. Hence these algorithms may not provide continuous delivery of a stream from the buffer. The algorithm that is closest to the current work is [15] 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 buffer miss to all jobs. Note that this does not translate into a continuous delivery guarantee.

The remainder of this paper is organized as follows. In section 2, we describe the details of the interval caching policy. In section 3, we will relate this problem to an order-statistics problem in probability theory and will show the inherent difficulty in solving the problem analytically. Therefore in section 4, we study the performance of the interval caching policy using a detailed simulation. We also compare the performance of our policy against the policy of caching just the popular movies. Finally, we summarize our results in section 5.

2 Environment

In the following section, we outline the server environment and discuss the details of the interval caching policy. Figure 2 shows the various software components of a video server environment, e.g., disk manager, communication manager, buffer manager. The buffer space is subdivided into a number of blocks, and the blocks containing data are in the buffer pool while the rest of the blocks are in the free pool. During normal operation, for each data stream the disk manager initiates a disk I/O in anticipation and reads data for that stream to a free buffer block. It does this by acquiring a free block, inserting it into the buffer pool and starting the disk I/O to read data into that block. The communication manager on the other hand initiates the communication I/O process and sends previous data block of a stream stored in the buffer to the corresponding client process.[1] In the absence of any long term buffering, at the completion of the communication I/O process for each data block, the data in that block is discarded, and

[1] The same data may be sent to multiple client processes if playback of the video by multiple clients are synchronized [7].

the block is returned to the free pool. Hence, only two buffer blocks per stream are required, in the absence of long term buffering.

The data brought in by a stream however can be re-used by other closely following streams, if sufficient buffer space is available to retain the data blocks in the buffer. The completion of the communication I/O process for each data block then invokes the buffer manager to decide whether the block should be returned to the free pool or retained in the buffer pool for re-use by other streams. The blocks in the buffer pool are thus subdivided into *in–blocks* that are in the process of having video data read into them, *out–blocks* from which data is being transmitted to one or more clients and *data–blocks* that are being retained for re–use by other client(s). Note that the processes share blocks by the exchange of pointers and not by copying data. We next describe the interval caching policy and detail the implementation issues.

2.1 Overview of the Interval Caching Policy

The main idea behind the interval caching policy is illustrated using Figure 3. The small arrows marked by S_{11} through S_{31} represent the pointers corresponding to the various playback streams on the movies 1, 2 and 3. Two streams, S_i and S_j are defined as *consecutive* if S_j is the stream that next reads the data blocks that have just been read by S_i. For example, in Figure 3, (S_{11}, S_{12}), (S_{12}, S_{13}) and (S_{13}, S_{14}) form three consecutive pairs for movie 1. The two streams of such a pair of consecutive streams are referred to as the *preceding* and the *following* streams. Understanding such dependencies makes it possible to guarantee the continuous delivery of the later stream with a small amount of buffer. The buffer requirement of a consecutive pair (to serve the following stream from the buffer) is a function of the time-interval between the two streams and the compression method used. Therefore, buffer requirement is different for different pairs. With a larger number of consecutive pairs across all movies, it is possible to choose the consecutive pairs to be buffered so as to maximize the number of streams served from the buffer. The ability to choose a small set of consecutive pairs from a large set of samples allows the disclosed policy to exploit the variation in buffer requirements across all pairs due to the variation in inter-arrival times and/or different compression rates for different movies.

The interval caching policy orders all consecutive pairs in terms of increasing buffer requirements (See, Figure 3). It then allocates buffer to as many of the consecutive pairs as possible. Note that the buffer requirement of a consecutive pair varies with time depending on the compression algorithm used. Therefore, for continuous delivery guarantee, each pair should be allocated its maximum requirement. However, the variance of the total buffer demand for a large number of consecutive pairs is much smaller. In any case, to reduce the possibility of jitter in the playback stream (due to buffer miss), the total allocated buffer is larger than the average total requirement. The expected value of buffer requirement for a consecutive pair is estimated when it is formed due to the arrival (or restart in the presence of VCR control) of a new stream. The ending (or pausing in the presence of VCR control) of a stream may result in the deletion of one or two consecutive pairs.[2] In the later case, a new consecutive pair is formed with the preceding stream of one pair and the following stream of the other pair. When a buffered consecutive pair is deleted, the allocated buffer space is returned to the free pool and subsequently, the available buffer is allocated to the smallest unbuffered consecutive pair if it satisfies its buffer requirement. (The finer details of switching of a stream from buffer to disk and vice versa are discussed in the next subsection.) Therefore, the algorithm for reordering of buffer requirement list and the allocation and deallocation of buffer to various consecutive pairs is executed only at the time of arrival, ending, pausing or restarting of a playback stream. Table 1 summarizes the buffer management algorithm.

[2]Note that a single stream can be part of two consecutive pairs, taking the roles of preceding and following streams, respectively.

Table 1: Buffer Management Algorithm

OPEN	**READ**	**CLOSE**
Form a new pair with its previous stream *Compute its buffer requirement* *Reorder buffer requirement list* *If space available, cache this pair* *else if buffer requirement of this pair is smaller than that of the largest cached pair* *Stop caching largest pair* *Cache this pair*	*Lookup cache* *If not in cache* *Read from disk*	*Delete this pair* *If pair was cached,* *Release its cache* *Cache next smallest pairs if possible*

