ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

Rainer Manthey and Viacheslav Wolfengagen (Eds)

Advances in Databases and Information Systems 1997

Proceedings of the First East-European Symposium on Advances in Databases and Information Systems, (ADBIS'97), St Petersburg, 2-5 September 1997

The Design of a High Performance Video Server for Managing TV Archives

Ahmed Mostefaoui

Published in collaboration with the
British Computer Society

©Copyright in this paper belongs to the author(s)

ISBN: 3-540-76227-2

The Design of a High Performance Video Server for Managing TV Archives

Ahmed Mostefaoui

Laboratoire de l'informatique du Parallélisme, Ecole Normale Supérieure de Lyon
France

Abstract

One of the key components in a multimedia interactive system is the continuous media server. In this paper, we first present the design and the architecture of such a server dedicated to support the storage, retrieval and delivery of video streams. The main feature of the proposed server is its independence of any physical configuration, so guaranteeing its portability. After that, based on the target application (video repositories), we have proposed a new scheduling technique that takes into account the data structuration. To cope with the difficulties implied by the content-based retrieving, a modeling process divides a video in several overlapping streams. We argue that the server can use this knowledge in order to promote media sharing between users requesting overlapped streams. The simulation results show the effectiveness of the proposed approach.

1 Introduction

Recent advances in computing and communication technologies have made it possible to provide on-line access to video clips. This is due to the availability of hardware and software tools (high bandwidth networks, new storage devices and technologies,...) for handling digital video and audio data. The advantages of digital storage of continuous data are numerous [13]. In fact, many applications use continuous data, ranging from professional activities to entertainment. All of such applications need a unified information system called *multimedia information system* which integrates communication, storage and presentation mechanisms for diverse data type including text, images, audio and video. To support audio and video data, multimedia database management systems need to deal with several tough issues. First, continuous data (audio and video) are delay-sensitive. Once started displaying data, the system must guarantee that enough resources (disk bandwidth, buffer space, etc.) are available so that the real time requirements and continuity are not violated. Second, continuous data (even compressed) have a huge volume and consequently consume large amounts of system resources mainly storage space and bandwidth. Third, due to the data type diversity of a multimedia objet (text, video, audio, etc.), the system must ensure that these different streams can be synchronized during presentation. Regardless these inherent requirements, the main objective of such systems is to deal with video information retrieval difficulties. For instance, one is trying to locate a specific clip in a video source, finding it can be not only a time-consuming and tedious process but also unreliable.

In cooperation with the French institute, INA and the TV broadcasting corporation, FRANCE3, we will develop a video repository where users (mainly reporters) can have a on-line access to video archives. Noted that INA is responsible for storing, under perfect conditions, all TV programs broadcasted by French broadcasting corporations and for making them accessible. Naturally, this involves developping a distributed multimedia system which allows users to search and browse the video archives. Obviously, the system will provide facilities for manipulating, storing and easily retrieving video data. Hence, it deals with several research areas ranging from data modeling to data storing at the disk level.

In this paper we first present the design of the system and its related aspects. Then, we focus on the video server which is responsible for storing and delivering continuous data within the time limits imposed. We propose a novel scheduling technique based on the use of data structuration. To cope with the difficulties

implied by the content-based retrieving, a modeling process divides a video in several overlapping streams. We argue that the server can use this knowledge by promoting media sharing between users requesting overlapped streams. Sharing of data can occur at various levels in TV repositories application. Sharing can occur when some news items are shared between topics, *e.g.*, news on a international trade agreement can be included in a variety of topics like sports, politics, business, etc. Sharing of continuous media can also occur at the media level, *e.g.*, for news items in different languages, additional storage is necessary only for the audio segments as the video segments can be shared. Those kinds of sharing can be handled by the modeling process [6]. In addition, sharing is also possible when there are multiple requests for the same topic. This is particularly true in TV repositories application. In this paper, we focus on this type of sharing since this is likely to be more frequent and we explore the potential benefits that can be achieved through using data structuration.

The rest of the paper is organized as follows. Section 2 describes the work related to this study. Section 3 presents a general overview of the proposed system. In section 4, we focus on the video server and its related components. Section 5 describes our proposed approach followed by the preliminary simulation results. Section 7 concludes this paper and outlines our future work.

