

Spatio-Temporal Effects of Multimedia Objects Storage and Delivery for Video-On-Demand Systems

Cyril Orji

Dept of Comp. Science,
Florida International Univ.,
Miami, FL 33199.

Patrick Bobbie

Dept. of CIS,
Florida A&M Univ.,
Tallahassee, FL. 32307.

Kingsley C. Nwosu

AT&T Bell Labs.,
67 Whippany Rd.,
Whippany, NJ 07981-0903.

Abstract

As the number of video streams to be supported by a Digital Video Delivery System (DVDS) increases, an improved understanding of the necessity for reliable and cost-efficient support for a considerable number of video streams (in the magnitude of tens of thousands), and the dependency largely on software capabilities, emerges. Even in the presence of an optimal hardware configuration, or model, and associated costs, using software to exploit the underlying hardware capabilities is of paramount importance. Although a number of DVDSs have become operational, their ability to deliver the required services mainly depends on the small number of streams supported and the hardware trade-offs. 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 use a DVDS architectural model with functionally dichotomized nodes: a single node partition is responsible for data retrieval, while the remaining partition of nodes 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 discussion includes the requirements for arranging and scheduling I/O requests and data buffers, 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

The recent advances in the manufacturing and use of Digital Video Delivery Systems (DVDS), indicate that intelligent and sophisticated techniques for exploiting the capabilities of hardware technologies, would ameliorate the major challenges and obstacles in the delivery of large video stream. Hardware and software modules of conventional operating systems and file management systems for process scheduling, data management, and I/O processing, unfortunately, do not adequately 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 disparity in the speed of storage devices and the number of streams needed, may adversely offset the hardware benefits. One characteristic of video data that impacts the number of streams, and bandwidth requirement, is the enormous data size. The size directly determines the amount, or volume, of data to be consumed per unit time during video display. A continuous display requires a very minimal or no data lapse. Thus, a high data availability rate is desirable for continuous 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. Consider the fact that uncompressed MPEG-2¹ encoded video requires a data rate of about 3.5 Mb/s; uncompressed NTSC-quality video requires about 216 Mb/s, and uncompressed HDTV-quality video requires about 648 Mb/s. Remember that current magnetic disks operate at a rate of about 1 MB/s. Further, consider also the fact that a single frame of colored video with 620- × 56- pixel frame at 24 bits per pixel requires about 1 MB storage space. However, with digital encoding techniques of 1.5 Mb/s, 3 Mb/s and 6 Mb/s rates, image qualities that are near VHS-VCR, NTSC Entertainment video, and NTSC Studio video, respectively, can be achieved

Several architectural configurations for efficient and reliable storage and delivery of video data have been proposed and utilized [1, 2, 4, 8, 11, 16, 19, 22, 9]. Some of these works address only the configurability of the system components while others address only the support of small numbers 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 development efforts account for the scalability of the number of video streams without commensurate increase in hardware requirement. Therefore, as the number of video streams increases, the ability to support the streams increasingly becomes dependent 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 partition 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 design requirements for arranging and scheduling I/O requests and

¹Motion Pictures Expert Group

data buffers are presented. The objective is to derive formal models which guarantee sound data availability rates for continuous media display.

2 DVDS Architectural Model

In this section, we briefly describe the DVDS model. There are several architectures and configurations for digital data transmissions and delivery systems that have proposed or are being developed. Figure 1 shows a basic architecture of a multimedia storage and delivery system. In general, a multimedia delivery system comprises several levels of transmission nodes. Data is transmitted from large content servers via high speed networks to smaller neighborhood servers via Local Area Networks (LANs) and Metropolitan Area Networks (MANs). Each storage and delivery node in Figure 1 must meet some spatio-temporal requirements for optimal delivery of information to the end-users. To that end, therefore, there are several architectures for the storage and delivery systems. We use the model depicted in Figure 2 to illustrate and present our spatio-temporal modeling techniques. Other proposed configurations for DVDS modeling and space-time analysis may not necessarily correspond to ours, depending on the level of detail, parameters of interest, and partitioning scheme. However, there are attributes which are common to several DVDS models, such as display factors (queues and buffers) and network issues (delays and bandwidth) to which our models could be extrapolated.

Figure 1: Basic architecture of a Multimedia Delivery System.

We primarily address the streams management problems with respect to the DVDS server nodes. First, we briefly describe the specific server architecture on which this work is based. The server architecture, as shown in Figure 2, is 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 to the customer sites. They accept customer requests, determine the location of a target object, and route the requests through the network. A given customer is not dedicated to any Nnode. A customer's request is routed dynamically based on the load and status of the Nnodes. The network is a non-blocking inter-connection network that permits any Nnode access to any storage device attached to any of the Snodes. In Figure 2, we have used two source points to illustrate the connectivities to destination points. The full accessibility applies to all the Nnode connections. The Snodes contain the storage devices where the video data are stored. The primary function of the Snodes is data retrieval; which we assume is supported by low-level utilities of the underlying hardware and software.

