

Data Allocation and Spatio-Temporal Implications for Video-On-Demand Systems

Kingsley C. Nwosu
AT&T Bell Labs.,
67 Whippany Rd.,
Whippany, NJ 07981 U.S.A.

Patrick Bobbie
Computer Science Dept.,
Florida A & M Univ.,
Tallahassee, FL 32307 U.S.A.

Bhavani Thuraisingham,
The MITRE Corp.,
Burlington Rd.,
Bedford, MA 01730 U.S.A.

Abstract

As the number of video streams to be supported by a Digital Video Delivery System (DVDS) increases, we more and more start to understand that the ability to reliably and cost-efficiently support a considerable number of video streams (in the magnitude of tens of thousands) largely depends on software capabilities. Even in the presence of the best hardware configuration (not ignoring software vs hardware costs), the software exploitation of the hardware capabilities is of paramount importance. It is imperative that current software developments account for the eventual scalability of the number of video streams without commensurate increase in hardware. In this paper, we present strategies for the management of video streams in order to maintain and satisfy their space and time requirements. We present a detailed analysis of the issues related to queuing I/O requests and data buffering. The designs for the arrangement and scheduling of I/O requests and data buffers are presented with the objective of guaranteeing the required data availability rates for continuous media display.

Keywords: buffer queues, buffer scheduling, data allocation, data striping, stream management, video delivery.

1 Introduction

As the realization of Digital Video Delivery Systems (DVDS) progresses, it becomes clearer that the major challenges and obstacles to supporting the magnitude of video streams for video customers lies mainly in intelligent and sophisticated exploitation of the hardware technologies. Functions and operations in conventional Operating Systems and File Systems that pertain to process scheduling, data management, and I/O's, do not, inherently, address the requirements of video streams. The continuous nature of video data, with respect to displays, necessitate that a DVDS be able to maintain minimum required throughput per video or audio stream. Although there may be high bandwidth and fast networks, and high speed CPU's, the relative speed disparity in storage device speed and the number of streams needed, may adversely offset those hardware advantages. It is necessary

to remember that one of the perturbing characteristics of video data is its enormous size. A result of that is the amount of data that is consumed per unit time during video display. Its continuous nature requires that there should not be a data lapse with respect to data availability during display. In addition to the fact that video data are inherently large, even after compression, the quality of video affects its size. The better the quality, the larger the size.

Several architectural configurations for efficient and reliable storage and delivery of video data have been proposed and utilized [1]-[8]. Some of these works address only the configurability of the system components while others address only the support of small number of video streams, in the magnitude of less than 10,000. Although it is arguable that the DVDSs are relatively in their infancy, it is necessary that current software developments account for the scalability of the number of video streams without commensurate increase in hardware. As the number of video streams increases, the ability to support them increasingly depends on the software capabilities.

In this paper, we describe a protocol for the management of video streams in order to maintain and satisfy their space and time requirements. We use a DVDS architectural model based on partitioned nodes where one node partition is responsible for data retrieval while the other accepts user requests, determines object locations, and routes requests through the network that connects both partitions. We present a detailed analysis of the issues related to queuing I/O requests and data buffering. The designs for the arrangement and scheduling of I/O requests and data buffers are presented whose objective is to guarantee the required data availability rates for continuous media display.

2 DVDS Architectural Model

In this section, we briefly describe our working DVDS model. Although there are myriad of architectures and configurations for DVDSs, we use, without loss of generality, Figure 1 as the working model. We only address the streams management problems with respect to the

DVDS server nodes. The figure assumes a partitioned DVDS server that consists of Storage Nodes (Snodes) and Network Nodes (Nnodes). The Nnodes are the nodes that connect to either public access networks or private networks for the customer sites. They accept customer requests, determine the location of a target object, and route the requests through the network. The network is a non-blocking interconnection network that permits any Nnode access to any storage device attached to any of the Snodes. The Snodes contain the storage devices where the video data are stored. Their primary function is data retrieval.

At any given time, a particular Nnode may be servicing any number of customers (streams) and a particular Snode may also be servicing any number of I/O streams.