2.2 Details of Implementation

We now detail the implementation of the interval caching policy. Figure 4 shows the data structure used, where each entry corresponds to a playback stream. The attribute current block id points to the block which is to be read next by the stream. Its value is incremented at the completion of the communication I/O process for each block. Also maintained are a sorted list of estimated buffer requirement, and an array of pointers to the earliest playback stream for each movie (not shown in the figure). When a client starts viewing a movie, an open request is received for the corresponding video file and a new table entry is created. Note that the pointer to the earliest stream of a movie, together with the following stream pointers in the stream information table form a singly linked list of all the streams viewing the movie in the order of arrival (until rearranged through pause and restart). The buffer manager thus locates the previous request (if any) for the same movie. If it exists, then the follow up stream id field of that entry is modified, and a new consecutive pair is formed consisting of the previous stream and the new stream. The buffer requirement of this consecutive pair is also estimated at this time. The buffer manager then inserts the new buffer requirement entry in the sorted list, and decides if any buffer should be allocated to this newly formed consecutive pair, i.e. whether all blocks read by the earlier stream of the pair are to be retained in the buffer for reading by the later stream. The new pair is allocated buffer if there is sufficient free space available, or if there is a buffered pair that requires more space. Similarly, the follow up stream id of the new table entry is set. It is set to null if it is a new request. Otherwise, it is a resumed stream and it forms a second consecutive pair for which it is the preceding stream. The buffer requirement of the new pair is also estimated and inserted in the sorted buffer requirement list. Note also that in the later case an existing consecutive pair relationship is destroyed and the buffer is deallocated from that pair (if any).

Two counters are used to keep track of the amount of free space available in the buffer. The counter ActFreeSpace keeps track of the actual number of free blocks while the other counter NomFreeSpace is the nominal (estimated) value of free space in the buffer. Note that the amount of buffer space consumed by a pair may vary due to data rate fluctuations. By sharing a common pool by all pairs, the statistical variations in the total requirement will be reduced. However, in order to safeguard against the occasional buffer miss due to data rate fluctuation the amount of allocated buffer will be somewhat greater than the total average requirement. Therefore, the NomFreeSpace indicates amount of buffer space that can potentially be allocated. When allocating or deallocating space, NomFreeSpace is adjusted as a function of the total average requirement by all allocated consecutive pairs.

If the newly formed consecutive pairs require less buffer space than existing buffered pairs, then the buffer is deallocated from those pairs with the largest buffer requirements. Neither allocation nor deallocation of buffer to or from a pair is a high–overhead operation. Allocating buffer to a pair consists of resetting the NomFreeSpace counter as a function of total average requirement and setting the read state in the stream table to indicate that the later request of the pair should read from the buffer, not from disk. The two bit read state (see Figure 4) indicates: if the following stream is reading from buffer or disk and if it will be switched to buffer or disk. The state 11 indicates that the buffer is allocated to this pair and the following stream is being served from the buffer. The state 01 however, indicates that the following stream is currently reading from disk, and will be switched to buffer when the blocks are available in the buffer, i.e., the buffer is already allocated to retain this stream. Similarly, the state 10 indicates the buffer is deallocated and the stream will be switched to the disk when the blocks are not available in the buffer. The state 00 indicates the following stream will continue to read from disk. The buffer management process consults this flag to determine whether or not blocks read by this stream should be discarded after they have been transmitted to the client. If the block is not discarded, the disk I/O process will find it in the buffer when it is needed by the later stream of the pair. This is because the disk I/O process uses the hash table to determine if a needed block is already in the buffer before initiating a disk I/O. Similarly, deallocating buffer from a pair consists of incrementing NomFreeSpace and appropriately setting the status flags for the following pair of the stream. The transition values are required since during the transition period, the stream is holding both disk and buffer resources, and this affects resource management and cleanup in the server. For example, while allocating buffer to a pair, the buffer management process sets the status of the later stream to 01 and allocates buffer space to the pair. However, the switched stream does not immediately start reading from the buffer but continues to read from the disk until it reaches the point at which data blocks from the previous stream started being available in the buffer. The disk I/O process invokes the buffer management process to change the status to 11 after it finds the first block of the switching process in the buffer.

When the client finishes viewing a movie, a close request is received. If the stream was reading from the buffer or was being switched to the buffer, the buffer management process releases the space being held by the pair. Note that since the pair that just terminated is smaller than any of the pairs reading from disk, it cannot be possible to allocate buffer to more than one pair. For similar reasons, it cannot allocate buffer to more than one pair during the open process even if the buffer is deallocated from an existing pair. Thus no combinatorial optimization is involved during open and close to determine the pair to which the buffer will be allocated since only a simple decision - to which pair to allocate buffer - has to be made.

3 Analytic Modeling

The analysis of the interval caching policy can be related to the general problem of obtaining order statistics [8]. However, as we will see, obtaining explicit expressions or evaluating them may be quite a difficult task. Let T^j be the time difference between the playback streams S_j and S_{j-1} on the same movie, i.e., the playback stream S_j is started after a time period of T^j since the start of playback stream S_{j-1}. Let w be the average data stream rate for playing a movie. Then, with a buffer size of $B^j = wT^j$, the playback stream S_{j-1} can be buffered to satisfy the playback stream S_j. The inter-arrival time, T_i and hence, buffer requirement, B^j, for the j^{th} stream are both random variables. If all the streams are for the same movie, then for n) concurrent streams in the system, n of the streams (i.e., all but the earliest one) can be served from the buffer given sufficient buffer space. For a limited buffer space, B, the interval caching policy orders the streams in terms of their buffer requirements and chooses the maximum number of streams, K, that can be satisfied with that buffer. Let B_1, B_2, ... B_{n-1} be the buffer requirements for the $(n-1)$ following streams ordered in terms of their buffer requirements. Therefore,

$$K = \max \left\{ k : \sum_{j=1}^{k} B_j \leq B \right\}. \tag{1}$$

To determine the expected number of streams that can be satisfied, $\overline{K}$, we have to use the following Equality:

$$\overline{K} = \sum_{n=1}^{\infty} P\left[N = n\right] \sum_{k=1}^{n-1} P\left[K \geq k | n\right]$$

$$= \sum_{n=1}^{\infty} P\left[N=n\right] \sum_{k=1}^{n-1} P\left[\sum_{j=1}^{k} B_j \leq B|n\right]. \quad (2)$$

$P\left[N=n\right]$ will depend on the interarrival time distribution. $P\left[\sum_{j=1}^{k} B_j \leq B|n\right]$ can be obtained by integrating the joint distribution of $(B_1, B_2, ...B_k|n)$ over the state space $(b_1, b_2, ...b_k|n)$ that satisfies the Inequality $\sum_{j=1}^{k} b_j \leq B$ and $k \leq n$.