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. Suppose η is the total number of customers in a DVDS where each customer may have up to κ number of concurrent streams. The practical maximum number of requests or streams that any of the nodes may be responsible for at any given time is limited by $\Pi = \eta\kappa$. Therefore, each of the nodes, theoretically, must be able to support Π streams at any given time. In Section 4, we discuss the pragmatics of this limit.

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 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 and continuity. These requirements also apply to the Nnodes when servicing individual customer or stream.

Figure 2: The working architectural model of a DVDS.

As we stated earlier, there are several other possible configurations besides the one proposed here. For an example, it is possible to build a multimedia delivery system such as a DVDS with only the Snodes or Nnodes where either the Snodes perform both the Snodes' and Nnodes' functionalities or the Nnodes perform both the Nnodes' and Snodes' functionalities. In either case, the nodes are responsible for the storage and retrieval of data as well as interfacing with delivery networks and routing customers' requests. In such a configuration, as shown in Figure 3, the nodes connect directly to the access network and also connect to another network for inter-node communication. When a request comes through $node_3$ for data stored in $node_2$'s storage device, it must be routed to $node_2$ via the inter-node network. Similarly, $node_3$ may be servicing other nodes' requests. Consequently, the nodes are not only retrieving data, but also sending requests to other nodes and waiting for responses from other nodes. As we will see later, taking cognizance of other things such as file system implications, request sending and queuing; this level of functional complexity may be detrimental to achieving the data availability rate for the target system.

The architecture and system analysis presented here are not targeted towards small scale systems for supporting tens of streams which one may achieve with a single server node. This work is targeted towards large scale DVDS systems with substantial investment to support hundreds of thousands of streams.

The architecture we have chosen provides modular responsibilities for the nodes. The Nnodes are responsible for routing requests to the Snodes and receiving retrieved data from the Snodes. With this clearly defined responsibility, its easier and more efficient for the Nnodes to manage the scheduling of retrieval requests for each stream in order to meet the stream's data availability rate. Furthermore, the Snodes are solely responsible for retrieving requested data and sending them to the Nnodes. They just retrieve requested data by applying the available I/O scheduling algorithm to minimize I/O request waiting time. There are no Snode inter-node communications - an Snode retrieves requested data from its local storage device.

Figure 3: A comparative architecture of a DVDS.

2.1 Data Allocation Units and Data Striping

The Data (block) allocation unit is one of the critical parameters in data storage and retrieval systems. It plays a paramount role in file system designs. The data allocation unit represents the minimum amount of data that is read or written to the physical storage as a unit operation. Since the bandwidth of a system is the amount of data that is read/written per unit time, the data allocation unit plays a paramount role in determining the bandwidth of a system.

Data striping is the strategy whereby an object is decomposed into blocks of units of allocation and each block or a group of blocks is stored in different storage device(s) which have been configured for parallel retrievability. As a result, it is possible to retrieve a number of blocks of an object concurrently by retrieving from the storage devices in parallel. Data striping is important and necessary because storing a video object (or any object) on one storage device limits the degree of concurrent accesses to the object, thereby, adversely affecting the potential bandwidth of the system. Although one can replicate the object in multiple storage devices by allocating more storage space for the object, however, that is not a cost effective solution. To illustrate with a simple example, let x bytes be the size of a block and s bytes the amount of data that a storage device can retrieve per second (i.e., the bandwidth). If we have decomposed and stored an object in n storage devices, therefore, we can potentially achieve a bandwidth of $(x \times s \times n)$ for the n storage devices.

The concept of data striping is a consequence of *Disk arrays* [7, 20] which is utilized in RAID (Redundant Arrays of Independent Disks) technology. The RAID method employs storage devices that are configured for parallel access. RAID is more or less a hardware approach to high performance I/O systems. However, as a result of current advances in file system development, several high performance file systems now employ different kinds of data striping.

Therefore, as a result of the impact of data striping in developing high performance (bandwidth) storage and delivery systems, it has become a *sine qua non* for any meaningful data allocation and retrieval strategy for continuous media or large objects. In that respect, we assume that a video object is typically decomposed into a number of segments whose sizes are defined by a system-wide variable: Allocation Unit (AU). A set of consecutive AUs of an object are further 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 storage device is implementation dependent. Assuming that two video objects, O^1 and O^2 , were allocated sequentially (i.e., O^1 before O^2). Figure 4 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 to the k^{th} storage device when an AU is striped) of AU_j^i . Under this assumption, the maximum parallelism at the Snode level is two, i.e., we can read two consecutive AUs of an object's stream in parallel. Also, several uniformly-defined combinations of different AUs of different objects can be interleaved and retrieved in parallel. We assume that the contiguous or de-clustering allocation of AUs using striping techniques are done at

optimal locations with respect to seek and rotational latencies of the storage devices. The concept of Allocation Units employs two levels of logical decomposition, namely, the AU for object allocation per Snode, and the AU for object allocation per storage device in a Snode. The AU at each level is logically atomic, i.e., per Snode, an AU is an atomic entity that must be wholly allocated to the node; per storage device, an AU is an atomic entity that must be wholly allocated within the storage device.

