Design of a Scalable Multimedia Storage Manager*

Steven Berson, Ali Dashti, Martha Escobar-Molano,
Shahram Ghandeharizadeh, Leana Golubchik, Richard Muntz, Cyrus Shahabi

Abstract

During the past decade, the information technology has evolved to store and retrieve continuous media data types, e.g., audio and video objects. Unlike the traditional data types (e.g., text), this new data type requires its objects to be retrieved at a pre-specified bandwidth. If its objects are retrieved at a rate lower than its pre-specified bandwidth then a display will suffer from frequent disruptions and delays termed hiccups. Storage managers that support a hiccup-free retrieval of continuous media data types are commonly referred to as multimedia storage managers. They are expected to play a major role in many applications including library information systems, entertainment technology, educational applications, etc.

For the past three years, we have been investigating the design of a multimedia storage manager. The goals of the project are to realize a storage manager that is: 1) economically viable, 2) scalable, 3) fault tolerant, and 4) provides a fraction of a second latency time. By scalable, we mean that the system should be able to grow incrementally as the size of an application grows in order to avoid a degradation in system performance. To realize this objective, the overhead of techniques employed by the system should not increase prohibitively as a function of additional resources. This paper provides an overview of our proposed techniques based on a hierarchical storage platform to construct a scalable multimedia storage manager.

*This research was supported in part by grants from AT&T, Hewlett-Packard, IBM grant SJ92488, and NSF grants IRI-9110522, IRI-9258362 (NYI award), and CDA-9216321. S. Berson is with USC Information Sciences Institute. A. Dashti, M. Escobar-Molano, S. Ghandeharizadeh, and C. Shahabi are with USC Computer Sicence Department. L. Golubchik and R. Muntz are with UCLA Computer Science Department.

1 Introduction

During the past decade, the information technology has evolved to where it is economically viable to store and retrieve continuous media data types (e.g., audio and video). These systems are expected to play a major role in library information systems, educational applications, entertainment industry, etc. A challenging task when implementing these systems is to support a continuous display of video objects (and high quality audio objects). This is because:

- Video objects require a high bandwidth for their continuous display. For example, the bandwidth required by NTSC for "network-quality" video is about 45 megabits per second (mbps) [Has89]. Recommendation 601 of the International Radio Consultative Committee (CCIR) calls for a 216 mbps bandwidth for video objects [Fox91]. A video object based on HDTV requires a bandwidth of approximately 800 mbps.
- Video objects are large in size. For example, a 30 minute uncompressed video clip based on NTSC is 10 gigabytes in size. With a compression technique that reduces the bandwidth requirement of this object to 1.5 mbps, this object is 337 megabytes in size. A repository (e.g., corresponding to an encyclopedia) that contains hundreds of such clips is potentially terabytes in size.
- The bandwidth of the current magnetic disk technology is typically rated between 20 to 30 mbps (and is not expected to improve much in the near future [PGK88, Pat93]).

One may employ a lossy compression techniques (e.g., MPEG) in order to reduce both the size and the bandwidth requirement of an object. These techniques encode data into a form that consumes a relatively small amount of space, however, when the data is decoded, it yields a representation similar to the original (some loss of data). While they are effective, there are applications that cannot tolerate the use of a lossy compression technique (e.g., video signals collected from space by NASA [Doz92]). Clearly, a technique that can support the display of an object for both application types is desirable. Assuming a multi-disk architecture, staggered striping [BGMJ94] is one such a technique. It is flexible enough to support those objects whose bandwidth requires either the aggregate bandwidth of multiple disks or fraction of the bandwidth of a single disk. Hence, it provides effective support for a database that consists of a mix of media types, each with a different bandwidth requirement. Moreover, its design enables the system to scale to thousands of disk drives.