The joint distribution of $(B_1, B_2, ...B_k|n)$ is difficult to obtain. Let $f(\cdot|n)$ and $F(\cdot|n)$ be the density and cumulative distributions for buffer requirement of a single stream, given that there are $(n+1)$ concurrent streams in the system. If the buffer requirement of a stream is independent of that of the other $n-1$ streams, then the joint distribution of $(B_1, B_2, ...B_k|n)$ is given by (following [8])

$$f_{1,2,...k}(b_1, b_2, ...b_k|n) = \frac{(n-1)!}{(n-k-1)!} \prod_{i=1}^{k} f(b_i|n)[1-F(b_k|n)]^{n-k-1}, \quad (3)$$

where $b_1 \leq b_2 \leq ... \leq b_k$ and $k \leq n$. Unfortunately, in the present problem, the distribution for buffer requirement of the j^{th} ordered stream is dependent not only on n but also on the distributions of those of the $j-1$ smaller buffer requirements, and hence, Equation 3 cannot be applied. The n playback streams divide the movie space into $n+1$ segments. The buffer requirement of each stream is the length of the segment between itself and the previous stream. Therefore, $\sum_{j=1}^{n-1} B_j = L - B_{begin} - B_{end}$, where L is the length of the movie, B_{begin} is the length of the segment between the beginning of the movie and the latest playback stream, and B_{end} is the length of the segment between the first playback stream and the end of the movie. Therefore, $B_j = L - B_{begin} - B_{end} - \sum_{l=1}^{j-1} B_l$, i.e., the distribution of B_j is dependent on the distribution of the $j-1$ smaller buffer requirements.

There are several other factors that add to this complexity for the case with multiple movies. First note that the inter-arrival time distribution and hence, the buffer requirement distribution for a random stream, are different for different movies. Since, the streams with the smallest buffer requirements (irrespective of the movie identity) are always chosen to be served from the buffer, the state space now becomes very complex. Let $N_1, N_2, ...N_m$ be the number of concurrent streams for the movies $1, 2, ...m$, respectively. As before let B_1, B_2, ... B_n the buffer requirements for the n streams over all movies ordered in terms of their buffer requirements, where $n = \sum_{i=1}^{m}(n_i - 1)$. The Equation 2 can now be rewritten as

$$\overline{K} = \sum_{(n_1,n_2,...n_m)} \prod_{i=1}^{m} P\left[N_i = n_i\right] P\left[\sum_{j=1}^{k} B_j \leq B|(n_1, n_2, ...n_m)\right] \tag{4}$$

Note that j^{th} smallest buffer requirement, B_j, may correspond to the request for any of the movies. Therefore, the state space, $\mathbf{x}$, is now represented by $(b_1, b_2, ...b_k, c_1, c_2, ...c_k)$ where $b_1, b_2, ...b_k$ are the k smallest buffer requirements and $(c_1, c_2, ...c_k)$ are the corresponding movie identities. Given a state space, $\mathbf{x}$, it can be mapped to $(k_i, b_{i,1}, b_{i,2}, ...b_{i,k_i},\ i = 1, ..., m)$ where k_i is the number of smallest buffer requirements in $\mathbf{x}$ for the movie i, and $(b_{i,1}, b_{i,2}, ...b_{i,k_i})$ be the k_i smallest buffer requirements for the requests of movie i. The last product term of Equation 4, $P\left[\sum_{j=1}^{k} B_j \leq B|(n_1, n_2, ...n_m)\right]$, can now be computed by integrating over the state space $\mathbf{x}$ that satisfies $\sum_{j=1}^{k} b_j \leq B$ and $k_i \leq n_i$, for $i = 1, 2, ...m$. The distribution for state space, $\mathbf{x}|(n_1, n_2, ..., n_m)$, can be written as

$$f(\mathbf{x}|n_1, n_2, ..., n_m) = \prod_{i=1}^{m} f^i_{1,2,...k_i}(b_{i,1}, b_{i,2}, ...b_{i,k_i}|n_i) \tag{5}$$

where $f^i_{1,2,...k_i}(B_{i,1}, B_{i,2}, ...B_{i,k_i}|n_i)$ is the joint distribution of the k_i smallest buffer requirements of the i^{th} movie given there are n_i concurrent streams for that movie.

The above expressions are extremely cumbersome or difficult to obtain. We therefore resort to simulation to study the effectiveness of the interval caching policy.

4 Simulation and Results

The effectiveness of the interval caching policy is studied by modelling a server with various amounts of buffer sizes delivering MPEG-1 videos, i.e., with a delivery rate of 150 KB/s. Movies are assumed to be approximately two hours long so that the total amount of storage required per movie is one GB. The arrival process is a Poisson process with inter-arrival time of T seconds. Simulations are run with $T = 14, 7$ seconds corresponding to a server simultaneously serving on an average 400 and 800 streams, respectively. These values are typical of the range of active clients expected for video-on-demand servers [11]. Since the effectiveness of the buffering policy depends on the distribution of access to the movies, two distributions are used in the simulation. The first distribution is based on the empirical data on video rentals in various video stores during a particular week [21]. Figure 5 shows the video rental frequency distribution. The empirical data can be fitted using a Zipf distribution over 92 movies with the parameter 0.271. In our experiments, we use the Zipf distribution to generate accesses to video files. Since buffering policies commonly work poorly when the access to objects is uniform, we also simulate a uniform distribution over all the movies. The buffer is modelled as a collection of 1MB blocks. While computing the amount of buffer space required for a consecutive pair, 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 buffer. Thus the simulations are run, in general, for 8 hours.

