Buffer Management for Continuous Media Sharing in Multimedia Database Systems

Mohan Kamath†, *Krithi Ramamritham*† and *Don Towsley*‡
Computer Science Technical Report 94-11
Department of Computer Science
University of Massachusetts
Amherst MA 01003

Abstract

Buffer management in database systems deals with allocation, search and replacement of buffers for the data to be accessed so as to promote data sharing and reduce the number of disk accesses. These activities are much more complex in multimedia database systems (MMDBS) which handle time variant data. Multimedia data have timeliness and synchronization requirements and hence traditional buffer management schemes like FIFO and LRU are not quite suitable for multimedia buffer management. Since the data rates involved are very high, despite the development of efficient storage and retrieval strategies, disk I/O is likely to be a bottleneck, thereby limiting the number of concurrent sessions supported by a system. This calls for better use of data that has already been brought into the buffer by exploiting sharing whenever possible using advance knowledge of the multimedia stream to be accessed. Since most of the research work in multimedia data management has concentrated on data modeling and storage management, no work has been done to explore the benefits of data sharing when concurrent users are accessing multimedia data.

In this paper, by considering the specific application of News-On-Demand-Service (NODS) where extensive sharing of continuous media data is possible, we explore the potential benefits of continuous media sharing. We first introduce the notion of *continuous media caching* which is a simple and novel technique where buffers that have been played back by a user are preserved in a controlled fashion for use by subsequent users requesting the same data. We then propose new buffer management schemes that use this technique to utilize the available buffer space effectively to promote sharing, thereby reducing the number of disk I/Os. Finally we compare our schemes with traditional buffer management schemes through simulation. Our results indicate that buffer management schemes that utilize continuous media caching outperform the other schemes in environments where sharing is possible. This indicates that for improved performance, it is essential to develop new MMDBS-specific buffer management schemes that exploit the characteristics of multimedia applications.

† Supported by the National Science Foundation under grant IRI - 9109210.
‡ Supported by ARPA under contract F19628-92-C-0089.

Contents

1 Introduction

Multimedia systems have attracted the attention of many researchers and system developers, because of the multimedia nature of the data used in modern applications and due to their capability to improve the communication bandwidth between computers and users. These multimedia information systems cater to a variety of scientific and commercial fields ranging from medicine to insurance. Lately the construction of the national information super-highway has received enormous attention and many commercial ventures are focusing on providing *on-demand* services. Multimedia information systems will form the backbone of these services. Multimedia data can be classified as *static* or *dynamic* (also known as *continuous*) media. While static media, like text and images, do not vary with time (from the perspective of the output device), dynamic media, like audio and video, vary with time. Whenever continuous media is involved, in order to ensure smooth and meaningful flow of information to the user, *timeliness* and *synchronization* requirements are imposed. Timeliness requires that consecutive pieces of data from a stream be displayed to the user within a certain duration of time for continuity. Synchronization requires that data from different media be coordinated so that the user is presented with a consistent view. Hence, unlike real-time database systems where deadlines are associated with individual transactions, multimedia data has continuous deadlines. Though efficient storage and retrieval strategies are being developed, since the data rates involved are very high (between 2-5 MB/sec of compressed data per request) disk I/O could still be a bottleneck, *i.e.*, the number of concurrent sessions supported could be limited by the bandwidth of the storage system. Hence, it is important to make better use of data brought from the disk into the database buffer.

In this paper, we explore the benefits of continuous media sharing in multimedia database systems (MMDBS). Specifically, we consider the News-On-Demand-Service (NODS) which typifies applications where extensive data sharing is possible and focus on buffer management schemes that promote sharing of continuous media data. Since multimedia data are typically read only, by sharing the data between users, many trips to the disks can be avoided. Unlike traditional multimedia systems where data is typically retrieved and sent directly to the output device, in applications where sharing can be exploited, we will show that it is preferable for the data to be brought into the buffer from the disk at the server and retained in a controlled fashion to promote sharing. The motivation for our work comes from the fact that sharing of continuous media data can increase the number of users that can be supported by a system. Furthermore there are many interesting issues that arise in such an environment.

The most important fact is that once the user has requested a playback, advance knowledge of the multimedia stream to be accessed can be used to anticipate demand and promote sharing (planned sharing for future accesses). This is particularly true with NODS since users watching news do not request fast-forward or rewind. To achieve this, we introduce the notion of *continuous media caching*. It is a simple and novel technique where buffers that have been played back by a user are preserved in a controlled fashion for use by subsequent (or lagging) users requesting the same data. This technique can be used when there is sufficient buffer space such that it is possible to retain data for the required duration. With this technique, the system can avoid fetching the data from the disk again for the lagging user. Hence this technique makes it possible to support more numbers of sessions than the number permitted by the storage system. Also the throughput of the system, *i.e.*, the percentage of user requests permitted, can be increased by using the cheaper option of additional memory instead of choosing a storage system with a higher bandwidth. In the two buffer management schemes we propose, SHR1 and SHR2, data sharing in exploited using the technique of continuous media caching (planned sharing). Since multimedia applications have certain timeliness and synchronization requirements, and the amount of buffer and disk bandwidth available is limited these schemes also have admission control integrated with them. SHR2 also tries to utilize the data that might already be in the buffer (unplanned sharing) and hence it is one of the best possible schemes for buffer management of continuous media data.

Thus far, most of the research work in the area of multimedia data management has concentrated on data modeling and storage mangement. While some of the issues relevant to MMDBS are enumerated in [4, 5, 12, 16], object-oriented approaches for implementing MMDBS are suggested in [6, 12, 17, 26]. Some architectural aspects of multimedia information systems like data placement, buffering and scheduling are described in [6] while issues related to a distributed multimedia information system, performance in particular are discussed in [4]. Extensive research and performance studies on efficient storage and retrieval strategies for multimedia data can be found in [2, 9, 13, 20, 23, 25, 27]. Is-News [21] and ORION's Multimedia Information Manager (MIM) [26] are examples of some of the many multimedia information systems that have been developed. Some aspects of buffer management for multimedia data are addressed in [2, 6]. While [6] does not specifically consider concurrent sessions in multimedia systems, [2] does address some performance issues when concurrent users are trying to access continuous media data. However it disregards sharing of continuous media data. The Use-And-Toss-it (UAT) policy has been suggested in [6] for multimedia data, under which, once a piece of data is used, it is tossed away so that the next piece of required data is brought into the same buffer. Generally the UAT policy is good only if data sharing does not exist or cannot be exploited. Buffer management schemes have been widely studied in traditional database systems [8, 22] and in real-time database systems [15]. These typically consider only textual data. The objective of these buffer management schemes is to promote sharing and reduce disk accesses, thereby increasing the throughput. In real-time database systems an additional objective is to maximize the number of transactions that execute within their deadlines. Commonly used buffer management schemes like first-in first-out (FIFO) and least-recently-used (LRU) are not suitable for multimedia data since they have continuous deadlines. Essentially, there has not been any research to evaluate the potential benefits of continuous media sharing when users are accessing multimedia data concurrently.

The rest of the paper is organized as follows. In section 2 we discuss the data characteristics of multimedia data and NODS in particular. Issues related to continuous media sharing are discussed in section 3. Using simple examples, we illustrate how continuous media sharing can be useful in increasing the number of users supported by a system. Section 4 first reviews some of the traditional schemes which we will be considering in our performance studies for comparison purposes. Then it discusses the proposed buffer management schemes that exploit continuous media sharing. Details of the performance tests are discussed in section 5. It describes the various assumptions made, parameters used for the tests and the results. These results show the superior performance of buffering schemes that exploit data and application characteristics. Possible extensions to this work are described in section 6. Section 7 concludes with a summary of this work.

2 Data Characteristics

Multimedia objects are captured using appropriate devices, digitized and edited, until they are in a form that can be presented to the user. These independently created objects are then linked using spatial and temporal information, to form a composite multimedia object that may correspond to a presentation or some specific application.