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 improved fault-tolerance. That is, for a single-server Snode with multiple storage devices, it is beneficial for the storage devices of an object's AUs to be mutually exclusive. On the other hand, when multiple Snodes are utilized, copies of a video object's AUs can also be allocated to the Snode on mutually exclusive basis.

Data replication allows for performance improvement 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. Similarly, when either a storage device or Snode becomes unavailable, requests for the data stored in it are directed to other devices. Furthermore, the advantage of striping is the improved mix or alternatives in the allocation of AUs across multiple storage units. Striping also allows for parallel access and interleaving, a performance improvement method that strives to minimize I/O latencies.

Figure 4: An example of a striped allocation of video objects.

2.2 Data Allocation Issues

As stated before, one of the most important issues in distributed data storage and retrieval is the problem of having sufficient bandwidth to support customer requests; and which also guarantees data availability and transmission rates. A related problem is the distribution of streams of concurrent customer requests to different nodes. In [12], some detailed discussions and solutions are proposed for the relative speed disparity between the storage devices and other computing system components. Specifically, the speed of storage devices have not improved relative to the speeds of CPUs, networks, etc.; therefore, in many instances like multimedia applications, the speed of the storage device (also network transmission delay) rather than the speed of the CPU is the limiting factor. We analyze the storage and retrieval problem from single-server or multiple-server standpoint.

Figure 5: A single server VOD system

Assume that we are dealing with a single-server system that supports n concurrent video streams. Let each stream require a data rate of r Mb/s . Therefore, the data retrieval bandwidth is $r \times n$ Mb/s . In the presence of MPEG-1 data encoding rate (about 1.5 Mb/s) and hundreds of concurrent streams, it becomes obvious that using one storage device to achieve the bandwidth is hard. It, therefore, becomes necessary to utilize multiple storage devices configured with parallel access. Under this configuration, a single video object can be decomposed and stored in multiple storage devices. That alone may not solve the problem. The video objects have to be stored in such a way that the parallel accessibility can be exploited. This is usually accomplished by *intra-node striping* which represents the decomposition of an object at the storage device level. A full-realizable intra-node striping occurs when an object is decomposed into AUs that can be stored individually on different storage devices attached to a single machine such that the whole object can be retrieved in one I/O unit-time. Deductively, this implies that the size of the AUs at the Snode level is large enough to accommodate the entire object, thereby, rendering it allocatable to a single Snode. For example, using Figure 5, given an object of size s , and m storage devices, we can decompose it into m segments as shown in Figure 6. If we allocate the segments to mutually exclusive storage devices (SD_1, \dots, SD_m), then a concurrent access to all the storage devices for the object can retrieve it in one unit access. Of course, we assume that the size of each segment is not greater than the bandwidth of the target storage device.

Figure 6: A sample decomposition of a video object

One of the problems with intra-node striping is that the cumulative bandwidth of video streams may exceed the system bandwidth which will result in significant delays in satisfying the I/O requests. A good way of handling the problem is to utilize multiple nodes with *inter-node striping*. Inter-node striping represents the decomposition of an object at the Snode level. An object is inter-node striped when it is decomposed into a number of segments and each segment is allocated to storage devices that are attached to mutually exclusive Snodes. Using the example above and Figure 2, it implies that *segment i* is allocated by $Snode_k$ such that there does not exist *segment j* ($i \neq j$) that is also allocated by $Snode_k$. Within each Snode, each segment can undergo intra-node striping as discussed above. Consequently, the task of retrieving an entire object does not become the responsibility of one Snode. The number of storage devices configured for the Snodes is largely determined by the bandwidth of a given number of video streams. For example, depending on the size of each disk, 1000 MPEG-1 streams require 200 MB/s throughput.

2.3 Availability and Reliability Issues

Replication of data in a distributed system serves multiple purposes. First, from the user's point of view, multiple copies of data provide the opportunity for substantially increased availability. From the system point of view, some form of replication is more than convenient; it is absolutely essential for both availability and performance. In distributed computing environments, replication of data on different machines is more meaningful than on different media on the same machine. Different machine replication can be beneficial to performance since selecting an adjacent replica to service an access request results in shorter service time. The overriding requirement for replication is that different replicas of the same object or part of an object reside on failure-independent machines. In other words, the availability of one copy is not affected by the availability of other copies, but that each copy's availability is location dependent. Above all, it is imperative that object replication be transparent to the users.

Since data replication affects a system's performance, it invariably affects one of the customers' crucial requirements - response time. In video delivery systems, the primary objective for data replication is to minimize the possibility of over-utilization of a storage device or node. Consequently, there are two levels of replication, namely, Intra Node Replication (INRra), i.e., replication at the storage device level per node, and Inter Node Replication (INRer), i.e., replication at processing nodes level. Given the architecture in Figure 5, the only meaningful replication is INRra where different copies of a video file or segments of a video file are stored in different storage devices. Given a set of storage devices, we can maintain two forms of replication aggregations, namely, Storage Device Exclusivity (SDE) or Storage Device Inclusivity (SDI). A replication strategy in INRra uses SDE when there is a set of storage devices that contain only replicas of objects and another set contains the originals. A replication strategy in INRra uses SDI when every storage device can contain both primary and secondary copies of objects. For example, using Figure 5, if $\{SD_1, SD_2, \dots, SD_{\frac{m}{2}}\}$ are used for the primary copies of video objects and $\{SD_{\frac{m}{2}+1}, \dots, SD_m\}$ are used for

the secondary copies, then we maintain SDE technique.