4.1 Comparison to static caching

In order to evaluate the effectiveness of the interval caching policy, we compare its performance to that of the policy of statically buffering the hottest movies. The primary metric used for this comparison is the number of buffered streams under both policies for a given buffer size, since the larger the number of buffered streams, the more effective is the usage of buffer. Figure 6 shows the results of the comparison for a buffer size of 1GB (sufficient to buffer one movie) and for an average inter-arrival time of 14 seconds. The values plotted are the instantaneous values of the number of buffered streams, after every 15 minutes interval. The two solid curves show the performance of the interval caching policy under the empirical and uniform access frequency distributions, respectively. The dashed curves are for the static buffering policy. As can be seen from the graph, under both the access distributions the interval caching policy outperforms the static policy. Under the uniform distribution, it can be seen that as expected, the static policy performs poorly since the access rate to any individual movie is small. On the other hand, even under the uniform distribution, the interval caching policy buffers a substantial number of streams. The performance difference between the two policies is almost 10 times in this case. The performance of the static policy improves under the empirical distribution. The interval caching policy however still outperforms the static policy buffering by twice as many streams. Note that the best-case performance of the static policy is poorer than the worst-case performance of the interval caching policy.

Figure 7 compares the two policies when the arrival rate is twice as large (inter-arrival time = 7 seconds). The number of buffered streams under the static policy nearly doubles for both the empirical and uniform distributions since the overall arrival rate as well as that for the most popular movie are doubled. The number of buffered streams under the interval caching policy also nearly doubles. This is because the inter-arrival time for each movie is halved, halving the amount of buffer required by consecutive requests for the same movie. Thus while the absolute performance of both algorithms increase, their relative performance does not change.

4.2 Robustness of policies with respect to changing load

We next compare the performance of both algorithms in the presence of a change in the frequency of access to the movies. Robustness with respect to such change is clearly a desirable property. In order to cope with such changes, any static buffering policy must include a means for detecting when a movie is no longer popular and then replacing it with the newly popular movie. Distinguishing a long term load change from a statistical fluctuation is a difficult task particularly in the presence of a slow load change and replacing the buffered movie involves substantial overhead since during the change in buffering the efficiency of the buffer may be degraded.

In the simulation, a load change is achieved by rotating the Zipfian distribution over movie indices so that while the shape of the distribution remained the same, the actual peak of the distribution is displaced. Due to the asymmetrical shape of the Zipf distribution, a simple rotation would have produced a new distribution where access change for the movies at the ends of the distribution would have been very large. In order to avoid this discontinuity at the ends, the Zipfian distribution is first converted into a symmetric distribution by permuting the movie indices so that movie 0 is mapped into index 46, movie 1 into index 45, movie 2 into index 47 etc. This symmetric distribution is then rotated and the inverse mapping is applied to determine the new load on each movie. Both a small load change and large change are simulated by a small and large rotation of the symmetric distribution. In each case, the load change is applied after 4 hours of simulation time.

Figure 8 compares the two algorithms before and after the load shift. Under a large load change, the static policy performs very poorly, showing a sharp drop after 4 hours (i.e., after the load change). No such change can be seen in the performance of the interval caching policy. Under the small load change, the performance of the static algorithm drops by a smaller amount showing that as expected, the degradation of the static algorithm depends on the extent of the load change. The performance of the interval caching algorithm is again unaffected, showing that the interval caching algorithm is robust with respect to a sudden load change.

The performances of the static and interval caching algorithms are next compared in the presence of a continuously fluctuating load. While the static algorithm can deal with a single load change by replacing some of the buffered movies, it cannot adapt to constant load fluctuations by replacing the buffered movies. This is due to the high overhead of replacement and the difficulty of detecting load change. Changing access patterns are simulated by periodically superimposing a random load variation on top of a uniform access pattern, such that the total load in the system remained constant. Figure 9 compares the two algorithms in the presence of rapid load changes (every half hour). The performance of the two algorithms are studied for both a small load change (variations of upto 25% from uniform) and a large change (upto 50%). Interestingly, though the interval caching algorithm outperforms the static algorithm by a large factor, the performance of the static algorithm is largely unaffected by the fluctuating load. This can be seen from the fact that there is no observable difference between the performance of the static algorithm in the presence of large and small load changes. This is because the time period between access rate changes is small compared to the length of a movie. Hence the number of clients viewing a movie is the average of the number of requests over many load changes. Since the average load is constant, the number of requests for any movie is nearly constant. As before, the performance of the interval caching policy is unaffected by the load changes.

In the presence of slow load changes, Figure 10 shows that the relative performance of the static algorithm under small and large change is correlated with the rate of change in the access rates (load change is applied every two hours). This indicates that the performance of the static algorithm depends on the access rate to the movies. On the other hand, there is no such correlation in the interval caching algorithm showing that the performance of the interval caching policy is independent of the load change.

4.3 Characteristics of interval caching policy

The previous subsection clearly indicates that the interval caching policy is superior to static buffering, both with respect to the number of streams that can be buffered with a given amount of buffer and the adaptability to load change. Hence in the following, we do not consider static buffering but focus on exploring some important characteristics of our interval caching policy. First, we consider the amount of required buffer space per stream. This is important since it determines the cost of serving streams from the buffer. The buffer space per stream is computed by dividing the total amount of buffer space by the average number of streams reading from the buffer. This is computed by sampling the number of streams reading from the buffer every 5 minutes in an 8 hour run and then taking the average. As mentioned earlier, trial runs showed that this yielded an estimate of sufficient accuracy. In recording the number of streams reading from the buffer, those streams which are in the process of switching to the buffer are not counted. Only those streams that are actually reading from the buffer (read state $= 11$) are counted.