Since each Snode may have several streams to service at any given time, and due to the I/O latencies, it must have a means of storing pending requests and data that

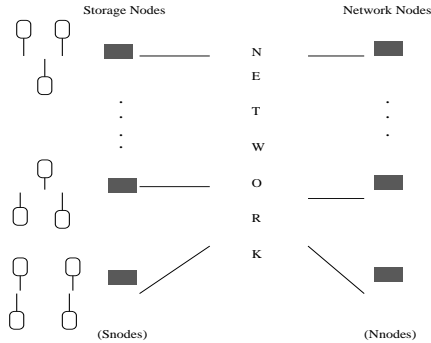


Figure 1: The working architectural model of a DVDS.

are ready for transmission to the Nnodes. Consequently, each Snode employs a queue for pending I/O requests and buffers for pending data transmissions. The arrangement and scheduling of the buffered data and requests must be done with the objective of satisfying each stream's data availability. These requirements also apply to the Nnodes when servicing individual customer or stream.

As has been numerously discussed, e.g., in [9]-[11], data striping is a *sine qua non* for any meaningful data allocation strategy for continuous media or large objects. Therefore, we assume that a given video object is decomposed into a number of segments whose size is defined by a system-wide Allocation Unit (AU). A set of consecutive AUs of an object are allocated to different storage devices or Snodes to foster parallel accesses. Multiple AUs of a single video object may be stored in a single storage device or Snode. The contiguity or de-clustering of related AUs within a single target is implementation dependent. Assuming that two video objects, O^1 and O^2 , were allocated sequentially (i.e., O^1 before O^2), then Figure 2 shows an example of a striped allocation between two Snodes, $Snode_1$ and $Snode_2$, where $Snode_1$ has two

storage devices and $Snode_2$ has one storage device. AU_j^i is the j th AU of O^i and $AU_{j,k}^i$ is the k th stripe unit (unit of allocation per storage device when an AU is striped) of AU_j^i . Obviously, the maximum parallelism on the Snode level is two, i.e., we can read two consecutive AUs of an object in parallel. Also, several uniformly-defined combinations of different AUs of different objects can be retrieved in parallel. We assume that the contiguous or de-clustering allocation of AUs are done at optimal locations with respect to seek and rotational latencies of the storage devices.

Another important property of the data allocation is that the AUs of a video object may be replicated across different storage devices or Snodes. For the purposes of data availability and system reliability, it is imperative that objects be replicated to foster failure independent locations. That is, in the presence of a single-server Snode with multiple storage devices, the storage devices for a video object's AUs must be mutually exclusive. On the other hand, when multiple Snodes are utilized, copies of a video object's AUs must also be allocated on mutually exclusive Snodes. Data replication allows for load balancing among storage devices and Snodes. I/O requests that may have been directed to a single copy can be directed to a different, less utilized device. Furthermore, when either a storage device or Snode becomes unavailable, requests for the data stored in it are directed to the alternates.

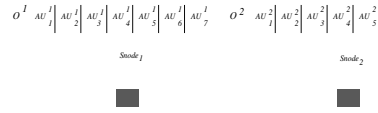


Figure 2: An example of a striped allocation.

3 Problem Formulation

Using the requirements discussed in the preceding section, we formulate the problem thus: Given that

- each video stream requires a given expected data availability rate,
- each Snode maintains a queue for pending requests and buffers for pending data transmissions,
- there is a limit on the number of each Snode's buffers,

- there is a quantifiable time delay for retrieving each AU from storage,
- there is a quantifiable time delay to service a buffer,
- there may be several copies of an AU across an Snode or several Snodes.

Therefore,

- how do we arrange, prioritize, and schedule the I/O requests and data transmissions to guarantee each stream's data availability?

4 Problem Analyses

We now present, in more detail, the implications of the problem formulation as they pertain to the expected objective. The data availability rate of a video stream is a function of its data consumption rate (dcr). The dcr of a stream is the amount of data that is used at the display device per unit time. It depends on the image quality and display speed. For example, a fast-forwarding of a video stream consumes more data per unit time than its slow-motion. We have already mentioned that the encoding rate of a video object determines its quality. Therefore, during fast-forwarding of a video stream, the delivery protocol must adjust its data transfer rate or scheduling frequency to satisfy the new data availability rate.