In order to ensure smooth and meaningful flow of information to the user, there are *timeliness* and *synchronization* requirements which vary from application to application. Timeliness requires that consecutive pieces of data from a stream be displayed to the user within a certain duration of time for continuity and hence multimedia data has continuous deadlines. Synchronization requires that data from different media be coordinated so that the user is presented with a consistent view. Essentially it refers to the temporal sequencing of events among multimedia data. It deals with the interaction of independent objects that coexist on different media such that the user is presented with a consistent view and is very much dependent on the type of application. Detailed analysis of synchronization requirements for multimedia data can be found in [19, 24]. This synchronization information forms the basis for buffer management.

The rest of this section describes the data characteristics of NODS, which typifies applications where extensive sharing of data is possible. In NODS, given that at any particular time the number of topics in high demand is likely to be limited, given the higher level of interest in certain topics like sports/business, the probability of one or more users requesting the same item for playback is very high. To request a service the user has to choose a topic and a language from, for example the following:

1. *Topic* - Politics, International, Business, Sports, Entertainment, Health, Technology etc.
2. *Language* - English, Spanish, Italian, French, German, Hindi etc.

Each topic consists of a list of news items. News items are composed of video and audio segments and possibly some images. All news items to be displayed when a topic is chosen are compiled in advance and known apriori.

Sharing of data can occur in various capacities as we discuss below. Typically there are many news items that are shared between topics. For example, consider the news item on NAFTA[1], which can be included in a variety of topics like Politics, International and Business since it has relevance to all. Sharing of continuous media can also occur at the media level. Since the same news items will be provided in different languages, to reduce storage, additional storage is necessary only for the

[1]North American Free Trade Agreement

audio segments as the video segments can be shared. In addition, sharing can occur when multiple users are requesting service on the same topic. This clearly shows that sharing at different levels can lead to compound sharing. In this paper, we mainly focus on the last type of sharing since it is likely to have the most impact on performance. Intelligent buffer management schemes can and must utilize this to improve the performance of a system. Typically users watching news do not request fast-forward and rewind. We discuss this further in section 6.

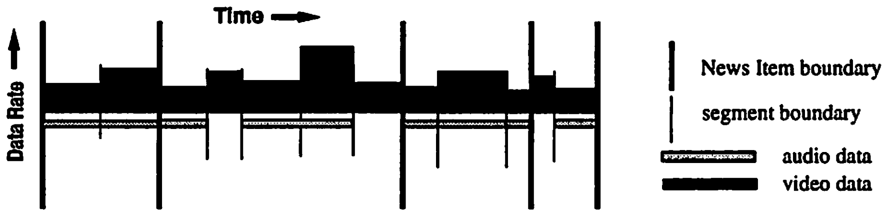

Figure 1: Multimedia Data Stream of a request

Figure 1 shows the data stream of a request. Since video is often compressed, we assume a variable bit rate for continuous media data. The data rate depends on the amount of action that is packed in the video. If the video is from a still camera (*e.g.* the newsreader reading the news) the data rate will be low, whereas if the video is from a moving camera (*e.g.* sports clippings) the data rate will be high. Hence each *segment* refers to a part of data that has the same data rate. A request may consist of many segments, each having its own data rate. In our work, segments are restricted to be of equal length for ease of implementation. However since the data rate of segments can vary, the amount of buffer required by a segment varies. In practice, if a segment is too long, it can be split into segments that have the same data rate.

3 Continuous Media Sharing

In this section we introduce the concept of *continuous media caching* which promote planned sharing. It is a simple and novel technique where buffers that have been played back by a user are preserved in a controlled fashion for use by subsequent users requesting the same data.

Since the buffer requirements for images and audio are less compared to video (as we will see later in this section), we ignore them and consider only the requirements of video data. Hence we mainly focus on sharing that occurs when multiple users request service on the same topic (though they have chosen a different language). When there is sharing, many possibilities exist and we study some alternatives here to see how the buffer and disk requirements analysis will change. When the same service is being requested by two (or more) users simultaneously, one of the following could be true:

1. they arrive at the same time (they start at the same time)
2. they arrive with a gap of few time units (or segments)
3. they arrive with a gap of large number of time units (or segments)

Depending upon the order in which users share an application, they may be classified as leading (the first one) or lagging (the subsequent ones that can share the buffers). Using the same buffers for both users can definitely payoff in case 1, whereas in case 2 it may payoff. In case 2, after the data has been played out of the buffer for the leading user, rather then discarding the data, it can be cached by preserving it in the buffer so that the same data can be reused by the lagging user (rather than fetching it again from the disk). We refer to this technique as continuous media caching. In case 3 it is easy to see that continuous media caching will not payoff since preserving buffers may prove to be too costly in terms of buffer availability. In this case it is better to totally ignore preserving data buffers. In our buffer management scheme, we allow sharing to occur as long as the difference between the arrival time of two consecutive users for the same topic is less than the length of the topic. Barring information about the future arrivals, this is the earliest time instant when a sharing situation can be recognized.

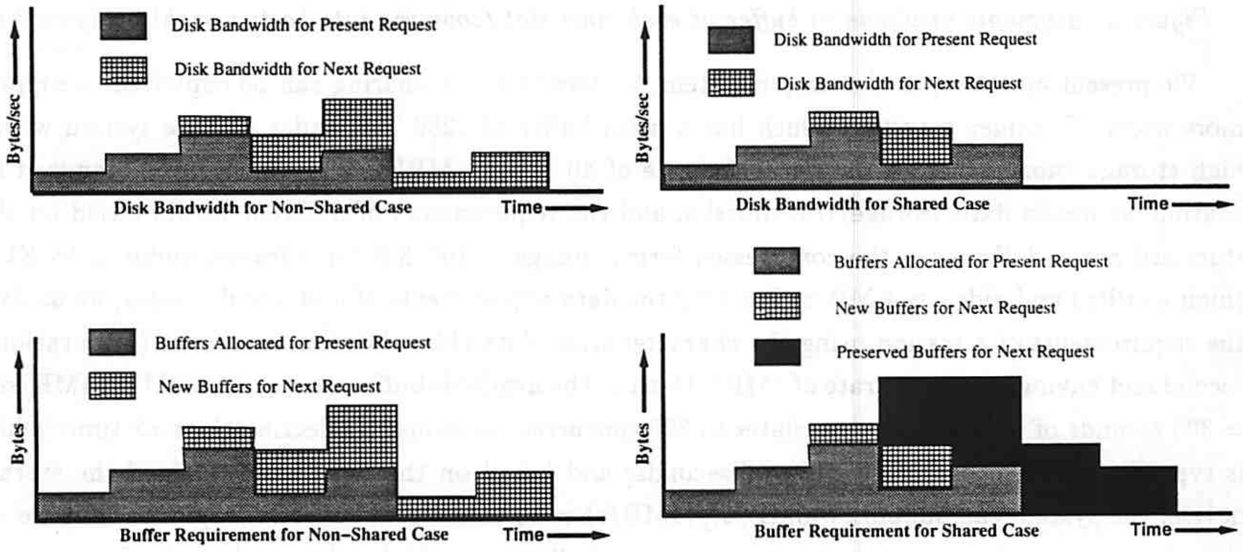

Figure 2: Buffer Sharing between two requests

Figure 2 compares the disk and buffer requirements with respect to time for the non-shared case and the shared case (continuous media caching). Two requests for the same topic (containing 5 segments) that arrive with a time difference of 2 segments is shown in the figure. In the non-shared case, disk and buffer bandwidths are allocated independently for each request. However for the shared case, since it is known that segments three through five will be available in the buffer (it will be loaded into the buffer for the first request) when the second request needs it, the buffers are reused for the second request, *i.e.*, the buffers are preserved till the corresponding playback time for the second user. These segments need not be fetched again from the disk for the second request. This results in an increase in the buffer requirement, but the disk bandwidth requirement drops since fewer trips are made to the disk. Thus it can be seen that continuous media caching exploits sharing to decrease the number of disk I/Os and hence improve performance.

Figure 3 shows the details of the segments that are actually present in the buffer at each time instant in the shared case (bottom right case of *figure 2*). It illustrates the amount of intelligence in the continuous media caching technique which preserves only the necessary segments for just the required amount of time. It differentiates the segments as follows — segments that are loaded for the first user, segments fetched from the disk for the second user and segments that have been used by the first user but preserved for the second user.