Figure 11 plots the buffer requirement per stream as a function of the buffer size for both the uniform and empirical distributions and two different inter-arrival times. With a small amount of buffer, only those pairs that are close together are captured. With increasing amount of buffer, pairs that are further apart can be captured, causing the buffer per stream to increase. The amount of buffer per stream increases asymptotically to the value when there is enough storage to buffer all the streams. The buffer per stream halves as the average inter-arrival time is halved, since this implies that pairs of requests for the same movie will be spaced by half the distance. This implies that large servers will require less buffer per stream than smaller servers since the large server will serve a larger number of streams and hence, will have a smaller request inter-arrival time. It is also evident that for a given inter-arrival time, more buffer per stream is required for a uniform access distribution. This is due to the fact that under the skewed (empirical) distribution, the inter-arrival time of the requests for the popular movies is smaller than that for the other movies and the interval caching policy buffers the smaller inter-arrival times.

We next consider the achievable buffer hit ratios by the interval caching policy. The buffer hit ratio determines the achievable reduction in required disk bandwidth. Figure 12 shows the change in the average hit ratio with the increase in the amount of available buffer. The hit ratio is the number of streams reading from the buffer divided by the total number of streams. As before, the average is computed by sampling the instantaneous values every 5 minutes and then taking the mean. The hit ratio initially rises rapidly with a small amount of buffer but then begins to rise more slowly, since the amount of buffer per stream increases as the total amount of buffer available increases. In the region of observation, the hit ratio is almost invariant with the inter-arrival time, since halving the inter-arrival time causes both the total number of streams and the number of buffered streams to double. Also, the hit ratio under the empirical distribution is larger than that under the uniform distribution as more buffer is required under the uniform distribution. Note also that with a buffer large enough to hold 8 movies, a hit ratio of nearly 50% can be achieved under the empirical distribution and 35% under the uniform distribution. Under the static policy of buffering the hottest movie, a buffer large enough to hold 16 and 32 movies, respectively would be required.

The reduction in the required disk bandwidth under the interval caching policy is shown in Figure 13. Note that the streams that can not be served from the buffer have to be served from the disks. The figure plots both the total number of streams and the number of disk streams required as a function of time. A buffer large enough to hold one movie is assumed, and the number of disk streams required under both the empirical and uniform distributions is shown. Since the total number of streams does not change with the access distribution (uniform or empirical), it is plotted only once. The variations in the amount of disk bandwidth required can be seen to be less than the variation in total bandwidth. This can also be inferred from the observation that the hit ratio is nearly invariant with the arrival rate; if the total stream requirement fluctuates by an amount x due to fluctuations in the arrival rate and the hit ratio is h, the resulting fluctuation in the disk bandwidth required will be hx.

5 Cost-effectiveness

Since large amounts of buffer (on the order of gigabytes) are required by the interval caching policy, it is natural to study if the interval caching policy is cost-effective with respect to disk. We do this in the following by comparing the incremental cost of delivering movies out of buffer with the incremental cost of using disk. The number of disks required in any video server can be determined either by the capacity required to store the movies or the bandwidth required to serve the desired number of customers. If the disk requirements are driven by capacity, buffering is not required since once there is sufficient capacity to store the movies, sufficient bandwidth is also available. Hence we consider only configurations where the disk requirements depend upon the bandwidth required to serve the desired number of customers. We believe that such configurations will be more common.

Let S_d be the number of streams supplied from disk, S_b the number of streams supplied from the buffer and B the amount of buffer required in MB. If all the streams are to be supplied from disk, enough disks to supply S_b streams are required instead of the buffer of size B. If the cost of B MB of buffer is less than the cost of disks required to deliver S_b streams, then the buffer is cost-effective. The cost per stream of delivering movies out of buffer is the storage required per stream (B/S_b) multiplied by the unit cost of storage. The cost per stream of delivering movies from disk is the cost of the disk divided by the number of streams that the disk can support. Note that the following analysis assumes MPEG-1 streams with a data rate of 150 KB/s as assumed in the simulation. However, the analysis is independent of the data rate since the costs of serving movies out of buffer and disk both scale linearly with the data rate.

The cost of serving movies both from disk and from the buffer under the empirical distribution is shown in Figure 14. The cost per stream from buffer is computed for two average inter-arrival times of 7 and 14 seconds corresponding to a server serving 800 and 400 concurrent users, respectively. It is assumed that the cost of storage is $50/MB. It can be seen that the cost of using buffer depends on the total amount of available buffer and the inter-arrival time of requests. The buffer cost for delivering movies increases with the total amount of available buffer since (as in the discussion of Figure 11) the amount of required buffer per stream increases with the total amount of available buffer. Since less buffer is required with a smaller inter-arrival time, the buffer cost per stream is smaller for the server with 800 clients. The cost of serving movies from disk is estimated by using parameters typical of current disks. For analyzing the disk costs, a 2 GB disk with a cost of $5000 (or $2.5/MB) with a bandwidth of 3 MB/s is assumed. Such a disk would deliver 20 MPEG-1 streams at 150 KB/s assuming 100% utilization and the cost per stream would be $250. Since this is independent of the amount of buffer available, it is plotted as the solid horizontal line in Figure 14. If the disks can be run only at 60% utilization, the cost of serving a stream from disk would be $250/0.6 =416.66.