The advantage of using SDE techniques is that when a given storage device becomes unavailable, the requests for objects stored in it are directed to the replicas. Furthermore, for SDE, when copies are identical, there exists the opportunity to perform parallel data retrieval to minimize response time. This is especially the case for heavily used (read-only) video objects. SDI's, like SNI's, have advantages similar in concept to maintaining hardware redundancy to foster fault-tolerance and dynamic reconfigurability (via reconfiguration map tables).

One may argue that this replication strategy is unorthodox and may lead to uncautionable waste of storage space. However, the fact remains that the advent of multimedia computing, with its concomitant novel storage and retrieval requirements, have rendered some of the traditional approaches inadequate and insufficient. Although the issue of the optimal utilization of devices is still a concern, however, other real-time, efficiency, and reliability issues take precedence. Primarily, the objective of SDE is to guarantee that no two similar AUs of an object are allocated to the same storage device during replication.

Similar replication concepts are applied to server configurations that consist of multiple server nodes where each node is dedicated to a set of storage devices. Two conditions are of paramount concern, namely,

- when an object has a potential for global access, i.e., access requests are expected from any of the clients with the network environment,
- when an object has a potential for only localized access, i.e., access requests are expected from only the local system.

In that case, objects are replicated for availability within the environment to minimize I/O waiting time by employing global or local replication strategy. Therefore, a Storage Node Exclusivity (SNE) is maintained when an object is replicated on different nodes and a Storage Node Inclusivity (SNI) is maintained when an object is replicated within a single node. If availability and reliability are of paramount concern, then SNI, in this case, is not recommended. A combination of SDE (at the storage device level) and SNE (at the nodes) level provide adequate availability and reliability for storage device or node failures. In the case of node failures, it is imperative that the system be able to dynamically re-distribute its requests.

3 The Parametric Constraints of I/O Scheduling

In this section, we briefly enumerate the parameters or constraints that characterize the problem formulation. The parameters are influenced by the assumptions made from the framework of video system architecture, and data storage and allocation schemes discussed in the preceding section. 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, and
- there may be several copies of an AU across an Snode or several Snodes.

Therefore, the central question is

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

4 Problem Analysis

Further analysis and derivation of formal models for the expected objective, and the constraining parameters, from the foregoing characterization are discussed in the sequel. 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, which depends on the image quality and display speed.

One of the operations during video display is fast-forwarding which allows a user to quickly scan a video. Under normal circumstances, this calls for a larger amount of data to be retrieved for display. However, due to the obvious degradation as result of the speed of display, the strategy is to skip frames. Although skipping frames helps to reduce the amount of data required, however, it disrupts the sequentiality of normal data retrieval. The I/O scheduling by the storage nodes must be interrupted to allow for a new scheduling sequence that permits intermittent blocks of data. 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. Considering the real-time implications in switching from one mode to another, it becomes imperative that efficient strategies are necessary for handling fast-forwarding of video displays.

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. The fact that the storage devices cannot satisfy all the arriving requests due to their 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. It is 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 replica 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 . Let's illustrate the above problem with an example that comprises a number of different situations.

Example: Let $\delta = 2$, $\eta = 10$, $\kappa = 2$, and $x \Rightarrow Snode_j$ implies that x number of I/O requests are directed to Snode $Snode_j$. Therefore, every video segment has two copies stored as defined by δ ; there is a total of 10 customers in the system; each customer may have up to two concurrent streams; and $\Pi = 20$. We use the objects shown in Figure 4 and assume the existence of two other Snodes, $Snode_3, Snode_4$, to accomplish the required replication. If every customer establishes 2 video streams for O^1 and simultaneously references AU_3^1 , then since there are two copies of AU_3^1 ,

$$10 \Rightarrow Snode_1.$$

Similarly, if every customer requests 2 video streams, one for O^1 and the other for O^2 , and simultaneously references AU_3^1 and AU_2^2 , then the same distribution of requests occurs. On the other hand, if every customer maintains 2 video streams of the two video objects and references two AUs

$$\begin{aligned} & (AU_i^1, AU_j^2) \text{ for } i \neq j \text{ or} \\ & (AU_i^1, AU_j^1) \text{ for } j = i + 1 \text{ or } i = j - 1 \text{ or} \\ & (AU_i^2, AU_j^2) \text{ for } j = i + 1 \text{ or } i = j - 1 \text{ or} \\ & (AU_i^1, AU_j^2) \text{ for } (i \bmod 2) = (j \bmod 2), \end{aligned}$$

at a given time, then similar distribution of requests occurs.

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, or using memory interleaving algorithms. 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 residence (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 scheduled network buffer transfer that may precede a given buffer, whenever applicable.