4.1 Request Queues and Data Buffering

Each Snode maintains a queue for pending requests and buffers for pending data transmissions. If an Snode were to service only one video stream, then it may be able to move data directly from the permanent storage to the network without an intermediate buffering. However, that is the best case scenario. In most cases, at any given time, an Snode may be responsible for any number of I/O requests for several video streams whose immediate data requirements are contained in the Snode's storage devices. Due to the fact that the storage devices cannot satisfy all the arriving requests due to its slower speed, and the fact that the network has to schedule its transfer during each cycle, it becomes impossible for every I/O request for numerous concurrent video streams to be satisfied on arrival. Consequently, some of the requests may have to be queued before they are satisfied and, after extraction from the storage devices, may have to be buffered before being transferred to the applicable Nnode.

An important requirement of data buffering is the total amount of data that may be buffered. Buffers are limited resources in any computing system, and may become very expensive. In determining the amount of data that may be buffered, the maximum number of video streams that can be targeted to a single Snode plays a paramount role. We have denoted this value, in Section 2, as Π . However,

that is a theoretical limit. It can only occur, for example, when every end-user requests for different parts of different movies such that every requested data resides in one Snode. Its hard to conceive of a situation when accesses to a single video object may create this limit. However, the limit may occur, but the probability is close to zero. The fact that it may occur may necessitate using Π as a determinant for the amount of buffers reserved in each Snode. This may be practical when Π and buffer size are small. On the other hand, a fraction of Π may be used, and the routing of requests to any Snode depends on buffer availability. That may be catastrophic, if the video stream is active and there does not exist an alternate site for the request. If δ is the degree of replication of each video object and there are δp^{snodes} Snodes such that all the single copies of all the stored objects can be accommodated in p^{snodes} Snodes, then the best estimate is to use a normalized value, $\Pi^{normalized} = \frac{\Pi}{\delta}$, as the expected maximum number of streams per Snode. It will take a near concurrent failure of $[(\delta - 1)p^{snodes}]$ Snodes for the expected maximum number of streams per node to become Π .

A very good way of controlling the request traffic is for each Nnode to have an intelligent way of knowing when two or more streams require, at the same time, the same blocks of data of a video object. In that case, only one request will go over the network.

4.2 Data Retrieval Delays

Irrespective of the physical layout of data in permanent storage, one always incurs data retrieval overhead (t_r) which is the sum of the data access (t_a) and block transfer (t_t) times. For certain types of devices, there is a rotational latency for the initial word of the desired block to come into position and additional seek latency to move the arms into the proper track position. In reality, one can eliminate or significantly minimize the rotational and seek latencies with intelligent contiguous allocations of objects. But for all practical purposes, it is more reasonable to account for these latencies because of the worst case situations. Given that we have consecutive I/O requests to a given storage device, for related or unrelated AUs, we need to account for the worst case delay in satisfying both requests. That can easily be determined from the maximum expected elapsed time from the time of completing one request and starting and completing the next. Therefore, using the t_r 's, we can dynamically determine when it is propitious to schedule any particular request to maintain its data availability rate.

4.3 Data Migration Overhead

Usually, data buffers have to be moved from their point of resident (in Snodes) to the network adapters for transmission over the network. This movement takes some

quantifiable time delay. If it takes t_m time to prepare a buffer for network transfer, then given that a given buffer is preceded by n buffers implies that nt_m time must elapse before it is ready to be transferred. If this waiting time will adversely affect the data availability rate of the object, then n becomes unacceptable. Therefore, we can use t_m to determine the acceptable number of buffer network transfer schedules that may precede a given buffer, whenever applicable.