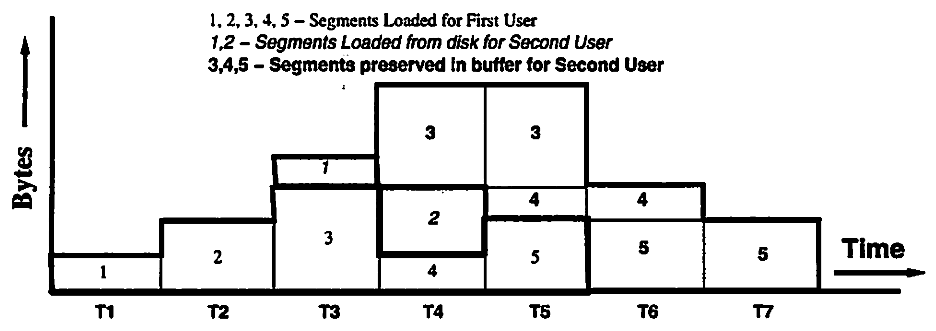
1, 2, 3, 4, 5 - Segments Loaded for First User
1,2 - Segments Loaded from disk for Second User
3,4,5 - Segments preserved in buffer for Second User
Bytes
Time
1
2
3
1
3
2
4
3
4
5
4
5
5
T1
T2
T3
T4
T5
T6
T7

Figure 3: Segments available in buffer at each time slot (zooming into bottom right of figure 2)

We present an example of a simple system, to illustrate how sharing can be exploited to support more users. Consider a system which has a main buffer of 1280 MB and a storage system with a high storage capacity and a data transfer rate of 40 MB/s. MPEG-2 is an accepted standard for continuous media data storage/transmission and the requirements of different media based on this standard are as follows (in the compressed form): image $\approx$ 100 KB for a frame, audio $\approx$ 16 KB/s (high quality) and video $\approx$ 4 MB/s. Ignoring the data requirements of audio and images, we analyze the requirements of a session using the characteristics of its video data on a segment (of duration 1 second and having same data rate of 4MB/s) basis. The available buffer can hold 1280MB/(4MB/sec) = 320 seconds of video, which translates to 320 concurrent sessions. Neglecting the seek time (which is typically in the order of 10's of milliseconds) and based on the data bandwidth of the storage device, the system can support (40MB/s)/(4MB/s) = 10 concurrent sessions (topics). Thus we see that the number of concurrent sessions supported is limited to 10 by the bandwidth of the storage system enenthough the buffer can handle 320 sessions. However by exploiting sharing, the buffer that can support 320 seconds of video can be effectively utilized to satisfy more number of users, provided they request one of the 10 topics that can be fetched using the available disk bandwidth. If the 10 topics are shared by $u_1, u_2, ..., u_{10}$ concurrent users respectively, then the number of users supported will be $\sum_{i=1}^{10} u_i$. This is however the best case and the probability of this is very low, since all the users sharing the sessions arrive at the same time in this scenario. On the other hand, the worst case occurs when the users requesting the same topic come with larger inter-arrival times. Continuous media caching may not be possible in this case (due to insufficient buffer bandwidth), resulting in only 10 users being supported by the system. However if they arrive within short intervals, instead of fetching the data from the disk for the lagging user, data read from the disk by the leading user can be preserved for reuse by the lagging user. Thus depending on the inter-arrival time and other possibilities of sharing (three cases discussed earlier), the number of users supported will be between 10 and $\sum_{i=1}^{10} u_i$. This simple example gives an idea of how sharing can be exploited in a MMDBS for the case of NODS to provide better buffer management. If seek times were also considered in the above example, continuous media caching will prove to be even more beneficial.

4 Buffer Management Alternatives

In this section we describe the five different schemes that have been considered for comparison. We first review traditional schemes and then present our proposed scheme.

4.1 Traditional Schemes

The objective of the buffer manager is to reduce disk accesses for a given buffer size. Usually fixed size pages are used to avoid fragmentation. When the requested database page is not in the buffer, it is fetched from the disk into buffer. Then the page is kept in the buffer using the *fix* operation, so that it is not paged out and future references to the page do not cause page faults in a virtual memory operating system environment. Once it is determined that the page is no longer necessary or is a candidate for replacement an *unfix* is done on that page. Buffer allocation and replacement are considered orthogonal issues in traditional database systems. A page can be allocated anywhere in the global buffer in a global scheme whereas in a local scheme, specific portions of the buffer are allocated for a given session and a page for a session is allocated in the area allocated for that particular session. Hence in the global scheme there is only one page replacement algorithm whereas in the local scheme several page replacement algorithms may exist. In either case the goal of the page replacement algorithm is to minimize the buffer fault rate. Before allocating a page, the buffer manager checks if the corresponding page is in the buffer and if not, it retrieves it from the disk. While static allocation schemes allocate a fixed amount of buffer for each session when it starts, dynamic allocation schemes allow the buffer requirements to grow and shrink during the life of the session.

In virtual memory operating systems where there is paging, code pages (which are read only) are typically shared, for example if many users are using a particular program like an editor, then the code pages of the editor are shared between the different users. Though this scheme cannot be used as such for continuous media data, it provides some motivation for the continuous media caching scheme. Some of the common page replacement strategies include FIFO and LRU [8]. We consider FIFO and LRU schemes which have been modified for our implementation. The modifications are necessary since we use a segmented buffer management scheme *i.e.*, buffers are allocated and replaced on a segment basis. During segment replacement, depending upon the buffer space required for the new segment that is to be brought in from the disk, one of more segments may have to be replaced. Segments in turn can be expressed in terms of standard pages (to avoid fragmentation), but we do not discuss this any further. In the FIFO scheme, if the next segment of a requested topic is in the buffer, it is reused and not fetched from the disk. If the next segment is not in the buffer, one or more of the oldest segments in the buffer are chosen for removal and replaced with the required segment. In contrast, the LRU scheme replaces the segments that were used least recently. Hence, in this scheme when the probability of choosing a particular topic is high, it is very likely that some of the segments of the topic may be in the buffer when needed. An important aspect to note is that since FIFO and LRU schemes do not utilize any information about the future use, they do not perform admission control and hence could fail any time during the processing of a request. This would result in the wastage of the disk and buffer bandwidth that have been used thus far.

Before we discuss the UAT scheme, we introduce the memory model, admission control and bandwidth reservation scheme that is used by UAT and the other schemes we are proposing. In the case of a MMDBS, advance knowledge of the data stream to be accessed is available and this has to be utilized for better buffer management. Since we are assuming a large main buffer, we assume that this information about the topic and segments can be stored in main buffer (*i.e.*, the number of segments that make up a topic and the segment data rates for each segment). Buffer space that is not used exists in a *free-pool.* Buffers needed for the segments of a request are acquired (analogous to fix) from the free-pool and returned (analogous to unfix) to the free-pool when they have been used. When buffers are requested from the free-pool, the oldest buffer in the free-pool is granted to the requester. The admission control and reservation scheme is dependent upon the sharing policy. Given $(n-1)$ sessions already in progress, the n^{th} request is accepted iff the following inequalities hold.

$$\sum_{i=1}^{n} l \cdot d_{i,t} \leq M \tag{1}$$

$$\sum_{i=1}^{n} d_{i,t} \leq R \tag{2}$$

where $d_{i,t}$ represents the continuous media rate for request i at time slot t, l represents the length of each time slot, typically 1 second. The first equation gives the limiting case for the buffer bandwidth (buffer size $= M$) and the second for disk bandwidth (data rate supported $= R$). If the inequalities hold, the required buffer and disk bandwidth is reserved.