5 Stream Display Factors

In order to present design solutions to the problems associated with the management of streams, 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, which is greatly affected by a video object's data availability rate. Remember that an object's data availability rate influences the size of its AU which impacts the amount of the object's data that can be retrieved by unit time. If an object's availability rate is not sufficient to accommodate its consumption rate, then the potential for starvation is increased; inordinate amount

of buffering becomes necessary; and the start-up time is greatly affected. The consumption rate, i.e., the volume of data to display (or consumed) in a unit time, depends on how readily or the rate at which that data is buffered just before display. When the availability rate is low, the waiting time to display is high, and frames or AUs can be displayed but very slowly. In the worst case, when the availability rate is too low, it is possible to display AUs or frames in a still or snap-shot mode instead of a continuous mode.

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.

The amount of time required for the muv of an object to become available is a function of the dcr and muv . From the above derivations, one can deduce that, from the start of a video stream, it takes $stime$ time before data are actually sent to the display device. $stime$ is a simple ratio given by

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

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 is a factor 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

$$bur \leq (bufnum \times r). \quad (4)$$

The variable r is a factor necessary for multi-user environments to help balance the utilization of the system resources. It is usually determined by the system's resource reservation subsystem and mostly determined dynamically. It is a non-uniform value for all the processes. It is expected that streams with high data consumption rate should have higher r values than those with lower data consumption rates. If bur is bounded from below by equation (3), it can be deduced that bur is a bounded from above by inequality (4) as

$$\lceil \frac{muv}{bufsize} \rceil \leq bur \leq (bufnum \times r) \quad (5)$$

For example, given that a system has 1000 buffers; each buffer is of size $100KB$; a process can use a maximum of 2% of the buffers; we want to display an object with an expected retrieval rate of $500KB$; and the minimum amount of data that must be available, at any given time, from the secondary storage devices is $1500KB$; therefore,

$$\begin{aligned}
 buf^{num} &= 1000, \\
 buf^{size} &= 100, \\
 r &= 0.02, \\
 dcr = err &= 500, \\
 \text{start-up time, } stime &= 3, \\
 muv = 1500 \text{ where } n &= 3, \\
 \text{and } 15 \leq bur &\leq 20.
 \end{aligned}$$

In the example above, 3 time units must elapse from the time of request initiation to the time of actual display of data. Each I/O request to the secondary storage devices must deliver 500KB per unit time, and there must be at least 15 free buffers for each I/O request to the devices.

5.1 Multi-user and Network Effects

The availability of the required number of buffers to satisfy *stime* and subsequent continuous display of data, as described above, apply to the best case situation. However, since most operating environments are multi-user or distributed, the inter-process buffer sharing and network delays affect buffer availabilities.

In the best case scenario, buffers become available on request so that *stime* and the continuous-display requirements are maintained and achieved as needed. In other words, there may not be buffer contentions or network delays to exacerbate display quality and efficiency. However, in multi-user system environments, where any number of users and processes may be active concurrently, buffer contention between processes arise. Furthermore, in a distributed environment, where data must be carried over networks, network delays become a concern. The degree of contention, processes' buffer utilizations, and network issues affect display quality and efficiency of continuous display. Therefore, with respect to a system environment, it may be necessary to accentuate display buffer acquisition characteristics to account for potential delays.

5.2 Accentuating Buffer Acquisition Factors

When buffer contention is anticipated or network delays are imminent, which may lead to data availability delays, the data acquisition factors should be adjusted to mitigate the adverse effects. Obviously, it is more efficient to make the adjustments at the start of display than during the period between frame or AU retrieval. We can use the actual or real start-up time (*rtime*) vis-à-vis the expected time to the start of a display, (*stime*), to predict the degree of system buffer contentions or network delays. The real start-up time is the observed time to achieve *muv* as opposed to the computed time. [Note that *rtime* would be typically larger than *stime* when a large

video object is displayed, and availability delay is imminent. For smaller objects, and in the presence of readily available data which would require less buffers, $rtime$ would typically be less than $stime$. Thus, the ratio between the $rtime$ and $stime$ can be used to accentuate the muv . The new muv computed as a result of adverse buffer contention or network effects is called the *accentuated muv* ($amuv$) and derived as

$$amuv = \left\lceil \frac{rtime}{stime} \right\rceil dcr$$

The objective of $amuv$ is to guarantee that after its initial satisfaction, subsequent consumption of data for display, and acquisition of buffers and data retrieval will maintain muv . Therefore, the delays incurred as a result of buffer contentions, especially for large video objects or AUs, do not affect display continuities.

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., MPEG-1 and MPEG-2), 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 have to be grouped together. Given that there are n^{dcr} different dcr rates, Figure 7 shows an example design for the request and data buffer queues. It shows a Request Hash Table (RHT) and Buffer Hash Table (BHT) where

- R^i is the i th dcr request queue ($1 \leq i \leq n^{dcr}$),
- B^j is the j th dcr request buffer queue ($1 \leq j \leq n^{dcr}$),
- 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 $i > m$.

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 average time for servicing the number of I/O requests (retrieval and transmission times) for the muv of 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, 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.

Figure 7: An example arrangement of request and buffer queues.