5 Stream Display Factors

In order to present design solutions to the problems associated with the streams' management, it is important to present the pertinent display properties and characteristics. We use the following variables:

data consumption rate (dcr): The amount of data that is consumed by a display device. It is related to a video object's data availability rate.

minimum utilization value (muv)

The expected minimum amount of data that must be available during each display session. It is expressed as:

$$muv = dcr \times n \quad (1)$$

where $n > 1$. The value n is used to create a read-ahead strategy or *time comfort zone*. It implies that we try to maintain n time units of available consumable data in case of unforeseen circumstances in I/O delays or system mishap. The objective is to always keep the muv maintained at each Nnode per stream.

startup time ($stime$) The amount of time required for the muv of an object to become available. At the initiation of a video stream, it takes $stime$ time before data are actually sent to the display device. Actually,

$$stime = \frac{muv}{dcr}. \quad (2)$$

buffer utilization rate (bur): The minimum number of buffers required by a stream control process at an Nnode to maintain muv . If we denote the size of a buffer as $bufsize$, the maximum fraction of a system's buffers that a single process can utilize as r , and the number of buffers in a node as $bufnum$, then

$$bur = \lceil \frac{muv}{bufsize} \rceil \quad (3)$$

This is acceptable only if

$$bur \leq (bufnum \times r).$$

6 Design and Management of Queues

Having presented the analyses of the problems and issues in satisfying a video stream's expected data availability, we now discuss the designs and mechanisms to address the problems. First, we discuss the arrangement and scheduling of the I/O requests and data buffers. Obviously, from the preceding discussions, given a number of requests with uniform dcr (which implies uniform muv and $stime$), I/O requests can be serviced using First-In-First-Out (FIFO) queue. Furthermore, in order to maintain muv , it is imperative that the time to service $\Pi^{normalized}$ requests per node be at most $stime$. However, we have ascertained that in spite of uniform encoding for some given video objects (e.g., MPEG¹-I and MPEG-II), the dcr may dynamically change as a result of either fast-forward or slow-motion display of a video object. Obviously, a set of video streams that are being displayed with a certain slow-motion rate has the same dcr . It, therefore, becomes necessary that streams with similar dcr has to be grouped together. Given that there are n^{dcr} different dcr rates, Figure 3 shows an example design for the request and data buffer queues. It shows a Request Hash Table (RHT) and Buffer Hash Table (BHT) where each dcr request queue is represented by R^i , $1 \leq i \leq n^{dcr}$, and each dcr buffer queue by B^j , $1 \leq j \leq n^{dcr}$. r_k^i is the k th request in R^i , and b_l^j is the l th buffer in B^j . We assume that the dcr of R^i is greater than the dcr of R^m , if $i > m$.

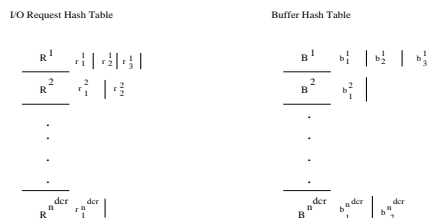


Figure 3: Arrangement of request and buffer queues.

6.1 Request and Buffer Queues Scheduling

The most important thing in the I/O process is the scheduling priorities of the requests and buffers which greatly affect the satisfiability of each stream's data availability rate. It is necessary to guarantee that the time for servicing each I/O request (retrieval and transmission times) for a given stream does not exceed or be very close to the stream's $stime$. If that happens, then it means that all the data, at the display site, may have been consumed before requested data becomes available,

¹Motion Picture Experts Group

thereby, violating the stream's *muv*. Since the two resources, I/O requests and data buffers, are independent of each other, they are serviced by different processes. A retrieval process (P^{ret}) services a pending request from the queue and appends the data read in the applicable buffer queue. A different process (P^{sched}) schedules the buffers for network transfer. Each R^i or B^i is controlled by a process, P_i^{ret} , P_i^{sched} , respectively, that schedules one of its I/O requests or data buffers for P^{ret} or P^{sched} , respectively. Each P_i^{ret} continuously monitors all the requests in R^i with respect to the request's arrival time and current time. Periodically, each P_i^{ret} obtains from P^{sched} the average waiting time, t_{wait}^i , for a buffer in B^i and the average waiting time, t_{wait}^{ret} , of a request in P^{ret} 's queue. Consequently, a P_i^{ret} makes a decision based on the t_r^i (the t_r of R^i), t_m^i (the t_m of B^i), and t_{wait}^i , about when to schedule its oldest request. The goal is that, if τ is the expected maximum elapsed time for the data arrival at the Nnode, t^{init} the time the request was initiated, t^{cur} the current time, then

$$[(t^{cur} - t^{init}) + t_{wait}^i + t_{wait}^{ret} + t_r^i + t_m^i + t^{network}] < \tau \quad (4)$$

where $t^{network}$ is the anticipated network transfer time. Given the expected waiting time for an I/O, we introduce a time zone at which the request must be dispatched when in a queue. This time zone is created with ϵ time units. When an I/O request has spent a total time which is within ϵ time units to its expected waiting time, then it must be dispatched to the next queue. We use Figure 4 to illustrate the transitions of a single I/O video data request. The time scales do not imply any time-magnitude relationships between the different events.