2 Related Work

Video data management has been a great topic for research during the last years. Many researchers have proposed various approaches to handle the requirements needed by the storage and retrieval of continuous data [1, 8, 4]. Most of those approaches focus on how to support Video-On-Demand services which allow users to carry and watch videos stored on digital servers [12, 14]. Thus, the main interest has been focused on the development of storage strategies and optimizing of the network bandwidth in order to support as many simultaneous video streams as possible. To achieve this, different architectures and techniques have been proposed. Anderson et al. [1] have established the basic concepts for designing a file system to provide hard guarantees for continuous media. Ghandeharizadeh and Ramos [4] have tackled the problem of maximizing retrieval bandwidth by striping continuous data across several disks in a round robin fashion. This enables the system to utilize the aggregate bandwidth of those disks to retrieve an object in real-time. Reddy and Willy [15] have proposed a hybrid disk arm scheduling called Scan-EDF. Jadav et al. [7] have proposed a logical model of a multimedia on-demand server which exploit data access patterns to maximize the number of simultaneous streams served. To a very large extent, most of these proposals aim to minimize seek latencies so as to satisfy the real-time requirements of multimedia streams. However, none of the previous approaches has explored the potential benefits that can be achieved through using knowledge about data structuration. The motivation for doing that comes from the following features of the target application :

- Small granularity : instead of a long video documents (more than 1h of duration in general) , TV archives are mostly composed of small video documents (less than 5 min in general). Furthermore, they are fragmented by a modeling process in smaller streams in order to support content-based retrieval [5].
- High interactivity : in a typical request scenario, users ask for videos related to specific topics (for instance plane crashes). They view the corresponding streams (not necessary all) in order to select those desired. Then they refine their query and so on until they find what they are searching for.

Obviously, the techniques proposed in those studies can be used in our model, especially data striping, round robin distribution of successive stream fragments, and contiguous allocation within a given fragment.

3 Overview of the Distributed System

In this section, we present the architecture of the proposed system. The design process was mainly influenced by the need of a flexible and extensible architecture so that it will be possible to port it on different physical configurations. Figure 1 shows the overall architecture of the system which we consider. It is divided into three components :

Figure 1: System architecture

• Archive Server

As mentioned above, the volume of data to be stored is of the order of several Tera-bytes¹ (1 TB = 1000 GB). Storing all these data on disks is definitely too expensive and unreliable. Therefore it is necessary to include storage devices that offer a high capacity with relatively cheaper cost. For example, tape robots [9] will be good candidates. Providing this level of storage allows to cope with the following problems :

- In a multi-user environment, users may ask for different video formats depending upon the target use. Such situation occurs frequently in managing video archives. One solution is to store all the formats of a video document in the server and then creating a catalog to manage them as proposed in [16]. This solution is too expensive in term of storage space and suffers from rigidity (e.g. adding of a new format). Oppositely, the archive server can store data in uncompressed format (tape) and consequently provide the possibility to offer to clients all needed formats. Of course, this means to get a on-line appropriate compression software. Thus, the compression is triggered only when it is necessary² and in the requested format.
- One key of the success of a video system is its availability. Many research studies have tackled this problem due to hardware as well as software failures³ [17]. In our model, the problem of availability is solved by the archive server. Indeed, when a storage component in the video server falls down, the data are still available on the archive server. Thus the system is safe and does not need sophisticated techniques (e.g. redundancy codes as used in RAIDs) to recover lost data.

• Video server

¹For instance, INA archives each year nearly 6 Tera-bytes of compressed copies of broadcasted programs.

²If the requested format is different from the format managed in the video server.

³Typical MTTF (Mean Time To Failure) values for a disk is 300,000 hours, while for a node (including software failure) is around 18,000 hours [2]

At the heart of the system is a high performance video server optimized for a large throughput and fast I/O. The server is based on logical components called *storage components*. Their main task is to temporarily store data (loaded from archive server) and deliver them to remote clients. Of course, each storage component has a large storage space ranging from a high single disk to a disk-array. This allows parallelism of data retrieval that can be obtained by data striping across storage components. Similarly, each storage component has its own buffer space. Noted that the server is responsible for serving client requests with the guarantee that all continuity requirements will be satisfied. A detailed description of the server components is given below.