Each R^i is managed by the process P_i^{ret} while each B^i is managed by the process P_i^{sched} . Each of the processes 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. Reiterating from previous discussions, if

- t_r^i is the data retrieval overhead for a request in the i th dcr request queue,
- t_m^i is the preparation time for network transfer of a buffer in i th dcr request buffer queue,
- t_{wait}^i is the average buffer waiting time in the i th dcr request buffer queue,
- τ is the expected maximum elapsed time for the data arrival at an Nnode,

therefore, a P_i^{ret} makes a decision based on the t_r^i , t_m^i , and t_{wait}^i , about when to schedule its oldest request. The goal is that the following condition must be satisfied:

$$(t_{wait}^i + t_{wait}^{ret} + t_r^i + t_m^i + t^{network}) < \tau \quad (6)$$

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 defined as ϵ time units. When an I/O request has spent about a total time within ϵ time units to its expected waiting time, then it must be dispatched to the next queue. We use Figure 8 to illustrate the temporal implications of the transitions of a single I/O video data request. The time scales do not imply any time-magnitude relationships between the different events.

The following apply to Figure 8:

- 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_{wait}^i),
- 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., $t^{network} + t_m^i$).

Figure 8: An example transitions of an I/O request.

6.2 Algorithmic Consideration

A pseudo-code of **Algorithm I** describes the activities or functions of the processes in a typical DVDS. 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 inequality (5) 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
     $T = COMP1(R_{head}^i)$ 
    If  $[(T + \epsilon) > (\tau - \epsilon)]$  or RECVMSG do
      Extract  $R_{head}^i$ 
      Schedule on  $P^{ret}$ 
    end
  end
end

```