It can be seen that the disk cost and buffer cost curves intersect, implying that for some ranges, it is cost-effective to use buffer for serving movies. For an 800-client server, between 0.5GB and 1.25 GB of buffer is justifiable depending upon the maximum allowable disk utilization while for a 400-client server, between 250MB and 500MB of buffer is cost-effective. Smaller amounts of buffer are justifiable for smaller servers since the buffer cost per stream of smaller servers is higher due to the larger amount of buffer/stream required. A similar analysis under the uniform distribution is shown in Figure 15. Lower amounts of buffer are cost-effective since as discussed earlier, the buffer per stream requirements of the interval caching policy is higher under the uniform distribution, resulting in a higher buffer cost.

6 Conclusions

In the paper, a new buffer management policy for video-on-demand servers called the *interval caching* policy is proposed. By considering the implementation of the policy in detail, it is shown to be easy to implement in multimedia servers. Since analytical modelling of the policy is intractable, the performance of the interval caching policy is studied by simulation. Its performance is shown to be superior to that of the static policy of buffering the most popular movies, using the number of streams buffered as the performance metric. For the same amount of buffer, the interval caching policy buffers more streams than the static policy. The relative difference depends on the actual distribution of the accesses over the movies. For a uniform distribution, the interval caching policy buffered 10 times as many streams as the static policy while under an empirically observed (skewed) distribution, the interval caching policy buffered twice as many streams. In addition, the interval caching policy automatically adapts to the changes in the access distribution always providing superior performance whereas the performance of the static policy may be affected severely by the changes in the access rates. This is a serious drawback of any static policy since detecting changes in the access distribution is not an easy task and changing the buffered movies requires a lot of overhead. Additionally, the performance of the static algorithm is degraded during this transition. The interval caching policy is also very effective under a steady-state load, since substantial hit ratios (up to 40%) are achievable with a relatively small amount of buffer (enough to store 8 out of the 92 movies).

The interval caching policy is also shown to be cost-effective for multimedia servers by comparing the cost of serving movies out of buffer with that of using disk. The cost of using buffer is estimated as the cost of average amount of required buffer per stream. The cost of using buffer depends on several factors: the access distribution, the size of the server (i.e., the number of streams being served) and the total amount of available buffer. The buffer cost is higher for the uniform distribution than for the empirical distribution. Interestingly, the buffer cost of larger servers is lower than that for smaller servers since less buffer per stream is required for larger servers. The buffer cost also increases with the amount of available buffer since the average buffer requirement per stream under the interval caching policy increases with the total amount of available buffer. On the other hand, the cost of using disk is independent of the amount of available buffer. There is a region where it is economical to use buffer. Since larger servers have lower buffer per stream costs, more storage is justifiable on larger servers. With the empirical distribution of access to movies and an 800-client server, between 500MB and 1.25GB of buffer storage is shown to be justifiable based on typical current prices.

References

[1] Anderson, D. P., Y. Osawa, and R. Govindan, "A File System for Continuous Media", *ACM Transactions on Computer Systems*, Vol. 10, No. 4, Nov. 1992, pp. 311–337.

[2] Anderson, D. P., "Metascheduling for Continuous Media," *ACM Transactions on Computer Systems*, Vol. 11, No. 3, August 1993, pp. 226–252.

[3] 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.

[4] Chou, H. T., and D. J. Dewitt, "An Evaluation of Buffer Management Strategies for Relational Database Systems," *11th International Conference on Very Large Databases*, Stockholm, Sweden, 1985.

[5] 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.

[6] Dan, A., P. Yu, and J. Y. Chung, "Database Access Characterization for Buffer Hit Prediction", *Ninth Intl. Conf. Data Engg.*, Vienna, Austria, April 1993, pp. 134–143.

[7] Dan, A., D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching", *IBM Research Report*, Yorktown Heights, NY, 1993.

[8] David, H. A., *Order Statistics, Second Edition*, John Wiley and Sons - New York, London, Sydneey, Toronto, 1980, pp 10.

[9] Effelsberg, W. and T. Haerder, "Principles of Database Buffer Management", *ACM Trans. Database Systems,* Vol. 9, No. 4, Dec. 1984, pp.560-595.

[10] Faloutsos, C., R. Ng, and T. Sellis, "Predictive Load Control for Flexible Buffer Allocation," *17th International Conference on Very Large Databases*, Barcelona, Spain, 1991, pp. 265–274.

[11] *Electronic Engineering Times*, March 15, 1993, pp 72.

[12] Fox, E. A., "The Coming Revolution in Interactive Digital Video," *Communication of the ACM*, Vol. 7, July 1989, pp. 794–801.

[13] 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.

[14] 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.

[15] Rahm, E., and D. Ferguson, "Cache Management Algorithms for Sequential Data Access" *IBM Research Report, RC15486*, Yorktown Heights, NY, 1993.

[16] 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.

[17] Sincoskie, W. D., "System Architecture for a Large Scale Video On Demand Service," *Computer Networks and ISDN System*, Vol. 22, 1991, pp. 155–162.

[18] Smith, A. J., "Sequentiality and Prefetching in Database Systems", *ACM Trans. Database Systems,* Vol. 3, No. 3, Sept. 1978, pp. 223-247.

[19] 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.

[20] Tobagi, F. A., J. Pang, R. Baird, and M. Gang, "Streaming RAID – A Disk Array Management System for Video Files", *Proc. ACM Multimedia 93*, Anaheim, CA, Aug. 1993, pp. 393–400.

[21] *Video Store Magazine*, Dec. 13, 1992.