- *Intelligent interface*

The remote clients are connected to the server *via* a high-speed wide-area network. As the data is stored at the video server and transmitted in compressed digital form, the decompression is done at the remote client's multimedia terminal. The latter is a computer, fully equipped to handle multimedia data (stereo speaker, high resolution monitor, etc.). At the client level, the aim is to develop an intelligent interface that can take profit from the manner how the user asks for videos. This is particularly helpful when the most accessed data are stored at the client computer reducing in this way the response time and the access to the server.

4 The Video Server

As mentioned, the goal of the video server is to maximize the number of simultaneous real-time streams that can be sourced to clients. To achieve this, we propose a *logical* model similar to that proposed in [8], which is independent of the architectural implementation. Indeed, the same model can be implemented on a parallel machine or a collection of PCs/workstation interconnected by a high-speed switches such as myrinet [11]. The logical model of the server is based on a number of storage components interconnected by a high-speed interconnection network. The main advantage is that this model might provide for scalability. As the demand increases, new storage components can easily be added to the video server, which is therefore able to serve more users. All these storage components are *governed* by a node called *Global Object Manager*. In figure 1, solid lines indicate data traffic, while dotted lines indicate control traffic. Below, we present in greater detail the description of each component :

- *The Global Object Manager (GOM)* is at the top of the server hierarchy. It is responsible for scheduling all incoming requests for media objects onto storage components. To do so, it has a knowledge of which storage component the media object resides and the workload of all storage components. Once a request has been accepted, the GOM *delegates* the responsibility of serving it to the storage component chosen. It also has a knowledge of data structuration and client's request patterns and uses it to improve the server response time and throughput. This will be explained in section 5.
- *The storage components* are responsible for storing and serving media objects to clients whose requests have been accepted. The video streams are striped across storage components in a round-robin fashion. As mentioned in [4], this strategy allows to tackle the problem of low speed of data transfer from secondary storage to main memory by using parallelism of retrieval. Thus, the effective retrieval bandwidth is proportional to the number of storage components. Furthermore, since the stripe fragments on any given storage component are not consecutive fragments, it is not necessary to store them contiguously. Each of storage components is composed of :
 1. The Storage Manager (SM) retrieves media streams from storage space and transmit the data to the *Local Object Manager*. If the requested data are not on the disks, the storage manager can then ask the archive server to get them. Different strategies for managing the archive server have been proposed in the literature, like those reported in [3], and can be used in our model. It is to be noted that the single disk assumption is not literal and the storage manager can have a disk-array instead

for greater I/O throughput. In addition, disk scheduling algorithms to optimize retrieval from the disk surface have been proposed [15], and can be used by the storage manager.

2. The Local Object Manager (LOM) is responsible for serving stream requests that have been accepted. To do that, he asks others LOMs for stream fragments as the data are striped across storage components. Once all requested data are ordered, the LOM begins serving the client. All those tasks will be performed within the real-time requirements. Noted that the LOM is also responsible for buffer management.

In a typical request-response scenario, the GOM (Global Object Manager) first receives a request from a client interface for an objet, M . Then it chooses a storage component for serving the object M . To do this, the GOM needs to have a global view of the server (location of the objects, workload of each storage component). According to that knowledge and the resource requirements needed for processing the request, this latter is accepted or rejected. The admission control policy depends on the physical state of the server (available bandwidth and buffer space). Once a request is accepted, the GOM *delegates* it to a storage component with addition information (location of the object M fragments for example) for serving it.

5 Exploiting Data Structuration

As mentioned above, video data managed in news-on-demand systems (NODS) have a complex structuration. This is due to the modeling process which leads to a overlapping streams. Noted that the video server has no knowledge about this structuration. Consequently, the request-response scenario, exposed above, relates only to the objects stored without considering the relationship that might exists between them. We argue that the video server can take benefits from this structuration especially the overlapping between streams to improve the server performance. The key idea behind is to promote a media sharing between users requesting overlapping streams. We illustrate this notion by the following example. Assuming a client **U1** would request for a video stream **V1** that contains an other video stream **V2**. Now assuming that the latter is requested by client **U2**. Then the typical behavior of the video server is to load the **two** video streams, **V1** and **V2**, instead of loading only **V1**. The reason for this is that the server considers these two objects like two different streams. In this case, the server does not use the knowledge about the data structuration to improve the response time.

At the server level, the most relevant information about the data structuration is the overlapping between streams. At that stage, we deal only with basic streams *e.g.*, the virtual streams which result from a virtual editing process [10] are out of interest because they are not managed at the server level. Before we discuss how to use data structuration, we first give a preliminary analysis about streams overlapping.