```

        endif
    endforever
endbegin
 $P_i^{sched}$ : begin
    Forever do
         $T = \text{COMP2}(R_{head}^i)$ 
        If ( $(T \geq (\tau - \epsilon))$  or RCVMSG) do
            Extract  $B_{head}^i$ 
            Schedule on  $P^{sched}$ 
        endif
    endforever
endbegin
endforall
endbegin

```

7 Effects of Distributed Environments

When satisfying the data availability rate of a multimedia object, we have assumed that the data is being transported only from permanent storage via the server to the client without any intervening transport medium. This is feasible if the users are directly connected to the multimedia servers, and the transmission time from server to client is inconsequential. However, nowadays, that situation rarely arises. The ubiquitous configuration usually involves various users connected to one or more servers via a Local or Wide Area Network (LAN, WAN). In that case, even if the I/O devices to server data transmission satisfies the data availability rate, there is no guarantee that the network infrastructure will cooperate accordingly. Even in the presence of a network, it is still required that an object's data availability rate be achieved at the client site. That implies that the network bandwidth and traffic must guarantee the achievability of the data availability rate. The bandwidth of a network is the amount of data that it can deliver from source to destination per unit time. The bandwidth of any network is affected greatly by the amount of traffic (requests) generated per unit time. Furthermore, when building the media delivery system over networks, we must also account for data loss and retransmissions. For certain multimedia data types, especially those that are continuous, data retransmission, most certainly, will adversely affect their on-time data arrival and utilization.

Even in the presence of a very efficient, effective, robust buffering strategy, the number of retransmissions and their temporal implications can become non-deterministic to be accounted for sufficiently by any buffering strategy. Consequently, network reliability becomes an important element in a multimedia delivery system.

For many multimedia applications, it is necessary that the underlying transmission networks have the capability of efficient and reliable resource management. This implies that the bandwidth for the data transmission from source to destination must be guaranteed prior to data leaving its storage site. This may become a problem when heterogeneous networks are involved, for example, as depicted in Figure 9 where T_1

and T_2 may be different types of networks with different protocols. This is one of the major problems that must be addressed by the much talked about ATM networks and protocols. The resource reservation requirement is one of the primary factors for request admission criteria and control for multimedia delivery networks.

Let's use Figure 9 to illustrate a temporal implication of multimedia delivery over heterogeneous networks. In Figure 9, let S_1, \dots, S_4 be multimedia servers where multimedia data are stored in permanent storage devices; each S_i , for $i = 1 \dots 4$ is connected to a network T_k , for $k = 1 \dots 4$, as shown; and U_1 is a client site where the multimedia data are requested by a user.

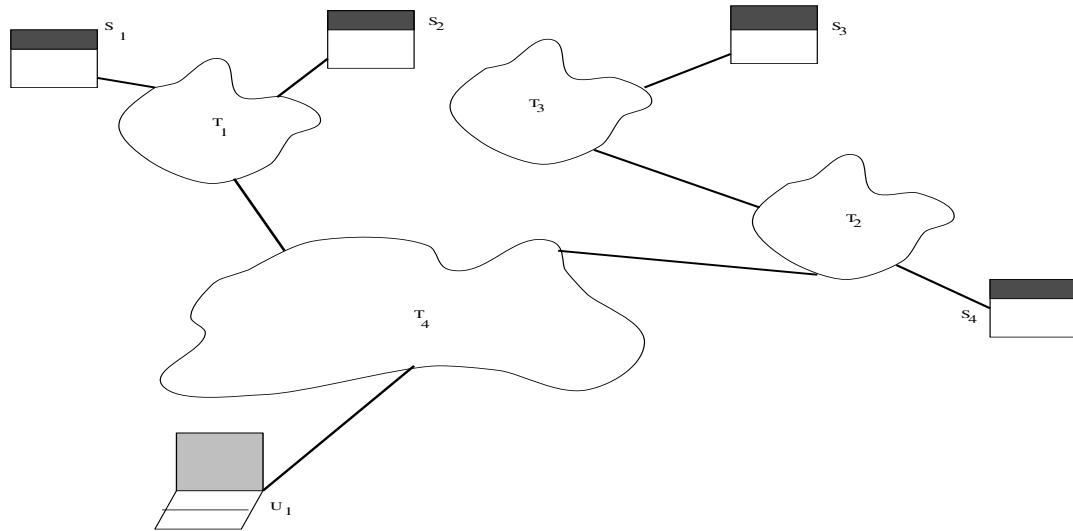


Figure 9: An example of a heterogeneous network environment

Let $\phi_{i,j}^{size}$ be the transmission time for *size* bytes of data between T_i and T_j and ρ_h^k the retrieval time of DAR_k at S_h . We assume that each S_h achieves any multimedia object's data availability rate.

Therefore, if t^{cur} is the current time (i.e., when a request is made); O_1, O_2 (which are stored at S_2 and S_3 , respectively) are required by U_1 at time ($t^{cur} + \delta$) for synchronized presentation, therefore, the following temporal condition must be satisfied:

$$\mathcal{P} = \rho_2^1 + \rho_3^2 \quad (7)$$

and

$$\mathcal{E} = \epsilon_2 + \epsilon_3 \quad (8)$$

and

$$\mathcal{T} = \left[\mathcal{E} + \mathcal{P} + t^{cur} + \sum_{k=1}^{\lfloor DAR_1/size \rfloor} \phi_{2,4}^{size} + \sum_{k=1}^{\lfloor DAR_2/size \rfloor} \phi_{3,4}^{size} \right] < t^{cur} + \delta \quad (9)$$

where ϵ_2, ϵ_3 are the times for the request messages to get to S_2, S_3 , respectively.

Taking cognizance of the condition derived in Equation 6, and assuming that $t^{network}$ here refers to the network of the distributed environment, therefore, we know that

$$t^{network} = \mathcal{T} - [\mathcal{E} + \mathcal{P} + t^{cur}] \quad (10)$$