The UAT policy has been modified for our implementation in the following manner: data is replaced on a segment basis and it uses admission control and bandwidth reservation to anticipate future demand. If there is no sharing, the UAT policy is ideal for continuous media data. UAT reuses segments that may already exist in buffer, *i.e.*, if any of the required segments happen to exist in the free-pool then they are grabbed from the free-pool, thereby avoiding fetching those segments again from the disk. If the free-pool is empty, then no buffer space is available and the request has to be rejected for insufficient buffer bandwidth. To fetch the remaining segments from the disk, UAT works with an integrated admission control and reservation scheme since they utilize information about future needs of a particular session. UAT does not consider sharing explicitly, *i.e.*, when the segments of a topic have been played back, they are not preserved in buffer for a future user who might request the same topic. Hence requests are considered independent of each other for admission control and reservation purposes, even if they are for the same topic. The admission control policy checks for each request separately if there is sufficient buffer and disk bandwidth available to fetch and store data for the request on a segment by segment basis (using inequalities 1 and 2). If the required bandwidth is available then the required disk and buffer bandwidths are reserved, else the request is rejected.

4.2 Proposed Schemes

In this section we propose new buffer management schemes which use continuous media caching techniques to explicitly exploit the sharing characteristics of MMDBS applications. These schemes (SHR1 and SHR2) are variations of the UAT scheme we discussed earlier, with enhancements for continuous media caching. The buffer and disk bandwidth for these schemes are determined by utilizing the technique illustrated in *figure 2* (page 5). The difference between SHR1 and SHR2 is in the way they handle segments that might already exist in the free-pool. While SHR1 does not reuse segments that might already exist in the free-pool, SHR2 does. Thus one would expect SHR2 to perform better than SHR1 and we will see later that it does indeed. We do not discuss details of disk scheduling here. For these shared schemes a rate monotonic type of scheduling is the best, *i.e.*, since the disk bandwidth will be assured for the various sessions, each session will be allocated some disk bandwidth corresponding to the ratio of the data rate of the segment it is fetching to the total demanded data rate.

4.2.1 SHR1 - Shared Scheme based on future accesses

Since SHR1 utilizes continuous media caching, segments used by a leading user are explicitly preserved for use by the lagging user. This can be explained better using *figure 3* (page 6). The segments that are explicitly preserved begin with the segment that is being played back by the leading user when the lagging user enters the system. All segments that follow this are also preserved in the buffer till the lagging user needs it for playback. (Referring to *figure 3*, the segments that have been preserved are 3, 4 and 5). Since SHR1 does not reuse segments that might already be in buffer, they (segments 1 and 2 in *figure 3*) are to be fetched from the disk. Since SHR1 considers sharing explicitly, the integrated admission control and reservation scheme has to be modified accordingly. Specifically, in SHR1, the admission control policy checks if there is sufficient disk bandwidth available for segments to be loaded from the disk (segments 1 and 2) and sufficient buffer bandwidth for loading these segments in buffer and preserving the other segments (*i.e.*, loading segments 1 & 2 and preserving segments 3, 4 and 5). If the required bandwidth is available then appropriate disk and buffer bandwidths are reserved, else sharing is not possible. When this is the case, the request in considered independently (similar to the UAT scheme) and an attempt is made to reserve the necessary buffer and disk bandwidth for each request. If this also fails, then the request is rejected.

```
do while there are requests in the queue
    select next request ;
    admitted = false  ;
    For each segment in the topic, check if the segment is already
    in the free-pool and grab it if it is available ;
    For each segment grabbed from the free-pool, modify the
    total profile to reflect preserving of these segments and
    check for buffer bandwidth violation ;
    For segments that have to be fetched from the disk, modify
    the total profile for disk and buffer accordingly and check
    for buffer/disk bandwidth violation ;
    For segments that exist in the buffer and are shareable, modify
    the total profile for buffer and check for bandwidth violation ;
    if no violation at any stage
        admitted = true ;
        /* total profile contains reservation for this request */
    else
        restore old total profile ;
        reject request ;
    endif
end do
```

Figure 4: Admission Control and Reserving future bandwidth

4.2.2 SHR2 - Shared Scheme based on past and future accesses

This uses segments that might already exist in buffer and is an improvement of the SHR1 scheme. Referring back again to *figure 3*, if it happens that some of the segments are in the free-pool (say segments 1 and 2 are in the free-pool after they have been used and returned by user 1), they can be reacquired from the free-pool if the buffers space corresponding to those segments has not already been granted to another session. If all the segments (loaded for the first user prior to this users arrival) are available, then no data is to be fetched from the disk for the second user (only buffer bandwidth is checked/reserved), else data is to be fetched from the disk for those segments that do not exist in the free-pool(both buffer and disk bandwidth checked/reserved). Thus SHR2 uses past and future information about segment access and hence should perform the best as we verify later in the performance tests. *Figure 4* describes the admission control and bandwidth reservation scheme for SHR2. The total profile refers to the total requirements of buffer/disk bandwidth. The figure describes the sequence in which the availability of segments in the buffer is checked along with admission control and buffer/disk bandwidth reservation.

```
do forever
   do for all active sessions
      if buffers are shared with another session
             if leading user
                  preserve the used buffers for lagging users ;
                  acquire new buffers from the free pool ;
                  fetch data from disk into the buffer ;
             else if lagging user
                  if there are more lagging users
                       preserve the used buffers for lagging users ;
                       read from preserved buffers ;
                  else
                       return the used buffers to the free pool ;
                       read from preserved buffers ;
                  endif
                  release current buffers to free pool ;
                  read from preserved buffers ;
             endif
      else
            when a new segment is encountered
                  return the used buffers to the free pool ;
                  acquire new buffers from the free pool ;
                  fetch segment from disk;
      endif
   end do
end do
```

Figure 5: Buffer Allocation & Replacement — SHR2

Buffer allocation and replacement is to be done in an integrated manner as shown in *figure 5*. Buffer allocation is done when a inter segment synchronization point is encountered. Buffers used by a segment that has been played back are freed first and then buffers needed by the next segment are allocated. Once the buffers have been allocated for a segment, they are not freed till the end of the segment. For every user, at the beginning of a data transfer from disk, new buffers have to be allocated. This is achieved by acquiring the necessary number of buffers from the free pool. Once data from these buffers are played back by a user, if there are lagging users, instead of returned them to the free pool, they are preserved in the buffer. For a lagging user, buffers need not be acquired from the free pool, since a disk transfer is not necessary. Instead data is read from the preserved buffers and when the reading is complete these buffers are returned to the free pool. Information about allocated buffers can be maintained using a hash table since it is necessary for sharing buffers. The five schemes we discussed are summarized below.

Scheme	Admission Control & Bandwidth Reservation	Sharing Past Data	Sharing Future Data
FIFO		√	
LRU		√	
UAT	√	√	
SHR1	√		√
SHR2	√	√	√

We do not explicitly consider prefetching of data [22, 23] but if it is considered, then our schemes have to be modified such that prefetching is applied to all or none of the sessions.

5 Performance Tests

We have performed a number of tests to compare the five schemes under various conditions. In this section we first describe the various parameters and metrics used for the tests and then present the results with a discussion.

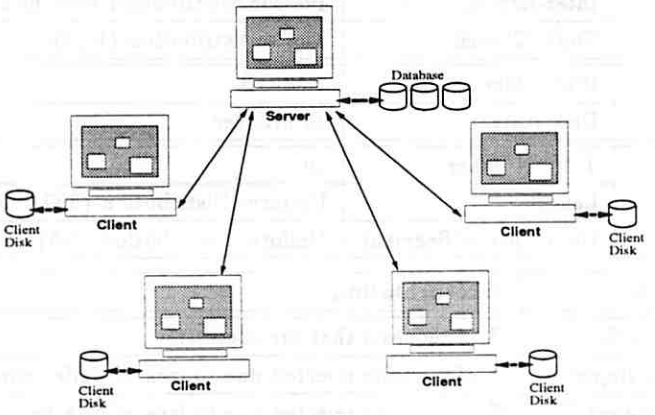

Figure 6: Configuration of the System

We assume a client-server type of architecture shown in *figure 6*. A DMA transfer is assumed between the disk and the buffer at the server and hence the CPU is not likely to be a bottleneck. So the CPU has not been modeled in the performance tests. It is assumed that network between the client and the server provides certain quality-of-service (QOS) guarantees about the network delay and bandwidth. A detailed discussion of reliability and networking issues for continuous media can be found in [3]. We will assume a two level storage system: primary storage (main memory buffer) and secondary storage (disk). Knowing the maximum seek time, rotational latency and data read rate, the effective transfer rate, R can be determined for a storage system. This is used in performing bandwidth allocations for the disk. The secondary storage device could be a magnetic/optical disk or even a RAID based system for increased bandwidth. We do not go into the details of how multimedia data is organized and retrieved from the secondary storage in an efficient manner since there has been extensive research in this area [2, 13, 20, 23, 25, 27] and it is orthogonal to the main focus of this work. We make a few assumptions for simplicity reasons — (1) Since the data rate for video dominates, we assume that requests desiring the same topic with different language options can share the segments. Hence though the topic chosen by the customer is a parameter, we do not consider the language as a parameter. (2) Segments are of equal length (this simplifies our implementation for the performance tests but the results will not be affected).