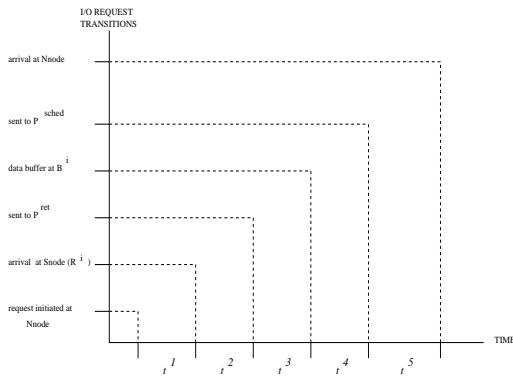


Figure 4: Transitions of an I/O request.

The following apply to Figure 4:

- t^1 = time for the request to get from source node (Nnode) to the target Snode,
- t^2 = waiting time in R^i 's queue,
- t^3 = waiting and data retrieval time in P^{ret} 's

- queue, (i.e., $t_{wait}^{ret} + t_r^i$),
- t^4 = waiting time in B^i 's queue, (i.e., t_{wait}^i),
- t^5 = time to transfer data over the network, (i.e., $t^{network}$).

We use the algorithmic pseudo-code of **Algorithm I** to describe the activities of the processes. When the service queue of either the P^{ret} or P^{sched} becomes empty, the process sends a message (SENDMSG) to all the P_i^{ret} 's or P_j^{sched} 's such that any of those processes can dispatch an I/O request or data buffer when the respective queue is not empty. A process checks its message queue via RECVMSG. By sending the messages when the queues are empty, we minimize the idle times of the processes. We use the function *COMP1* to compute the left-hand side of equation (1) and *COMP2* to compute the total time already spent by a request since its initiation. We use R_{head}^i or B_{head}^i to denote the first I/O request or data buffer, respectively, of a FIFO queue.

Algorithm I: Algorithmic pseudo-code for satisfying I/O requests.

```

begin
  Forall ( $P^{sched}$ ,  $P^{ret}$ ,  $P_i^{sched}$ ,  $P_j^{ret}$ ) do in parallel
     $P^{ret}$ : begin
      Forever do
        If queue is not empty do
          Extract a request
          Schedule the request for I/O (asynchronously)
          Put data in proper buffer queue
        endif
      else
        SENDMSG to all  $P_i^{ret}$ 's
      endelse
    endforever
  endbegin
   $P^{sched}$ : begin
    Forever do
      If queue is not empty do
        Extract a buffer
        Send buffer to network (asynchronously)
      endif
    else
      SENDMSG to all  $P_i^{sched}$ 's
    endelse
  endforever
endbegin
 $P_i^{ret}$ : begin
  Forever do
     $T = \text{COMP1}(R_{head}^i)$ 
    If  $[(T + \epsilon) > (\tau - \epsilon)]$  or RECVMSG do
      Extract  $R_{head}^i$ 
      Schedule on  $P^{ret}$ 
    endif
  endforever
endbegin
 $P_i^{sched}$ : begin
  Forever do
     $T = \text{COMP2}(R_{head}^i)$ 
    If  $[(T \geq (\tau - \epsilon))$  or RECVMSG] do
      Extract  $B_{head}^i$ 
      Schedule on  $P^{sched}$ 
    endif
  endforever
endbegin