To reduce cost, the storage manager is envisioned to be hierarchical [GS93] consisting of main memory, magnetic disk drives, and one or more tertiary storage devices (e.g., Ampex DST [Joh93], Digital Audio Tape [Sch93], CD-ROM [CP94, Com94], etc.). As the different levels of this hierarchy are traversed starting with memory, the density of the medium (the amount of data it can store)

and its latency increases, while its cost per megabyte decreases[1]. (At the time of this writing, from \$50/megabyte of memory to \$0.6/megabyte of disk storage to less than \$0.1/megabyte of a tertiary storage device.) The database resides permanently on the tertiary storage device. The disk storage is used as a temporary staging area for the more frequently accessed objects in order to minimize the number of references to the tertiary store. An application referencing an object that is disk resident observes the average latency time of this device (which is superior to that of a tertiary storage device). The memory is used as a temporary staging area to either display an object or render an object disk resident from tertiary store. The migration of data from the tertiary memory to disk (and from disk to memory for display) is transparent to the user. Ideally, the number of references to the tertiary should be minimized in order to minimize the latency time of requests. We provide an overview of dynamic replacement policies [GS94] that manage the disk resident portion of objects in order to achieve this objective. Moreover, we describe a pipelining mechanism [GDS94b] that overlaps the display of an object with its materialization in order to minimize the latency observed by a user referencing a tertiary resident object.

Depending on the requirements of an application, a multimedia storage manager may consist of a large number of disks – potentially thousands of disk drives. A large number of disks would provide both the bandwidth required to service many requests simultaneously and the storage capacity required to minimize the number of reference to the tertiary storage device. However, while a single disk can be fairly reliable, the probability of a disk failure in such a large system is high. For example, if the mean time to failure (MTTF) of a single disk is on the order of 100,000 hours, the MTTF of a disk in a 1000 disk system is on the order of 100 hours. In order to enable the system to continue operation in the presence of failures, we have introduced techniques that extend staggered striping to store redundant information that enable the system to reconstruct the contents of a failed disk [BGM94].

The rest of this paper is organized as follows. Assuming a fixed bandwidth for each media data type, Section 2 describes our striping techniques. In Section 3, we extend striping with an availability technique that enables the system to continue operation in the presence of disk failures. Section 4 describes how an object is materialized from the tertiary storage device. Subsequently, in Section 5, we describe a pipelining mechanism that overlaps the display of an object with its retrieval from tertiary in order to minimize the latency time of the system. Section 6 presents alternative management policy for the available disk space and their tradeoffs. In Section 7, we relax the fixed

[1] The transfer rate of a tertiary storage device might be either lower or higher than that of a magnetic disk drive depending on the assumed device. For example, the bandwidth of an Ampex DST drive is 116 mbps, while that of an EXABYTE is 4 mbps.

bandwidth requirement and discuss compressed objects with their variable bandwidth requirement from the disk subsystem. We describe a taxonomy of techniques to support this variant bandwidth requirement. In Section 8, we survey the work related to this study. Our conclusions and future research directions are contained in Section 9.

2 Striping

For the striping techniques described in this section, we assume the following: 1) each object has a constant bandwidth requirement, 2) the display stations have a limited amount of memory, which means that the data has to be produced at approximately the same rate as its consumption rate at a display station, 3) the network delivers the data to its destination both reliably and fast enough, which consequently, eliminates it from further consideration by this paper, and 4) the bandwidth requirements of the objects exceeds the bandwidth of both the tertiary storage device and a single disk drive (this assumption is relaxed in Section 2.2.2). We start by describing the simple striping technique. Next, we describe the limitations of this technique for a database that consists of a mix of media types, and present staggered striping as a solution to these limitations. (See Tables 1 and 2 for parameters and terms, respectively, used throughout this paper.)