5.1 Parameters & Metrics

The parameters to be considered fall under three different categories — customer, system and topic. They are listed below in the table with their default values. Topic details (length, data rate, etc.) are generated using a uniform distribution. Customer arrivals are modeled using Poisson distribution. The topic chosen by a customer is modeled using Zipf's law [28]. According to this, if the topics $1, 2, ..., n$ are ordered such that $p_1 \geq p_2 \geq ... \geq p_n$ then $p_i = c/i$ where c is a normalizing constant and p_i represents the probability that a customer chooses topic i. This distribution has been shown to closely approximate real user behaviour in videotex systems [11] which are similar to on-demand services. The disk rate we use here is the average effective transfer rate (this includes seek time and latency apart from the transfer time).

Category	Parameter	Setting
Customer	Inter-Arrival Time	Poisson Distribution with mean of 40 sec
	Topic Chosen	Zipf's Distribution (1-10)
System	Buffer Size	1.28 GB
	Disk Rate	40 MB/sec
Topic	Total Number	10
	Length	Uniform Distribution (500-700) sec
	Data Rate of Segment	Uniform Distribution (2-5) MB/sec

Metric	Interpretation
Success %	% of requests that are successful
Buffer Reject %	% of requests rejected due to lack of buffer bandwidth
Disk Reject %	% of requests rejected due to lack of disk bandwidth

The metrics we use are shown in the table above. Admission control and bandwidth reservation is performed on a segment basis for each request and if the bandwidth is insufficient at any point, the request is rejected. Before a segment is placed in the buffer, the following actions are taken. First the buffer is searched to see if the requested segment exists in buffer. If it does not exist then some victim is chosen in the case of FIFO and LRU or the required buffer space is acquired from the free-pool. If this cannot be done because of the available buffer size then the request is rejected and it counts as a reject due to insufficient buffer bandwidth. When segments have to be fetched from the disk, if sufficient buffer space is available then the disk bandwidth is checked for sufficiency. If the disk bandwidth is sufficient then the request proceeds. If all the segments can be brought in, then the request is successful else the request is rejected and it counts as a reject due to insufficient disk bandwidth. In UAT, SHR1 and SHR2 these checks are done at the beginning of the session as part of the admission control.

5.2 Results

Each simulation consisted of 25 iterations and each iteration simulated 200 customers. The confidence interval tests of the results indicate that 95 % of the values are within 3 % of the mean in almost all cases. The results are discussed based on the graphs that use the success % as the metric. A request can be rejected either due to lack of buffer or lack of disk bandwidth. Due to space constraints, graphs that show this breakup for all the cases are included in the Appendix (referring to these graphs as the results are discussed gives more insight into the dynamics of the schemes). All parameters are set to their default values specified earlier unless explicitly indicated in the graphs. Topic details (the database accessed by users) are generated for every iteration.

5.2.1 Comparison based on Customer Parameters

Figure 7 compares the success percentage of the various schemes for mean inter-arrival times between 20s to 100s. For smaller values of inter-arrival time, higher buffer and disk bandwidth is necessary to satisfy the requests. Hence it can be observed that the success percentage is lower at inter-arrival time of 20s and it gradually increases to 100% for 80s and more. We note that among all schemes SHR2 performs the best. This can be explained by the fact that while SHR1 does not utilize the buffers that exist in the free-pool and UAT/FIFO/LRU do not share buffers based on information about future accesses by a session, SHR2 uses both types of information and hence is able to perform very well. At the inter-arrival time of 20s SHR2 has a 20-40% advantage over other schemes and this clearly indicates that buffer sharing (based on past and future use) pays.

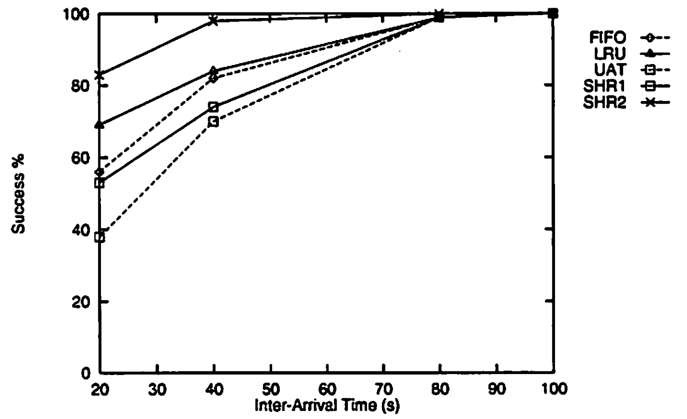

Figure 7: Comparison Based on Inter-Arrival Time

5.2.2 Comparison based on System Parameters

Next we compare the schemes based on the available disk bandwidth. SHR2 was primarily designed to reduce the disk I/Os by preserving buffers in buffer and this test verifies that claim.

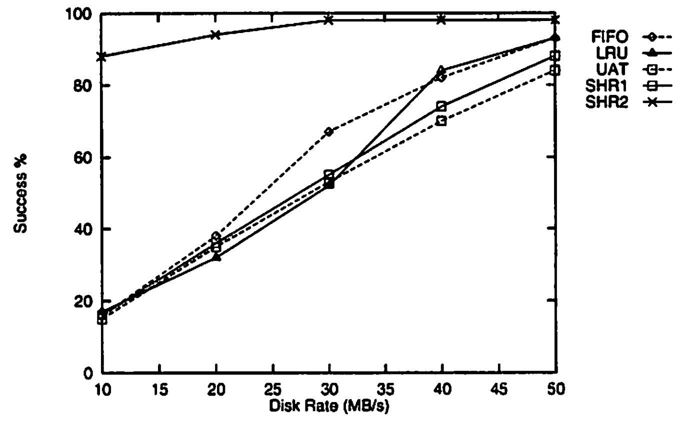

Figure 8: Comparison Based on Disk Bandwidth

It is clear from *figure 8* that SHR2 outperforms all the other schemes at lower disk bandwidths. The figure indicates that when the available disk bandwidth is 10MB/s, SHR2 outperforms the other schemes by 60% which is substantial. As the available disk bandwidth increases, all of the schemes begin to converge. This clearly goes on to prove that by using SHR2, when there is sizeable buffer space, the number of users supported can be increased considerably.

Figure 9 studies the effect of buffer size on the performance of the schemes. In general if the buffer space available is low, all buffer mangement schemes experience poor performance. Furthermore, to exploit sharing, SHR2 needs a sizeable amount of buffer. It can be observed that at lower buffer sizes of 400-800 MB, SHR2 is not able to exploit sharing as much and hence performs similar to the other schemes. However, as the buffer size available increases (800MB-1.28GB), SHR2 uses the extra buffer to exploit sharing and hence begins to outperform the other schemes. It is also interesting to note that the success % of all the other schemes begins to flatten out as buffer size increases due to lack of disk bandwidth. Hence schemes other than SHR2 cannot utilize increased available buffer beyond a certain point without sufficient disk bandwidth.