```

```

endbegin
endforall
endbegin

```

7 Simulation Model

In order to verify the temporal satisfiability of video streams data requirements, we simulated the control processes described for streams management. The architecture of the processes and their inter process communications are shown in Figure 5. It consists of a main process *Initp* which creates a number of child processes (*Nnodep*) that represent the Nnodes. Each *Nnodep*, periodically, generates a request that stipulates the video file to display. The time to generate a display request and the object to display are randomly determined. Upon generating a display request, a *Nnodep* forks off a child process (*Pcont* that is responsible for the data retrieval. Each *Pcont* controls one video stream. *Initp* also creates the *Pret* and *Psched* processes that run forever. It also creates the P_i^{ret} and P_i^{sched} processes depending on the number of different *dcr* rates. Each *Pcont* communicates with any of the P_i^{ret} via a named pipe ($pipe_i^{ret}$). Each pipe represents an R^i . A *Pcont* writes information about a given I/O request to the applicable R^i where it is read by P_i^{ret} . Each P_i^{ret} similarly communicates with *Pret* through a named pipe ($pipe_{ret}$). After retrieving some given data, *Pret* sends it to the applicable P_i^{sched} via a named pipe ($pipe_i^{sched}$). The named pipe, $pipe_{sched}$, is used by the P_i^{sched} 's to schedule data transfers for *Psched*. As is evident from Figure 5, more than one *Pcont* may want to write to the pipe at any given time. Consequently, exclusive write access must be obtained by the process before writing the request. Each *Pcont* is responsible for satisfying each streams *stime* and *muv*. Mapping this simulation model to the designs and management strategies described in Section 6, the *Pret*, *Psched*, P_i^{ret} , and P_i^{sched} processes run in the Snodes while the *Pcont* processes run in the Nnodes.

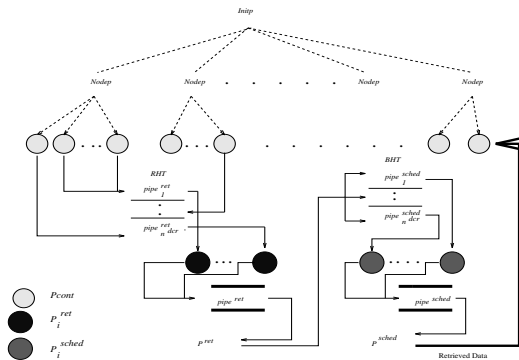


Figure 5: Simulation architecture.

7.1 Simulation Results

The simulation effort was primarily to determine the satisfiability of the temporal requirements of the data availabilities of a number of simulated video streams. Using the model described, we were able to generate a number of *Pcont* processes that were controlling the display of a number of large files. We use the following observed (real) time:

- average waiting time for a buffer in B^i ,
- average waiting time for a request in R^i ,
- time for actual retrieval of data from physical storage.

The network transfer time is related to the amount of data being transferred. We started with 20 *Nnodep*'s where each of the processes was able to generate a maximum of 200 *Pcont*'s. Each *Pcont* was generated after some specified time intervals. The original time interval was at least as big as the maximum *stime*. We need to understand the impacts on data availability of the streams by the variations in magnitude of some of the variables. Specifically, we did one of the following:

1. increased the number of *Pcont*'s per Nnode,
2. increased or decreased the inter-*Pcont* generation time, or
3. increased the degree of replication.

Given a number of *Nnodep*'s, as the number of *Pcont*'s increased substantially, the average waiting times of the requests and buffers increased until newly generated *Pcont*'s became inadmissible due to the fact that equation (1) became unsatisfiable (i.e., the left hand side exceeded the right hand side). Under realistic environments, at this point, it becomes necessary to increase the number of Snodes. Using the original inter-*Pcont* generation time, it took more number of *Pcont*'s than when the inter-*Pcont* was reduced to almost zero, before some requests were rejected. When a large number of requests are concurrently initiated, then all those requests will invariably need their requests satisfied concurrently, thereby, creating longer queues that affect succeeding requests. Figure 6 shows an example behavior with regards to increasing the number of *Pcont*'s per node and increasing the degree of replication (*dr*). It shows that both factors have profound impact on the average waiting times of each I/O request.