Parameter	Definition
$B_{Display}(X)$	Bandwidth required to display object X
tfr	Transfer rate of a single disk drive
B_{Disk}	Effective bandwidth of a single disk drive
$B_{Tertiary}$	Bandwidth of the tertiary storage device
M_X	Degree of declustering for object X, $M_X = \lceil \frac{B_{Display}(X)}{B_{Disk}} \rceil$
D	Number of disk drives in the system
R	Number of disk clusters in the system
T_{switch}	The delay incurred when the system switches from one cluster to another
T_{sector}	Time required by a disk to transfer a sector worth of data to memory
$S(C_i)$	Service time of a disk cluster per activation
size(fragment)	Size of a fragment
size(subobject)	Size of a subobject
k	The distance (number of disks) between $X_{i.0}$ and $X_{i+1.0}$
n	Number of subobjects in X, $n = \frac{size(X)}{size(subobject(X))}$
$PCR(X)$	Production Consumption Ratio, $\frac{B_{Tertiary}}{B_{Display}(X)}$

Table 1: List of parameters used repeatedly in this paper and their respective definitions

2.1 Simple Striping

Assuming a fixed bandwidth for each disk (B_{Disk}) in the system, and a database consisting of objects that belong to a single media type with bandwidth requirement $B_{Display}$, we must utilize

Degree of Declustering (M_X): the number of disk drives a subobj ect is declustered across; $M_X = \lceil \frac{B_{Display}(X)}{B_{Disk}} \rceil$.

Fragment: unit of data transfered from a single disk drive. Constructed by declustering a subobject (X_i) across M_X disk drives.

Subobject: a stripe of an object. It represents a contiguous port ion of the object. Its size is defined as $M_X \times size(fragment)$.

Disk cluster: a group of disk drives that are accessed concurrently to retrieve a subobject (X_i) at a rate equivalent to $B_{Displa y}(X)$.

Stride (k): the distance (i.e., number of disk drives) between the first fragment of subobject X_i and the first fragment of X_{i+1}.

Table 2: Defining terms

the aggregate bandwidth of at least $M = \lceil \frac{B_{Display}}{B_{Disk}} \rceil$ disk drives to support a continuous display of an object[2]. This can be achieved by a method we call *simple striping* as follows. First, the D disk drives in the system are organized into $R = \frac{D}{M}$ disk clusters. Next, each object in the database (say X) is organized as a sequence of n equi-sized **subobjects** (X_0,X_1,...,X_n). Each subobject X_i represents a contiguous portion of X. When X is materialized from the tertiary storage device, its subobjects are assigned to the clusters in a round-robin manner, starting with an available cluster. In a cluster, a subobject is declustered [RE78, LKB87, GD90] into M pieces (termed **fragments**), with each fragment assigned to a different disk drive in the cluster.

To illustrate, assume that object X requires a 60 mbps bandwidth to support its continuous display ($B_{Displa y}(X)$=60 mbps). Moreover, assume that the system consists of 9 disk drives, each with a bandwidth of 20 mbps (B_{Disk}=20 mbps). Thus, we need the aggregate bandwidth of 3 ($M = \frac{60}{20}$) disk drives to support the continuous display of X. Figure 1 demonstrates how the simple striping technique organizes the subobjects of X. In this figures, the disk drives are partitioned into 3 clusters ($\lfloor \frac{D}{M} \rfloor$), each consisting of 3 (M) disk drives. Each subobject of X (say X_1) is declustered into 3 fragments (denoted $X_{1.0}$, $X_{1.1}$, $X_{1.2}$). The fragments of a subobject (say X_1) are constructed using a round-robin assignment of its blocks to each disk drive (see Figure 1), allowing the system to overlap the display of X_1 with its retrieval from the disk drives (multi-input pipelining [GRAQ91, GR93]). This minimizes the amount of memory required for buffering the data. However, in practice, some memory is needed per disk drive to eliminate hiccups that may arise due to disk seeks incurred when the system switches from one cluster to another.

[2]For simplicity our initial discussion assumes that the minimum bandwidth allocated to a request is B_{Disk}; in a later section we show how fractions of B_{Disk} can be allocated.