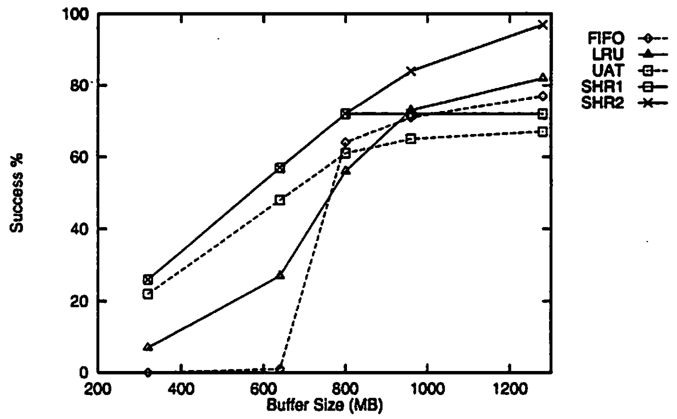

Figure 9: Comparison Based on Buffer Size

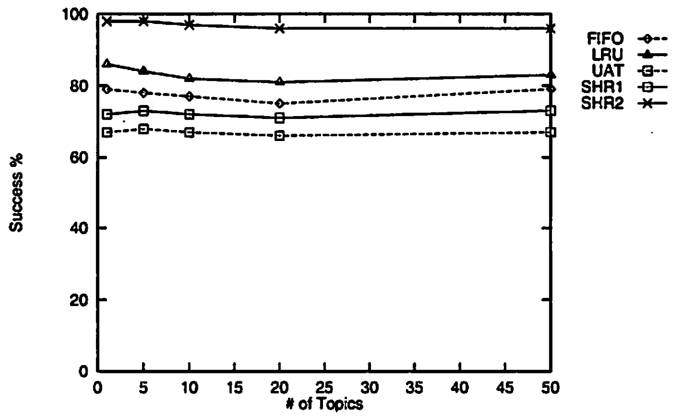

Figure 10: Comparison Based on Number of Topics

5.2.3 Comparison based on Topic Parameters

The next comparison is based on the number of topics available. It was mentioned earlier that the topics chosen by the users follows the Zipf's law. According to this distribution, the top few topics in *demand* have almost the same value of probability even with increased number of topics for selection. Hence it is observed in *figure 10* that the drop in success % is not much for all the cases. If a uniform distribution is chosen, the drop would be steeper. The main observation however is that SHR2 continues to dominate the performance even with increasing number of topics.

Figure 11 explores the effect of the length of the topic on the performance of the different schemes. This is important due to the fact that increasing the length of the topic results in a request taking longer time to complete and hence the number of concurrent customers in the system increases, which in turn demands increased buffer and disk bandwidth. Thus the success % of the schemes drop with increasing length of topic. Observe that while the drop for other schemes is steep, the drop for SHR2 is gradual. All schemes perform well when the length of the topic is between 100-300s. However when the length of the topic is 800s, SHR2 has an advantage of 30-50% over the other schemes.

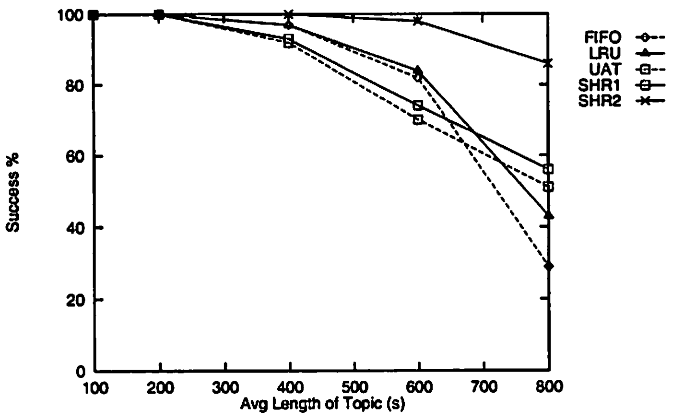

Figure 11: Comparison Based on Length of Topic

Another important parameter to be considered is the average data rate of the segments. Based on the MPEG-2 standards, the average data rate for a compressed high quality video is between 3-5MB/s. Hence we have performed tests for average data rate ranging from 2-10MB/s.

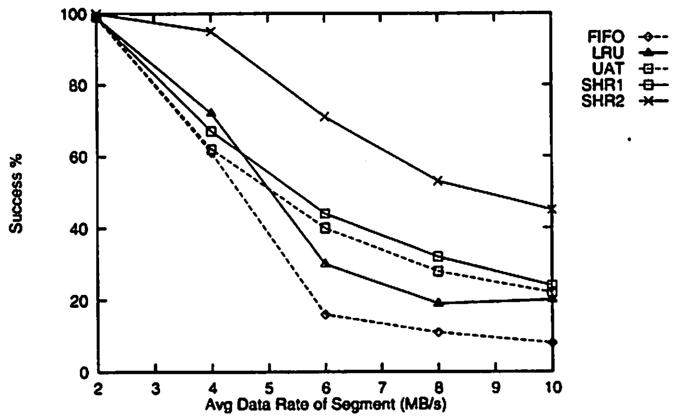

Figure 12: Comparison Based on Data Rate of segment

As expected, we note from *figure 12* that with increasing data rates, all schemes begin to perform poorly. But we observe that SHR2 degrades more gracefully than the other schemes. This is again due to the fact that SHR needs fewer disk accesses compared to the other schemes.

Thus using these tests we have been able to demonstrate that in multimedia information systems where sharing can be exploited, schemes can and must be designed such that they make use of this sharing potential based on information about past and future access to perform more efficient buffer management.

6 Extensions to this Work

We mentioned earlier in section 2 that in the case of NODS, sharing is also possible at the news-item level. *e.g.*, when news items are being shared by the topics. Though we have not performed any explicit tests to investigate this, it would be interesting to determine the amount of performance that can be gained. We intend to evaluate the schemes using other metrics that are more closely tied to QOS guarantees like the following: (1) The maximum arrival rate (minimum inter-arrival time) that can be sustained if 95% of the requests are to be accepted (2) The maximum arrival rate that can be sustained if the requests are queued instead of being rejected and the maximum tolerable waiting time for a user is 10 seconds.

Numerous improvements are possible for the schemes we proposed. To distinguish cases where sharing will be really beneficial, better heuristics can be designed based on the knowledge of topics, data rates, arrival rate etc. Bandwidth reservations can be prioritized based on the frequency of requests for topics. Another variation would be to delay a user for some time units, to explore the possibility of sharing with another user. A centralized server has its limitations and hence to improve the response time, data distribution and replication may be used in which case network delays may have to be considered during the analysis. Data distribution may be used to divide the load among the servers, by storing different most frequently requested topics on different servers.

The concept of continuous media caching and the buffer management schemes that utilize this are applicable in many other scenarios. A modification of this technique is applicable for storage management in a multi-level hierarchy, *i.e.*, data management on the secondary device while data is fetched from a tertiary device. This concept is likely to be useful for video-on-demand services (VODS) when some of the titles shown are in heavy demand. However VODS requires support for fast-forward and rewind and hence enhancements are necessary for admission control, bandwidth reservation and buffer allocation/replacement. While this is a topic of research, a few solutions can be perceived. To accommodate fast-forward, increased bandwidth may have to be reserved for each topic, *e.g.*, disk bandwidth reserved should be the maximum of all the segments. Similarly, for rewind, used data can be cached on the local disk and reused. Another issue to be considered in the context of continuous media caching is the way in which the schemes must be changed if there are alternatives paths available at intermediate points of the request.

7 Conclusion

Buffer management schemes for MMDBS are complex because they handle data that have timeliness and synchronization requirements. Since the data rates of multimedia data is very high, the storage system is likely to limit the number of concurrent sessions supported by the system. To make efficient use of data that has been fetched into the buffer, by considering the applications of NODS, we explored the potential benefits of continuous media sharing to reduce the number of disk I/Os and hence support more number of users. We effectively utilize the advance knowledge of the multimedia stream to be accessed to promote sharing. Using the simple technique of continuous media caching (planned sharing) that preserves buffers in a controlled fashion for use by subsequent users requesting the same data, we proposed two new buffer management schemes, SHR1 and SHR2. These schemes have admission control and bandwidth reservation integrated with them. Also in addition to planned sharing, SHR2 utilizes data that might already be in the buffer. Hence SHR2 is the best multimedia buffer management scheme when sharing is possible and has superb performance.