5.1 Video Segment Sets

A video is composed of a set of streams. Each of them is associated with a time interval $[t_s, t_f]$ where t_s and t_f are the relative discrete time instants in that video and $t_s < t_f$. To simplify the notation, we use $S_i = [t_{s_i}, t_{f_i}]$ to mean that segment S_i is associated with a time interval $[t_{s_i}, t_{f_i}]$. We define four relationships over the set of streams as follow :

DEFINITION 1. Let $S_i = [t_{s_i}, t_{f_i}]$ and $S_j = [t_{s_j}, t_{f_j}]$ be two streams. Then S_i is said to *precede* S_j , noted $S_i \preceq S_j$, iff $t_{s_i} \leq t_{s_j}$. Respectively, S_i is said to *strictly precede* S_j , noted $S_i \prec S_j$, iff $t_{s_i} < t_{s_j}$.

DEFINITION 2. Similarly, S_i is said to *succeed* S_j , noted $S_i \succeq S_j$, iff $t_{f_i} \geq t_{f_j}$. Respectively, S_i is said to *strictly succeed* S_j , noted $S_i \succ S_j$, iff $t_{f_i} > t_{f_j}$.

DEFINITION 3. S_i is said to *contain* S_j , noted $S_i \supset S_j$, iff S_i precede and succeed S_j at the same time *i.e.*, the both conditions, $t_{s_i} \leq t_{s_j}$ and $t_{f_i} \geq t_{f_j}$, hold.

DEFINITION 4. Two streams S_i and S_j are said to overlap, noted $S_i \cap S_j$, iff one of the following hold :

- $S_i \preceq S_j$ and $t_{s_j} < t_{f_i}$, or
- $S_i \succeq S_j$ and $t_{s_i} < t_{f_j}$.

This means that one segment starts before the end of the other, which leads to a overlapping between these streams. It is to be noted that if the two conditions hold, the overlap is total *i.e.*, the segment S_i contain the segment S_j .

Based on these relations, we define for every segment the set of all streams which overlap with it. Hence, the GOM knows, if the overlap is total, whether streams have been loaded by loading others streams or, if the overlap is partial, whether parts of streams have been loaded. In the latter case, this knowledge allows it to anticipate the loading of streams which have a higher overlap, improving by the way the response time. Consequently, it is necessary to *quantify* the overlap over streams. We define the *Degree of Overlapping* (DoO) as fellows.

DEFINITION 5. For every two streams S_i and S_j , such that $S_i \cap S_j$, the *degree of overlapping* for S_i with S_j is defined as $DoO_{(S_i, S_j)} = \frac{\text{length of overlap}}{\text{length of } S_i}$.

Figure 2 shows an example of the DoO between two streams. Obviously, if the overlap is total *i.e.*, one segment contains the other, the DoO is then equal to 1. The GOM maintains for every segment, S_i , a set of 3-uples, called *set of overlap*, for all streams which overlap with it. An 3-uple is composed of (see example in figure 2) :

1. The segment identifier, says S_j ,
2. The relation (succeed, precede, contain) between the streams S_i and S_j ,
3. The degree of overlap $DoO_{(S_i, S_j)}$.

Below, we discuss how we use those relations in our model and their effect on the admission control policy and the scheduling algorithms.

5.2 Scheduling algorithm

As we mentioned it, the contents of the buffers evolves according to the requests. Hence, in the above request-response scenario, the streams that have been served by a LOM are not *explicitly* preserved in the buffer and consequently their space is returned to the free-space. Furthermore, according to our target application, the sizes of streams are relatively small. Then, Since we assume a high buffer space, some objects can be preserved in the buffer in order to be shared by subsequent requests. Two criteria can determine whether preserving an object in the buffer space. The first criterion relates to the ability to detect that some streams are being to be accessed more frequently than others. This is achieved by logging the object access patterns over a specified *time* as proposed in [8]. The second criterion, which we introduce, is based on the overlapping between objects. A stream that had been accessed and which had a higher DoO with other streams, is preserved in the buffer. This task is performed by the GOM since it schedules all incoming requests. To do that, the GOM maintains for every LOM a list of streams, called *Stream-List (SL)*, those are available in its buffer space at an instant t . Based on the sets of overlapping, the lists (SL) are updated accordingly. Hence, the scheduling algorithm is not only based on the workload of the LOM but also on the presence of the objects in the LOM buffer or that have a high DoO. Figure 3 presents our proposed scheduling algorithm.