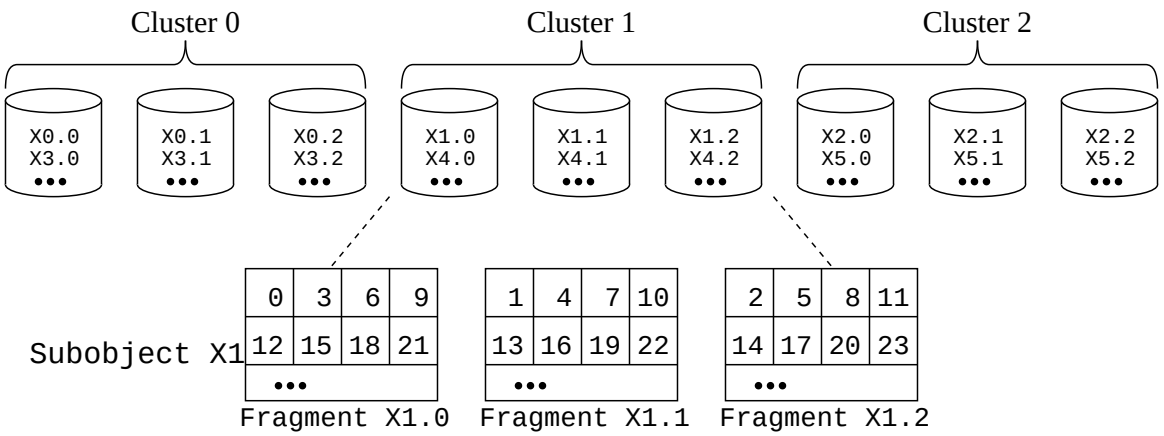


Figure 1: Fragment layout on disks

When the request service displays X, it employs a single cluster at each time interval. However, when the system switches from one cluster (C_0) to another (C_1), the disk drives that constitute C_1 incur the seek and latency times associated with repositioning their heads to the location containing the referenced fragments. To eliminate hiccups that might be attributed to this factor, simple striping computes the worst case delay (termed T_{switch}) for C_1 to reposition its heads and, relative to the consumption rate of a display station, produces the data such that a station is busy displaying T_{switch} worth of data when the switch takes place (see Figure 2).

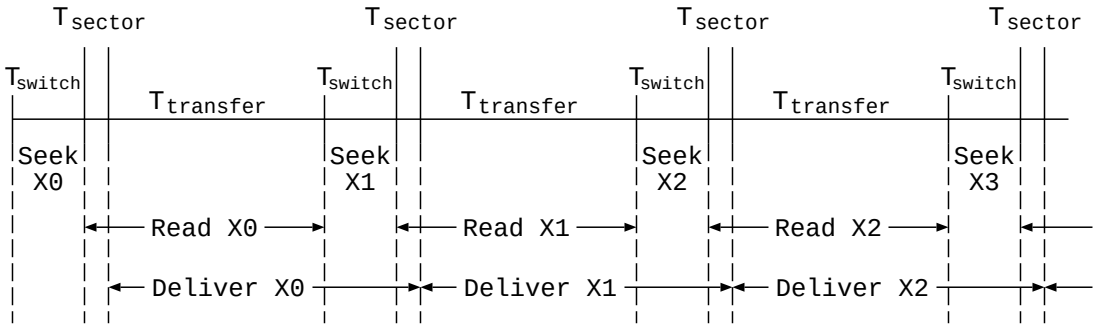


Figure 2: Fragment layout on disks

Assuming some memory is allocated for each disk drive, the protocol for this paradigm is as follows. Upon activation of the disk drives in a cluster, each disk drive performs the following four steps:

1. Each disk repositions its head (this takes between 0 to T_{switch} seconds)

2. Each disk starts reading its fragment (it takes T_{sector} seconds to read each sector)

3. When all disks have read at least one sector, the synchronized transmitting of data to the

display stations is started

4. Disks continue reading of the complete fragment overlapped with transmission to the display station

T_{switch} represents the maximum duration of the first step. T_{sector} is the time required to read a sector into memory. The minimum amount of required memory is a function of these two times and is defined as:

$$B_{disk} \times (T_{switch} + T_{sector}) \tag{1}$$

Simple striping splits time into fixed length intervals. A **time interval** is the time required for a disk drive to perform its four steps, and constitutes the service time of a cluster (denoted $S(C_i)$). The duration of a time interval is dependent on the physical characteristics of the secondary storage device (its seek and latency times, and transfer rate), and the size of the fragments. To illustrate, recall the physical layout of X in Figure 1. Once a request references X, the system reads and displays X_0 (using C_0) during the first time interval. The display of the object starts at step 3 of this time interval. During the second time interval, the system reads and displays X_1 (using C_1). The display time of the cached data eclipses the seek and latency time incurred by C_1 (step 1), providing for a continuous retrieval of X (see Figure 2). This process is repeated until all subobjects of X are displayed.

The fragment size is a parameter that has to be decided at system configuration time. The larger the chosen fragment size, the greater the effective disk bandwidth. This is because after the initial delay overhead to position the read heads (T_{switch}), there is little additional overhead no matter how much data is read. More formally if tfr is the transfer rate of a single disk, then the effective disk bandwidth B_{disk} can be computed as:

$$B_{disk} = tfr \times \frac{size(fragment)}{size(fragment) + (T_{switch} * tfr)}$$

There is also a tradeoff between the effective disk bandwidth and the time to initiate the display of an object. At the instant of arrival of a new request referencing an object X, the cluster containing X_0 might be busy servicing another request while the other clusters are idle. In this case, the request has to wait until the cluster containing X_0 becomes available. For example, if a system consists of R disks clusters and is almost completely utilized servicing $R-1$ requests, then in the *worst case* the latency time for a new request is $(R-1) * S(C_i)$. In summary, as one increases the size of a fragment, the display latency time increases (undesirable) while the effective disk bandwidth increases (desirable).

Without loss of generality, and in order to simplify the discussion, for the rest of this paper we assume that the size of a fragment for each object i is two cylinders (size(subobject)=M_i * 2 * $size(cylinder)$). This is a reasonable assumption because: 1) it wastes only about 10% of the disk bandwidth, and 2) the advantages of transferring more than 2 cylinder from each disk drive is marginal because of diminishing gains in effective disk bandwidth beyond 2 cylinders.

Simple striping provides for a flexible allocation of disk bandwidth and more efficient use of disk storage capacity. However, when the database consists of a mix of media types each with a different bandwidth requirement, the design of simple striping should be extended to minimize the percentage of wasted disk bandwidth. To illustrate, assume that the database consists of two video objects: Y and Z. The bandwidth requirement of Y is 120 mbps ($M_Y = 6$) and that of Z is 60 mbps ($M_Z = 3$). An approach to support these objects might be to construct the disk clusters based on the media type that has the highest bandwidth requirement, resulting in 6 disks per cluster (assuming $B_{Disk} = 20$ mbps). This would cause the system to employ a fraction of disks in a cluster when servicing a request that references object Z, sacrificing 50% of the available disk bandwidth. Staggered striping, described in the next section, is a superior alternative as it minimizes the percentage of disk bandwidth that is wasted.

2.2 Staggered Striping

Staggered striping is a generalization of simple striping. It constructs the disk clusters logically (instead of physically) and removes the constraint that the assignment of two consecutive subobjects of X (say X_i and X_{i+1}) be on non-overlapping disks. Instead, it assigns the subobjects such that the disk containing the first fragment of X_{i+1} (i.e., $X_{i+1.0}$) is k disks (modulo the total number of disks) apart from the disk drive that contains the first fragment of X_i (i.e., $X_{i.0}$). The distance between $X_{i.0}$ and $X_{i+1.0}$ is termed *stride*. If a database consists of a single media type (with a degree of declustering M_x) and D is a multiple of M_x then staggered striping can implement simple striping by setting the stride to equal to the degree of declustering of an object[3] ($k = M_X$).