To compare the performance of our proposed schemes with traditionally used schemes, we conducted a variety of performance tests. The parameters we used for the tests include — buffer size and disk rate (system parameters), mean arrival time (customer parameter), number of topics, length of topic and data rate of segment (topic parameters). The metrics we used for the tests are success percentage of requests, disk and buffer reject percentages. We observed that SHR2 performs very well in all cases since it uses shared data to the best possible extent — by using data that might already be available in the buffer when the request arrives and utilizing continuous media caching for future accesses. Specifically we observed that the disk reject percentage for SHR2 is low in all cases, thereby verifying the fact that SHR2 reduces the number of disk I/Os. Hence, even at low disk bandwidth SHR2 can give good performance. Also if the data rate increases, SHR2 degrades more gracefully compared to the other schemes. These tests indicate that by exploiting continuous media sharing, substantial performance benefits can be achieved. In summary, to increase the number of users supported, instead of enhancing the storage system to provide additional bandwidth, by adding some extra memory and by exploiting data sharing opportunities, higher performance can be achieved at lower costs.

Through this work, we hope to have motivated the significance of buffer management and continuous media sharing in the context of complex multimedia information systems which deal with huge amounts of data. Designers of these systems should consider this and exploit the characteristics of the multimedia application whenever possible to get the best performance out of their systems.

References

[1] J. F. Allen, "Maintaining Knowledge about Temporal Intervals", in *Communications of the ACM*, Vol 26, No 11, November 1983, pp 832-843.

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

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

[4] P. B. Berra et al, "Issues in Networking and Data Management of Distributed Multimedia Systems", in *Proceedings of Symposium on High Performance Distributed Computing*, September 1992.

[5] S.Christodoulakis, "Multimedia Data Base Management: Applications and Problems - A Position Paper", in *Proc. of ACM SIGMOD*, 1985, pp 304-305.

[6] S.Christodoulakis et al, "An Object-Oriented Architecture for Multimedia Information Systems", in *The Quarterly Bulletin of the IEEE Computer Society Technical·Committee on Data Engineering*, Vol. 14, No. 3, September 1991, pp 4-15.

[7] N. Davies and J. Nicol, "Technological Perspectives on Multimedia Computing", in *Computer Communications*, June 1991.

[8] W. Effelsberg and T. Haerder, "Principles of Database Buffer Management", in *ACM Transactions on Database Systems*, Vol. 9, No. 4, December 1984, pp 560-595.

[9] L. Felician, "Simulative and Analytical Studies on Performances in Large Multimedia Databases", in *Information Systems*, Vol. 15, No. 4, pp 417-427, 1990.

[10] Edward Fox, "Advances in Interactive Digital Multimedia Systems", in *IEEE Computer*, October 1991.

[11] J. Gecsei, *The Architecture of Videotex Systems*, Prentice-Hall, Englewood Cliffs NJ, 1983.

[12] S. Gibbs, C. Breiteneder and D. Tsichritzis, "Audio/Video Databases: An Object Oriented Approach", in *Proceedings of 9th International Conference on Data Engineering*, Vienna, April 1993, pp 381-390.

[13] J. Gemmel and S. Christodoulakis, "Principles of Delay-Sensitive Multimedia Data Storage and Retrieval", in *ACM Transactions on Information Systems*, Vol. 10, No. 1, January 1992.

[14] S.Ghandeharizadeh et al, "Object Placement in Parallel Hypermedia Systems", in *Proc. of 17th VLDB Conference*, Barcelona, Sept. 1991, pp 243-254.

[15] J. Huang and J. A. Stankovic, "Buffer Management in Real-Time Databases", *Technical Report 90-65*, Department of Computer Science, University of Massachusetts, July 1990.

[16] K. Kawagoe, "Continuous Media Data Management", in *SIGMOD Record*, Vol. 20, No. 3, September 1991, pp 74-75

[17] W. Klas, E.J. Neuhold and M. Shrefl, "Using an Object-Oriented Approach to Model Multimedia Data", in *Computer Communications*, Vol. 13, No. 4, pp 204-216, 1990.

[18] D. L. Gall, "MPEG: A Video Standard for Multimedia Applications", in *Communications of the ACM*, Vol. 34, No. 4, April 1991, pp 46-58.

[19] T.D.C.Little, A.Gafoor et al., "Multimedia Synchronization", in *The Quarterly Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, Vol. 14, No. 3, September 1991, pp 26-35.

[20] P. Lougher and D. Shepherd, "The Design of a Storage Server for Continuous Media", in *The Computer Journal*, Vol. 36, No. 1, 1993.

[21] W. Putz and E. Neuhold, "is-News: a Multimedia Information System", in *The Quarterly Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, Vol. 14, No. 3, September 1991, pp 16-25.

[22] A.J. Smith, "Sequentiality and Prefetching in Database Systems" in *ACM Transactions on Database Systems*, Vol. 3, No. 3, September 1978, pp 223-247.

[23] R. Staehli and J. Walpole, "Constrained-Latency Storage Access", in *Computer*, March 1993, pp 44-53.

[24] R.Steinmetz, "Synchronization Properties in Multimedia Systems", in *IEEE Journal on Selected Areas of Communication*, Vol.8, No.3, April 1990, pp 401-412.

[25] P. Venkat Rangan and Harrick M. Vin, "Efficient Storage Techniques for Digital Continuous Multimedia", in *IEEE Transactions on Knowledge and Data Engineering*, August 1993.

[26] D.Woeld and W.Kim, "Multimedia Information Management in an Object-Oriented System", in *Proc. of the 13th VLDB Conference*, Brighton, 1987, pp 319-329.

[27] C. Yu et al, "Efficient Placement of Audio Data on Optical Disks for Real-Time Applications", *Communications of the ACM*, Vol. 32, No. 7, July 1989, pp 862-871.

[28] G.K. Zipf, *Human Behaviour and the Principles of Least Effort*, Addison-Wesley, Reading MA, 1949.