5.3 Admission control policy

Once a request has been scheduled to a LOM, the GOM will check if there are enough resources (buffer space and available bandwidth) to sustain the new request, while still guaranteeing undisturbed service to the existing streams. In that case, two conditions must be satisfied. First, there must be sufficient buffer space to satisfy

Three temporal relations : Precede, Succeed and Contain

3-uple ('Stream identifier', 'Temporal relation', 'Degree of Overlapping')

Sets of overlap :

- Set-Overlap-S1={ (S2,P,2/4),(S3,C,1),(S4,C,1)}
- Set-Overlap-S2={ (S1,S,2/7),(S3,C,1),(S4,C,1)}
- Set-Overlap-S3={ (S1,S,4/17),(S2,S,7/17),(S4,S,15/17)}
- Set-Overlap-S4={ (S1,S,4/22),(S2,S,7/22),(S3,P,15/22)}

objects

Figure 2: temporal relation between streams and their degree of overlapping

buffering requirements of the new stream. Second, there must be enough bandwidth to sustain the need of the new request. These criteria are explained below :

- Given n streams in process by a LOM, and B_i is the buffer space occupied by the stream $S_i, i=1,...,n$, then in order to start serving a new stream request, B_{n+1} , there should be sufficient buffer space for the new stream :

$$\sum_{i=1}^{n+1} B_i < B_{total}$$

where B_{total} denotes the total buffer space.

- Similarly, if R_i denotes the bandwidth reserved by each of the m streams in process in the system, $i=1,...,m$, then there must be sufficient bandwidth for the new stream request R_i so that

$$\sum_{i=1}^{m+1} R_i < R_{total}$$

where R_{total} denotes the total bandwidth available.

In our model, since the data are striped across storage components which are interconnected by a high-speed network, the total bandwidth depends on both the disks bandwidth and the network interconnection bandwidth. In other terms, the traffic on the network affects the real available bandwidth. Furthermore, the derivation of admission control criterion for the interconnection network is highly dependent on the network-specific factors like topology, routing, and the switching techniques used. Since our model is a conceptual model, we do not go in details about the effect of the interconnection network on the admission control policy. Such a study can be found in [8]. Here, we deal only with the available bandwidth which might include the interconnection network traffic. In a traditional admission control policy, when a new stream request arrives, the system will check if there are enough resources to serve this request. This means that the two inequalities above must hold. However, by using data structuration, the admission control policy proposed proceeds differently. Although, if there are streams that have been loaded in the buffer and those streams overlap with the requested stream, the admission control policy checks resources only for the **un-overlapped parts** of the stream. If the overlap is total *i.e.*, the stream is available in the buffer, the latter can be served immediately without checking for resources.


```

Do while there are requests in the waiting queue
  select next request R
  check if R is present in one Stream-List
  if yes
    begin serving it
  else
    for every stream S in the set
      of overlap of R do
        check if S is present in one Stream-List
        if yes
          check the admission control for
            the un-overlapped parts
          if OK
            begin serving it
            update Stream-List
            exit
          endif
        endif
      endfor
    if R has not been admitted

      /* classical scheduling algorithm */
      for every LOM do
        check the admission control for R
        if OK
          update Stream-List
          begin serving it
          exit
        endif
      endfor
    if R has not been admitted
      reject request
    endif
  endif
endwhile

```

Figure 3: scheduling algorithm.

Category	Parameter	Setting (default values)
System	Number of storage components	4
	Buffer size at each LOM	64 MB
	Bandwidth	<determined using tests>
Data	Data rate	MPEG1 (1.5 MB/s)
	Number of videos	Uniform Distribution (50-100)
	Length of a video	Uniform Distribution (32 mn-52mn)
	Total number of streams per video	Uniform Distribution (12-20)
	Length of a stream	Uniform Distribution (120s-240s)
User	Stream chosen	Uniform Distribution (nb. streams)

Table 1: Simulation parameters with their default values

6 Preliminary Simulation Results

In order to evaluate the techniques proposed in this paper, we have implemented a discrete-event simulation model. First, we discuss our experimental assumptions and describe the various parameters. Then, we present our simulation results.