An intelligent and robust VOD delivery system will be able to adjust some of its temporal factors as the system load/traffic changes. Specifically, if the network data transmission starts experiencing delay time beyond a threshold, the \mathcal{P} could be reduced by the retrieval system by, for example, increasing the priority of scheduling the buffers of some of the requests.

By all intents and purposes, it's easier to design a system to achieve the requirements of a given DAR with respect to ρ than to deterministically account for the ϕ values.

8 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 10. 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 P^{ret} and P^{sched} 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 P^{ret} through a named pipe ($pipe_{ret}$). After retrieving some given data, P^{ret} 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 P^{sched} . As is evident from Figure 10, 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 P^{ret} , P^{sched} , P_i^{ret} , and P_i^{sched} processes run in the Snodes while the $Pcont$ processes run in the Nnodes.

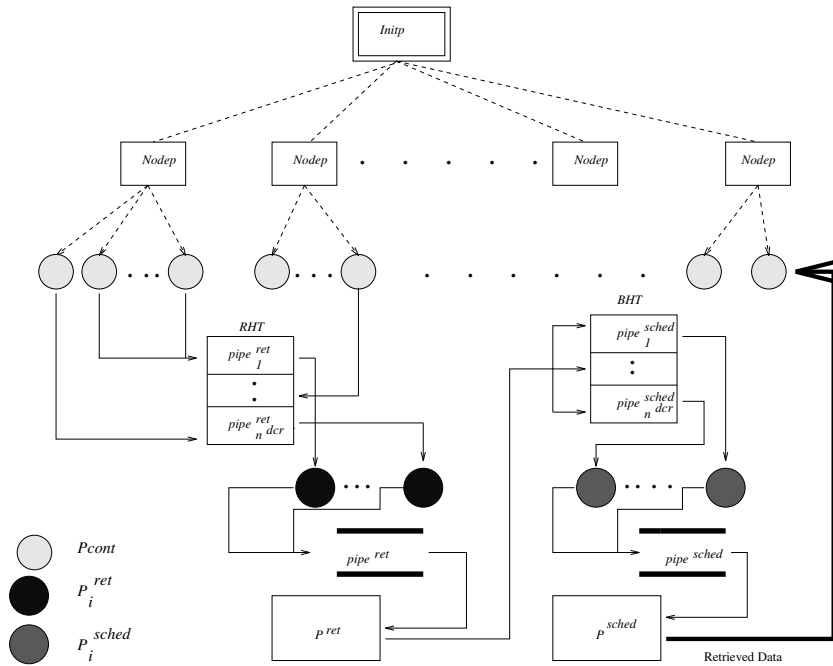


Figure 10: Simulation inter process communication for streams management.

8.1 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 11 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.

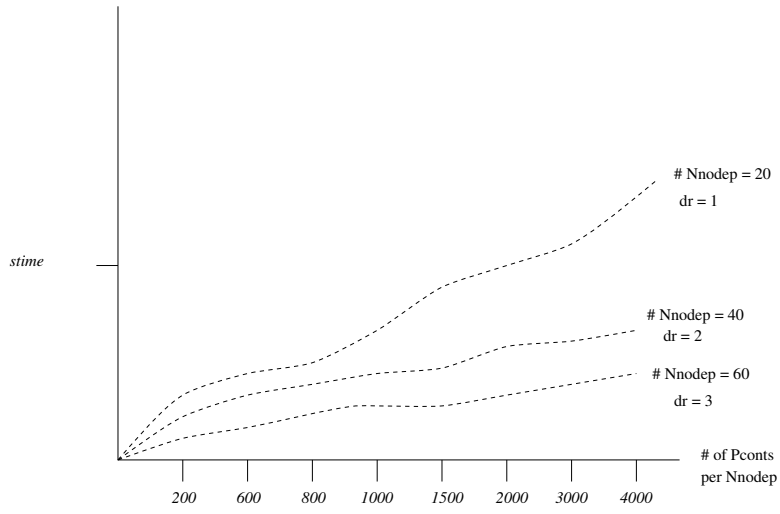


Figure 11: Example of impacts of increasing *Pcont*'s and/or degree of replication

9 Conclusions

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 has mainly depended 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. The only 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 result 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] 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.
- [2] 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.
- [3] T. D. C. Little and D. Venkatesh, "Prospects for Interactive Video-on-Demand Television," *IEEE Multimedia*, Vol. 1, No. 3, Fall 1994, pp. 14-24.
- [4] W. Hodge, S. Martin, J. T. Powers, jr., "Video On Demand: Architectures, Systems, and Applications," *SMPTE Journal*, Vol. 102, No. 9, Sept. 1993, pp. 791-803.
- [5] Y-H. Chang et al, "An Open Systems Approach to Video on Demand," *IEEE Communications*, Vol. 32, NO. 5, May 1994, pp. 68-80.
- [6] T. D. C. Little et al, "A Video-on-Demand System Supporting Content-Based Queries," *Proc. ACM Multimedia*, ACM Press, NY, 1993, pp. 427-436.
- [7] D. A. Paterson, G. Gibson, R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks," *ACM SIGMOD Conf.*, ACM Press, NY, 1988, pp. 109-116.

- [8] 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.
- [9] 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.
- [10] 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.
- [11] 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.
- [12] Kingsley. C. Nwosu, "Data Storage Modeling and Management for Multimedia Information Systems," *Ph.D. Dissertation*, Syracuse University, Syracuse, NY, Dec. 1993, pp. 1-166.
- [13] Y-T. Chen, R. L. Kashyap, A. Ghafoor, "Physical Storage Management for Interactive Multimedia Information System," *Proc. 1992 IEEE International Conference on Systems, Man, and Cybernetics*, Jan. 1993.
- [14] Ralf Steinmetz, "Synchronization Properties in Multimedia Systems," *IEEE Journal on Selected Areas in Communications*, Vol 8, No.3, April 1990.
- [15] Thomas D.C. Little, Arif Ghafoor, "Network Considerations for Distributed Multimedia Object Composition and Communication," *IEEE Network Magazine*, November 1990.
- [16] Rangan P. Venkat, Vin Harrick M., "Efficient Storage Techniques for Digital Continuous Multimedia," *Proc. International Conf. on Knowledge and Data Engineering*, August 1993, pp. 564-573.
- [17] Kenchamma-Hosekote D. R., Srivastava J., "Scheduling Continuous Media in a Video-On-Demand Server," *Proceedings of the International Conference on Multimedia Computing and Systems*, May 15-19, 1994, Boston, MA.
- [18] Sincoskie W. D., "Video on Demand: Is It Feasible?," *Proc. 1990 IEEE Global Telecommunications Conference & Exhibition*, San Diego, CA., Dec. 02-05, 1991, pp. 201-205.
- [19] P. Lougher and D. Shepherd, "The Design of a Storage Server for Continuous Media," *The Computer Journal*, Vol. 36, No. 1, 1993, pp. 32-42.

- [20] K. Salem and H. Barcia-Molina, "Disk Striping," *Proc. 2nd Int'l Conf. on Data Engineering*, IEEE CS Press, Washington, D.C., 1986, pp. 336-342.
- [21] Y. N. Doğanata and A. N. Tantawi, "A Cost Performance Study of Video Servers with Hierarchical Storage," *Proc. 1st Int'l Conf. on Multimedia Computing Systems*, IEEE CS Press, Los Alamitos, California, 1994, pp. 14-19.
- [22] W. D. Sincoskie, "System Architecture for a Large Scale Video on Demand Service," *Computer Networks and ISDN Systems*, Vol. 22, 1991, pp. 155-162.