Figure 1: Multimedia video server environment

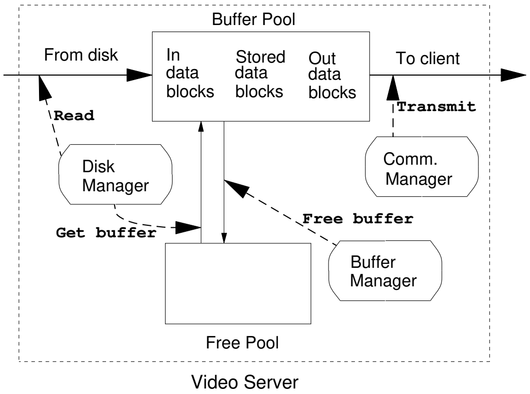


Figure 2: Video server components

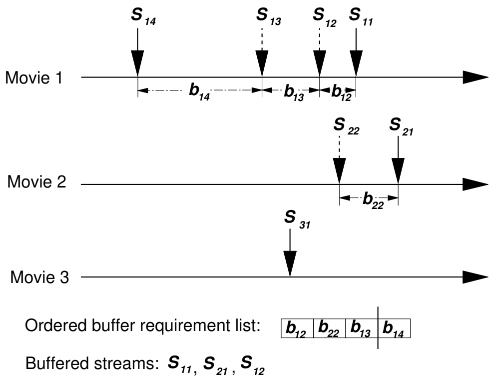
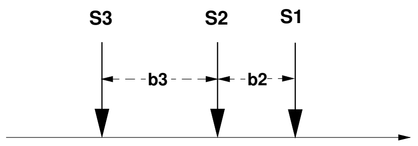


Figure 3: Proposed buffer management policy



Stream id	Current block id	Read state	Follow up stream id	Estimated buffer req.
S1	p1	11	S2	b2
S2	p2	01	S3	b3
S3	p3	null	null	null

Stream Information Table

Figure 4: Stream management data structures

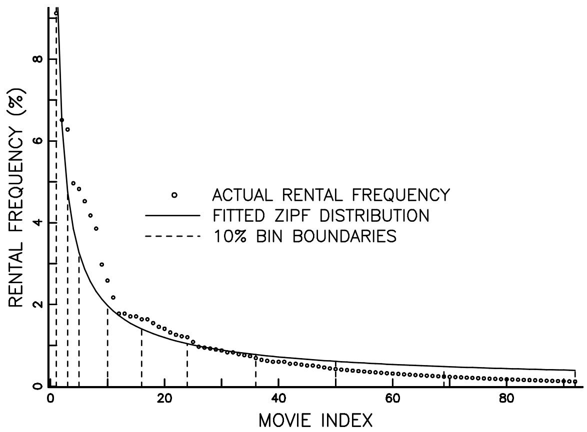


Figure 5: Movie request distribution

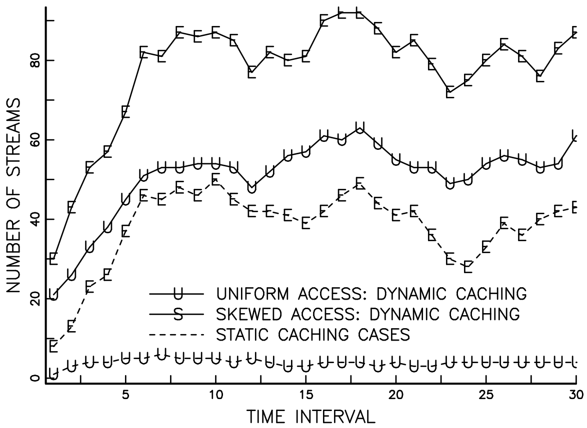


Figure 6: Comparison of static and interval caching policies, inter-arrival time=14 sec

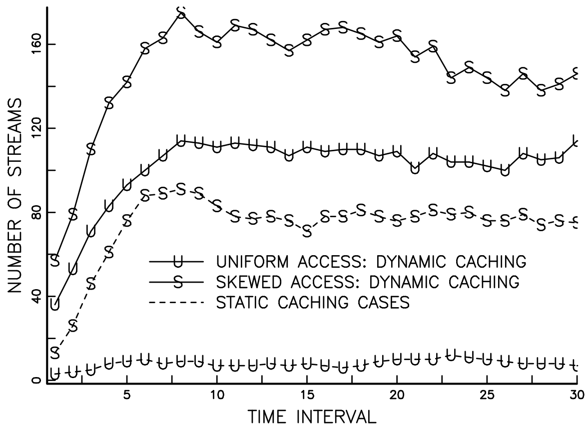


Figure 7: Comparison of static and interval caching policies, inter-arrival time=7 sec

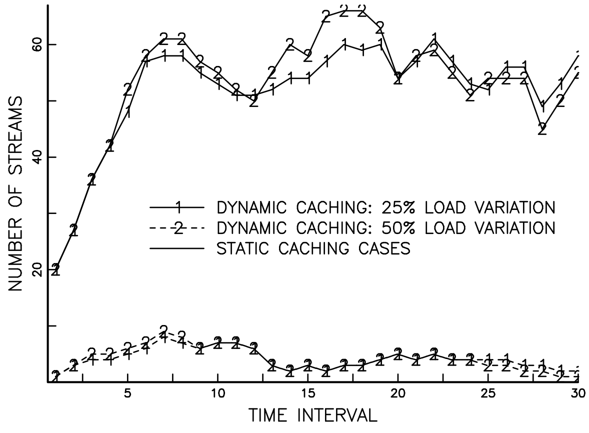


Figure 10: Comparison of static and interval caching policies, slow load change (every 2 hours)

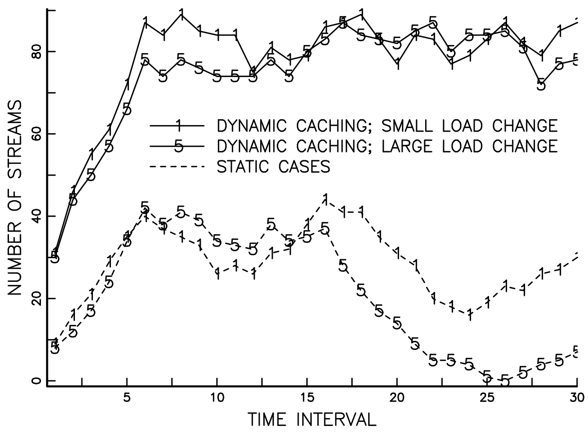


Figure 8: Comparison of static and interval caching policies in presence of load change