6.1 Simulation settings

At each LOM, we assume that the disk has the necessary bandwidth to support video rates. Furthermore, since the data are striped across storage components, the disk bandwidth cannot be a bottleneck. However, as we noted it above, the bandwidth depends on the interconnection network, hence the total bandwidth is a function of the interconnection network. To make our simulation as close to reality as possible, the bandwidth was determined using tests performed on a four Pentium PC's interconnected by a myrinet network [11] and each of which had a single disk. We assume that a user can display only one stream at a time. Consequently, once a user issues a request, he does not issue another until the first one is serviced (closed system). We also assume a zero think time between the requests. This parameter was chosen in order to stress the system. One important issue in our model is the seek time needed for the GOM to perform the scheduling algorithm since he must *keep* all streams with their sets of overlap. With regard to the number of streams a video repository can contain, this information might be stored in the disk as a database. However, since the overlap can occur only between the streams of a same video, the GOM can load, once a stream of a video has been accessed, all the sets of overlap related to that video. Hence, the seek time is reduced to a main memory access. For this reason the seek time was not considered in our simulation model.

The parameters to be considered fall in three categories -*system*, *data* and *user*. They are listed in table 1 with their default values.

The system parameters were set based on practical considerations. In cooperation with our partners, we have tried to obtain some interesting statistics for the data structuration. The default values listed in table 1 are typical values. The users consider this as a good trade-off between searchability and effort needed to annotate the streams *i.e.*, the modeling process.

6.2 Results

We performed several series of simulation and we evaluated the effectiveness of our technique. Here we analyze the behavior of the proposed scheduling algorithm and discuss how it helps to improve the system performance, especially the total bandwidth needed, the number of supported queries and buffer space used.

6.2.1 Bandwidth needed

We mean by the bandwidth needed, the bandwidth necessary for a LOM to load striped fragments of a requested stream in its buffer. Consequently, once all striped fragments are loaded in the buffer the bandwidth reserved is released and the request can be served. Thus, not every activated query decreases the total bandwidth available but only those are not loaded in the buffer. Figure 4 presents the performance of using the data structuration.

Figure 4: benefit of using data structuration on total bandwidth needed.

Those results were generated for one LOM. The figure shows clearly the effectiveness of the proposed technique. This is due to the overlapped streams since the bandwidth is checked only for un-overlapped parts. Oppositely, without using data structuration, the bandwidth is checked for the entirely requested stream. It is to be noted that the total number of activated queries at an instant t is higher with using data structuration than without using data structuration. This is because of the presence of overlapped parts in the buffer.

6.2.2 Number of supported queries

In this series of simulation, our aim is to test the system performance in the presence of “peak accesses” i.e., some streams can become very requested. To do that, we varied the percentage of accessing the same object and observed the number of supported queries. The results are shown in figure 5 for one LOM.

Figure 5: Total number of supported queries in presence of “peak accesses”.

Clearly, for a low percentage of requests for the same object, the performance of the two approaches are nearly the same. However, with increasing the amounts of requests for the same object, the approach using data structuration outperforms the other approach. This is so because of *preserving* explicitly streams in the buffer. Noted that without using data structuration, the total number of supported queries is limited by the available bandwidth.

6.2.3 Buffer space needed

As we have seen, the approach using data structuration outperforms the other approach. However, this performance is due essentially to the ability to preserve data in the buffer, especially overlapped data. This means it is necessary to have a sufficient buffer space to do so. In this series of simulation, we aim to observe the use of the buffer space by the two approaches. The results are summarized in figure 6.

Figure 6: Total buffer space used

Obviously, the approach using data structuration needs more buffer space than the other approach. One interesting result for the latter approach is the limited number of activated queries in spite of buffer space availability. This is due to the limited bandwidth available. This result shows clearly that the number of activated queries is proportional to the bandwidth available. Thus, in presence of limited bandwidth, it will be beneficial to increase the buffer space and to use an appropriate approach to preserve temporary some streams in the buffer.

7 Conclusion and further work

In this paper, we described a logical model of a video server. One important key that guides the design process is the extensibility and the scalability of the system. In addition, the model proposed is independent of any physical configuration, so that it will be possible to port it on massively parallel machine as well as on a cluster of PCs/workstations. It is based on logical components called storage components which are connected by a high-speed interconnection network. All the storage components are *governed* by an Global Object Manager (GOM) that is responsible of scheduling all incoming requests to storage components. After describing the overall architecture of the proposed server, we introduce a novel approach for scheduling incoming requests. This approach is based on the use of the information available about the data structuration, especially the overlapping between streams to improve the server performance. The key idea behind this is to promote a media sharing between users requesting overlapped streams. The results of simulation shows clearly the effectiveness of the approach proposed.