Appendix

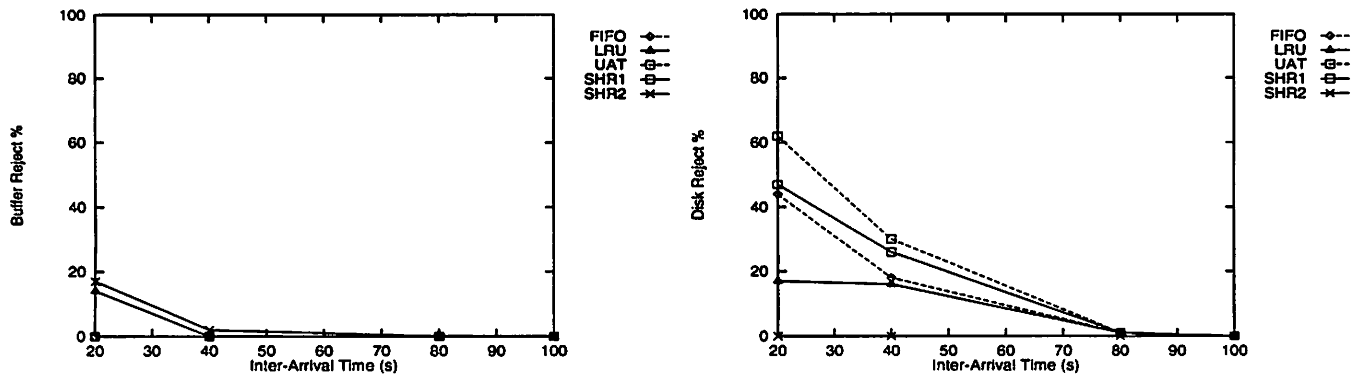

Figure 13: Comparison Based on Inter-Arrival Time

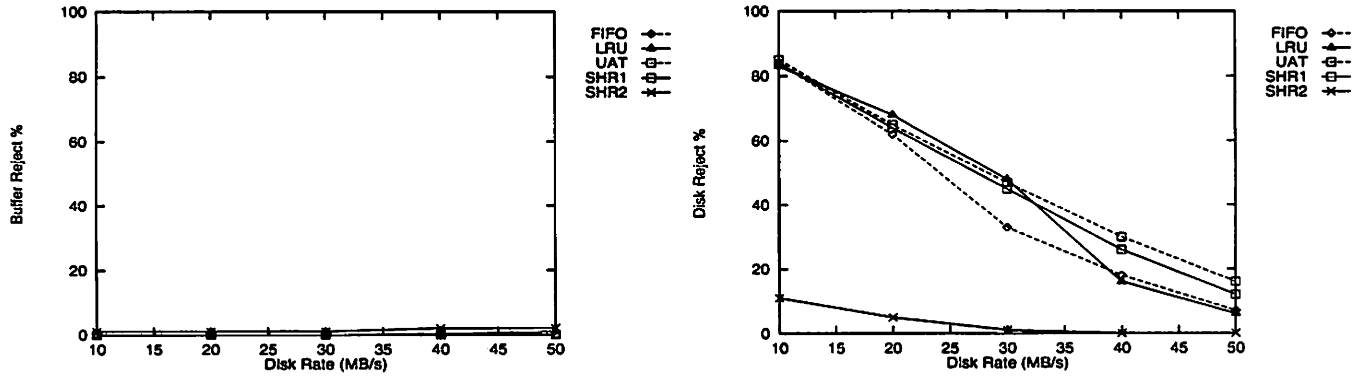

Figure 14: Comparison Based on Disk Bandwidth

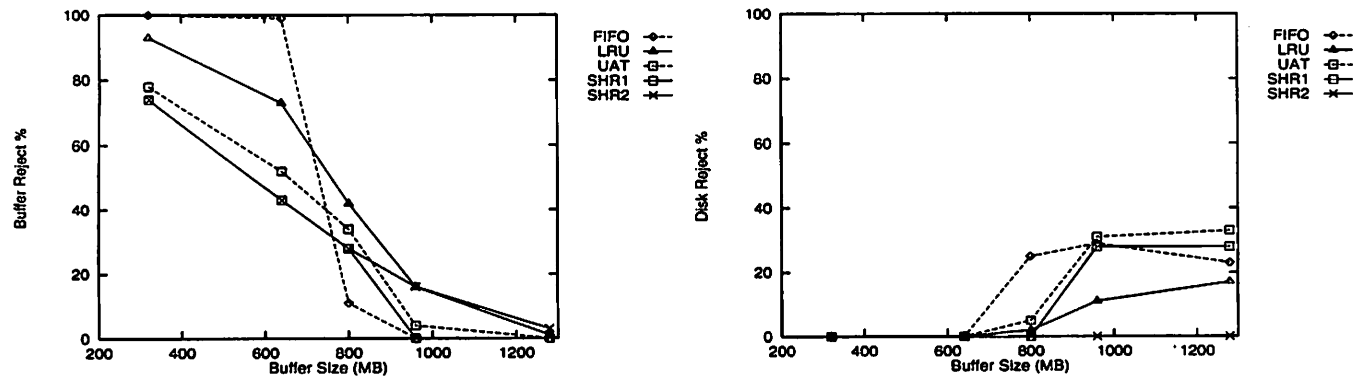

Figure 15: Comparison Based on Memory Size

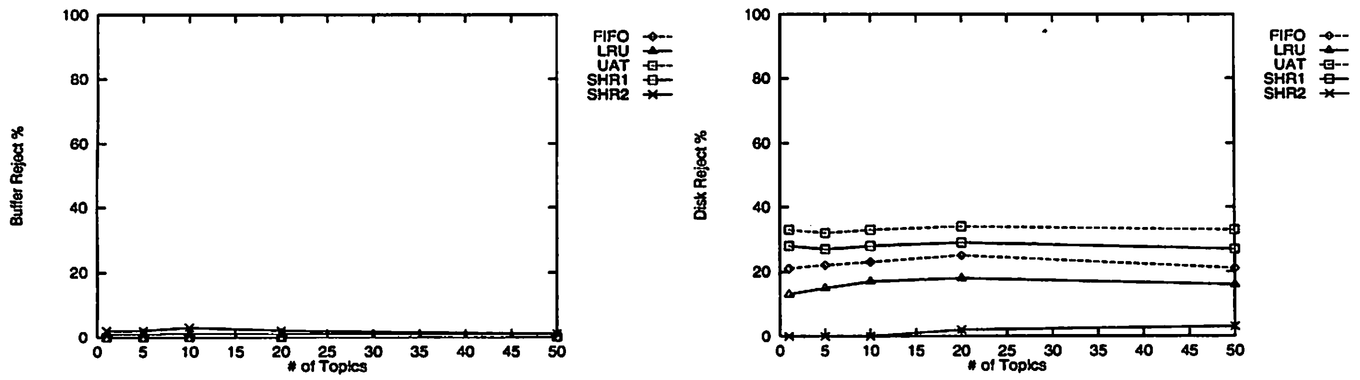

Figure 16: Comparison Based on Number of Topics

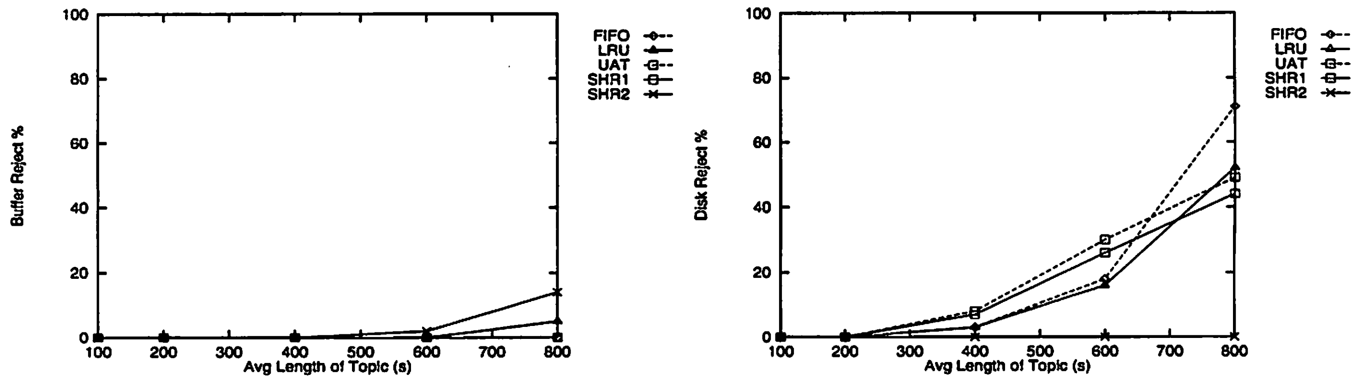

Figure 17: Comparison Based on Length of Topic

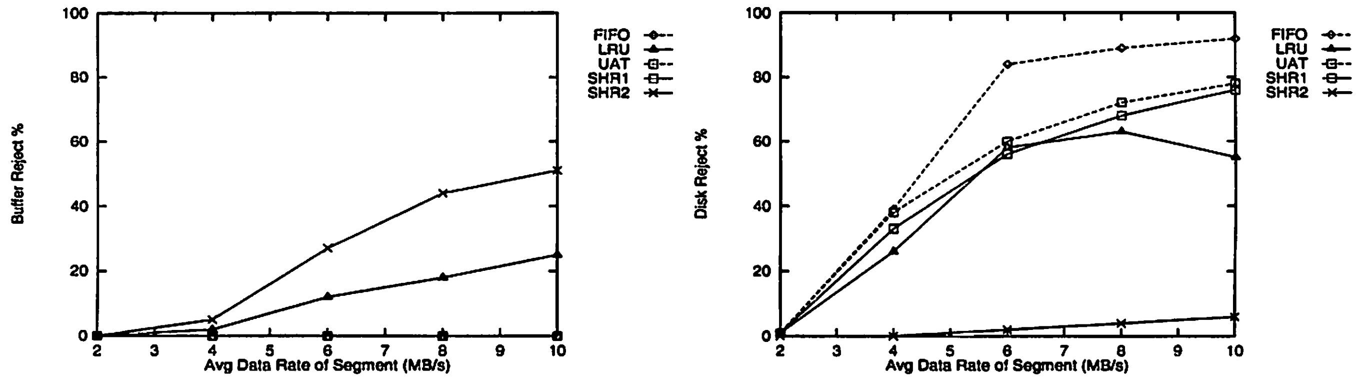

Figure 18: Comparison Based on Data Rate of segment