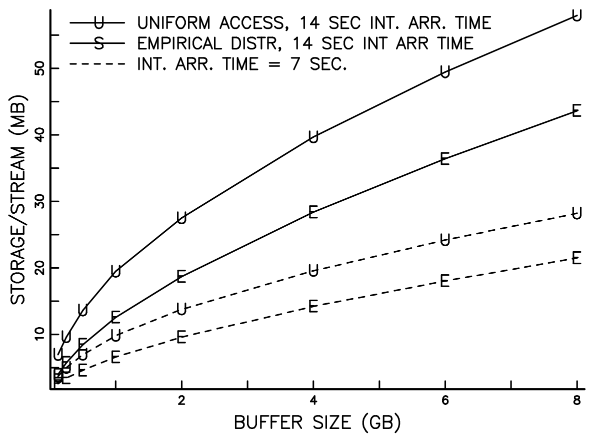


Figure 11: Storage/stream of interval caching policy

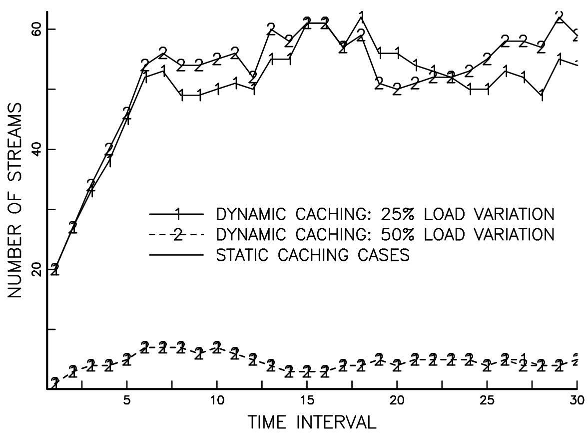


Figure 9: Comparison of static and interval caching policies, rapid load change (every 1/2 hour)

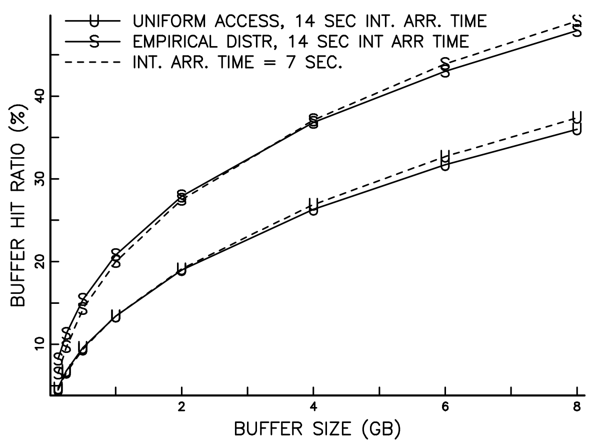


Figure 12: Buffer hit ratio of interval caching policy

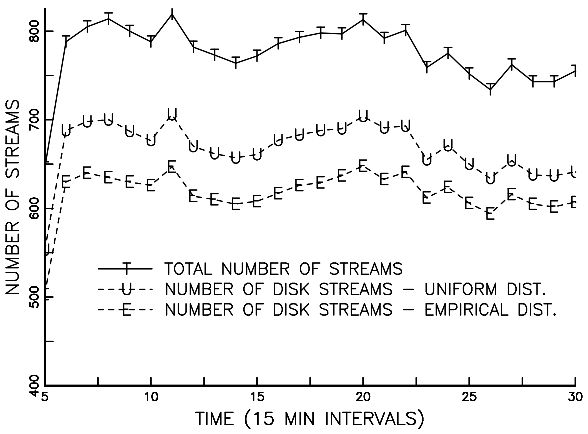


Figure 13: Effect of load fluctuation on required disk capacity

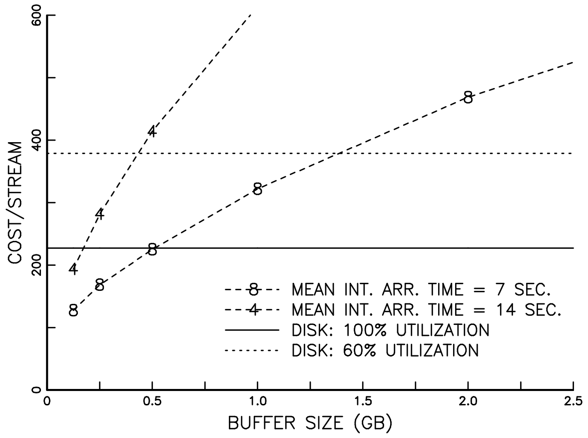


Figure 14: Comparative cost/stream of disk vs storage, realistic movie request distribution

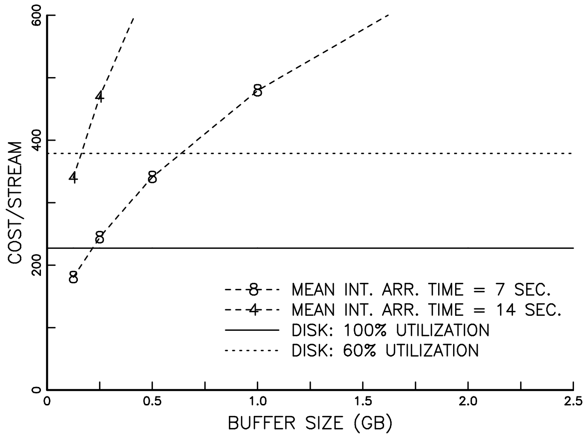


Figure 15: Comparative cost/stream of disk vs storage, uniform movie request distribution