Another aspect of our model, not addressed in this paper, is to use a virtual shared buffer. The motivation for using a virtual shared buffer comes from the fact that, as the results of simulation show, the total bandwidth is a real bottleneck. To cope with this limitation, some objects (more frequently accessed or having a high set of overlap) might be preserved in the buffer. However, as the volume of video streams is large, the fragmentation of the total buffer space between the LOMs seems to limit this operation. So, promoting a virtual shared buffer will resolve this problem. Future work will focus on the implementation of the server in order to test the proposed techniques at a real level. Now, we are in process of developping such server on a cluster of workstations/PCs interconnected by a Myrinet network. The choice of such platform is motivated by both satisfying performance a Myrinet network provides and economical considerations.

Acknowledgments

I would like to thank Lionel Brunie for his helpful comments and Ulrike Wilhelm to help me to prepare this paper.

References

- [1] D. Anderson, Y. Osawa, and R. Govindan. A file system for continuous media. *Acm Transactions on Computer Systems*, 10(4), 1992.
- [2] R. Chillarege, S. Biyani, and J. Rosenthal. Measurement of failure rate in commercial software. *IBM Research report RC-19889*, Dec. 1994.
- [3] C. Federighi and L. A. Rowe. A distributed hierarchical storage manager for a video-on-demand system. In *Storage and Retrieval for Imaging and Video Databases II. Symp. on Elec. Imaging Sci.*, Feb. San Jose, CA, 1994.
- [4] S. Ghandeharizadeh and L. Ramos. Continuous retrieval of multimedia data using parallelism. *IEEE Transaction on Knowledge and Data Engineering*, 5(4):658–669, 1993.
- [5] R. Hjelsvold and R. Midtstraum. Modeling and querying video data. In *Proceedings of the 20th VLDB Conference*, Santiago, Chile, 1994.
- [6] R. Hjelsvold, R. Midtstraum, and O. Sandsta. Searching and browsing a shared video database. In *Proceedings of the International Workshop on Multimedia Database Management Systems*, N.Y 1995.
- [7] D. Jadav, C. Srinilta, A. Choudhary, and P. Berra. Design and evaluation of data access strategies in a high performance multimedia-on-demand server. In *Second International Conference on Multimedia Computing and Systems*, May 1995.
- [8] D. Jadav, C. Srinilta, A. Choudhary, and P. Berra. Techniques for scheduling i/o in a high performance multimedia-on-demand server. *Journal of Parallel and Distributed Computing*, 30:190–203, 1995.
- [9] S. Lau, J. Lui, and P. Wong. A cost-effective near-line storage server for multimedia system. In *Proceedings of the 11th International Conference on Data Engineering*, pages 449–456, Mar. 1995. IEEE.
- [10] W. E. Mackay and G. Davenport. Virtual video editing in interactive multimedia applications. *Commun. ACM*, 32(7):802–810, July.
- [11] Myricom. www.myricom.com.
- [12] B. Ozden, A. Biliris, R. Rastogi, and A. Silberschatz. A disk-based storage architecture for movies-on-demand servers. *Information Systems*, 20(6):465–482, 1995.
- [13] T. Rakow, E. Neuhold, and M. Lohr. Multimedia database systems :the notions and the issues. In *Tagungsband GI-Fachtagung Datenbanksysteme in Buro, Technik und Wissenschaft (BTW)*, Berlin, 1995.
- [14] P. Rangan, H. Vin, and S. Ramanathan. Designing an on-demand multimedia service. *IEEE Communications Magazine*, July 1992.
- [15] A. Reddy and J. Wyllie. I/o issues in a multimedia system. *IEEE Computer*, pages 69–74, Mar. 1994.
- [16] O. Sandsta, S. Langorgen, and R. Midtstraum. Design and implementation of the elvira video server. In *Norsk Informatik konference*, Nov. 1996.
- [17] R. Tewari, D. M. Dias, R. Mukherjee, and H. M. Vin. High availability in clustered multimedia servers. In *ICDE New Orleans*, Feb. 1996.