Figure 3 illustrates both the logical and physical assignment of the subobjects of X with staggered striping and stride k=1 (for the moment we ignore objects Y and Z). As compared with simple striping, the display of X with staggered striping differs in one way: after each time interval, the disks employed by a request shift k to the right (instead of M_X with simple striping). When the database consists of a mix of media types, the objects of each media type are assigned to the disk drives independently but all with the same stride. Figure 3(a) demonstrates the assignment of objects X, Y, and Z with bandwidth requirement of 60, 40, and 20 mbps, respectively (M_X=3, M_Y=2, M_Z=1). The stride of each object is 1. In order to display object X, the system locates the

[3]Virtual data replication can be implemented by $k = D$.

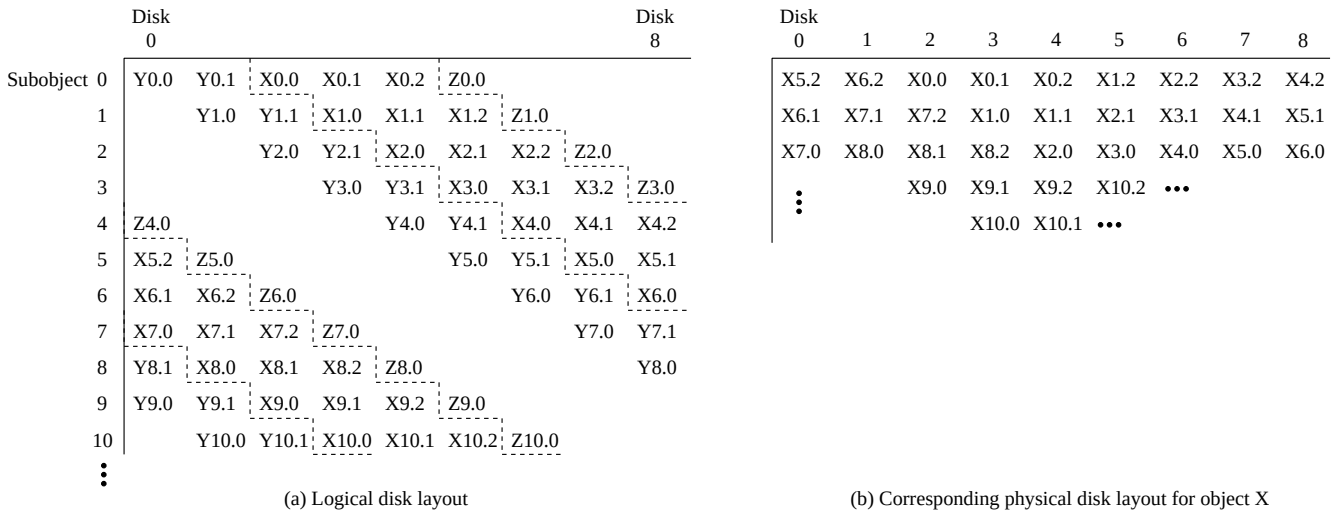


Figure 3: Staggered striping with 8 disks

M_X logically adjacent disk drives that contain its first subobject (disks 2, 3, and 4). If these disk drives are idle, they are employed during the first time interval to retrieve and display X_0. During the second time interval, the next M_X disk drives are employed by shifting k disks to the right.

With staggered striping, it is easy to accommodate objects of different display bandwidths with little loss of disk bandwidth. The degree of declustering of objects varies depending on their media type. However, the size of a fragment (i.e., unit of transfer from each disk drive) is the same for all objects, regardless of their media type. Consequently, the duration of a time interval is constant for all multimedia objects. For example, in Figure 3, the size of subobject Y_i is twice that of subobject Z_i because Y requires twice the bandwidth of object Z. However, their fragment size is identical because Y_i is declustered across twice as many disks as Z_i.