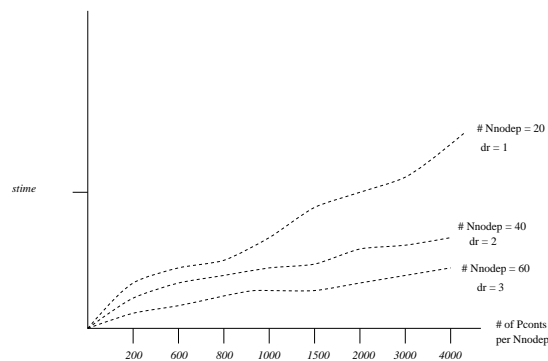


Figure 6: Impacts of increasing $Pcont$'s and/or degree of replication

8 Conclusion

In an effort to address the problems of video streams management in Video-On-Demand Systems, much of the attention has been focused on the capabilities of the hardware. Although a number of DVDSs have become operational, their ability to deliver the required services mainly depend on the small number of streams supported and the hardware trade-offs. As the number of streams to be supported increases, it becomes obvious that the cost of the hardware becomes exorbitant and unaffordable. The provision of a cost-effective DVDS depends on intelligent and sophisticated exploitation of the hardware technologies. An obvious way of achieving that is through efficient streams' management. In this paper, we present strategies for the management of video streams in order to maintain and satisfy their space and time requirements. We present a detailed analysis of the issues related to queuing I/O requests and data buffering. The designs for the arrangement and scheduling of I/O requests and data buffers are presented with the objective of guaranteeing the required data availability rates for continuous media display. An experimental model was implemented and its results presented to show the effects of some of the streams' management variables as they increase or decrease. This paper, in addition to analyzing the important issues for streams' management, presents simple (in terms of complexity) strategies to handle the high-lighted problems.

References

- [1] **K. Nishimura, T. Mori, Y. Ishibashi, N. Sakurai**, System Architecture for Digital Video-On-Demand Services, pp. 602-606.
- [2] **A.D. Gelman, H. Kobrinski, L.S. Smoot, S.B. Weinstein, M. Fortier, D. Lemay**, A store and forward architecture for video on demand service, *Proc. International Conference on Communications*, Vol. 2, June 1991, pp. 842-846.
- [3] **R. Ramarao, V. Ramamoorthy**, Architectural design of on demand video delivery systems: the spatio temporal storage allocation problem, *Proc. ICC'91*, Vol. 1, Denver, CO., 1991, pp. 506-510.
- [4] **Hodge, W., Mabon, S., Powers, J.T., Jr**, Video on demand: architecture, systems, and applications, *SMPTE Journal*, Vol. 102, No. 9, Sept. 1993, pp. 791-803.
- [5] **S L. Hardt Kornacki, L. A. Ness**, Optimization model for the delivery of interactive multimedia documents, *Proc. IEEE GLOBECOM'91*, Vol. 1, pp. 669-673.
- [6] **D. Maier, J. Walpole, R. Staehli**, Storage architectures for continuous media data, *Proc. 4th International Conf. on Foundations of Data Organizations and Algorithms*, Chicago, IL. Oct. 13-15, 1993, pp. 1-18.
- [7] **P. V. Rangan**, Video conferencing, file storage, and management in multimedia computing systems, *Computer Network. ISDN Syst.*, Vol. 25, No. 8, March 1993, pp. 901-909.
- [8] **G. Blair, D. Hutchison, D. Shepherd**, Distributed systems support for heterogeneous, multimedia environments, *Proc. First Intern. Workshop on Network and Operating System Support for Audio and Video*, Berkeley, CA., Nov. 8-9, 1990, pp. 7.
- [9] **C. Y. Roger Chen, Kingsley C. Nwosu, P. Bruce Berra**, Modeling and Storage Allocation Strategies for Heterogeneous Parallel Access Storage Devices in Real Time Multimedia Information Processing, *Proc. IEEE 17th Annual International Computer Software and Applications Conference (COMPSAC)*, Phoenix, Arizona, November 1-5, 1993, pp. 216-223.
- [10] **P. Lougher, D. Shepherd**, The design and implementation of a continuous media storage server, *Proc. Third Intern. Workshop on Network and Operating System Support for Audio and Video*, La Jolla, CA., Nov. 12-13, 1992, pp. 69-80.
- [11] **K. C. Nwosu**, Data Storage Modeling and Management for Multimedia Information Systems, *Ph.D. Dissertation*, Syracuse University, Syracuse, NY, Dec. 1993, pp. 1-166.