2.2.1 Stride

The choice of a value for the stride (k) must be determined at system configuration time. It may vary in value from 1 to D since a value (say i) greater than D is equivalent to i modulo D. The choice of k and D is important as particular combinations of values for k and D can result in very skewed load on the disks, both storage capacity and bandwidth capacity. We illustrate the above points by first considering the two extreme values for k (1 and D). Assume a system with 10 disk drives (D=10) and a large object X consisting of hundreds of cylinders worth of data. Assume that the degree of declustering for each subobject of X is 4 (M_X=4). If the system is configured with k=1, then the number of unique disks employed is 10 (D), 4 at a time and for $S(C_i)$ duration before moving to a new set of 4 disks. If k=D, then all subobjects of X are assigned to the same disk drive. Hence, the number of unique disks employed to display X is M_X, each for the entire display time of X ($\frac{size(X)}{B_{Display}(X)}$). Assume requests for objects X and Y arrive to both systems (one system

with k=1 and the other with $k = D$) and where $X_{0.0}$ and $Y_{0.0}$ reside on the same disk. Assume that the requests referencing X is followed by the one referencing Y. With k=1, the second request observes a delay equivalent to $S(C_i)$ (typically less than a second). With k=D, this request observes a delay equivalent to the display time of X which is much larger and generally unacceptable. To prevent data skew, the subobject size of every object in the system must be a multiple of the GCD (Greatest Common Divisor) of D, the total number of disks, and k (the stride). In particular, a stride of 1 guarantees no data skew. Similarly, any choice of D and k such that D and k are relatively prime guarantees no data skew.

Note that with k=D, the display of each object is very efficient because the system can cluster the different subobjects of X on adjacent cylinders in order to minimize the percentage of disk bandwidth that is wasted (saves less than 10% of the disk bandwidth). Simulation results presented in [BGMJ94] demonstrate the tradeoffs between these two alternative values for k and show that saving of less than 10% of the disk bandwidth as compared to the high probability of collisions is not beneficial. When k ranges in value between 1 and D, the size of an object X determines the number of disk drives employed to display X because the size of each fragment is fixed (a cylinder in our case). To illustrate, assume D=100 and an object X consist of 100 cylinders ($M_X = 4$). With $k = M_X$ (i.e., simple striping), X is spread across all the D disk drives. However, with k=1, X is spread across 28 disk drives. In this case, the expected display latency with $k = 1$ is higher than with $k = M_X$.

2.2.2 Low Bandwidth Objects

There are object types for which $B_{Display} < B_{Disk}$. These might include certain types of audio or slow scan video. Similarly, there are objects whose bandwidth requirement is not an exact multiple of B_{Disk}. In these cases there will be wasted disk bandwidth due to the request to use an integral number of disks. For example, an object requiring 30 mbps when $B_{Disk} = 20$ would waste 25 percent of the bandwidth of the two disks used per interval. Staggered striping can be used to more efficiently support these low bandwidth objects at a cost of some additional buffer space. To efficiently use disk space and disk bandwidth, we propose that subobjects of 2 or more low bandwidth objects be read and delivered in a single time interval. Consider two subobjects X_i and Y_j, each of which has $B_{Display} = 1/2\ B_{Disk}$, and which are to be read during the same time interval. The data in subobject X_i needs to be delivered during the entire time interval including the time when Y_j is being read. An additional buffer can be used to store part of X_i while subobject Y_j is being read. Similarly, part of Y_j needs to be buffered while X_{i+1} is being read during the next time interval. Note that we are again assuming that a node can concurrently transmit from a main memory buffer and from a disk using the pipelining scheme.

Figure 4 illustrates how this is accomplished. During the first half of the first time interval

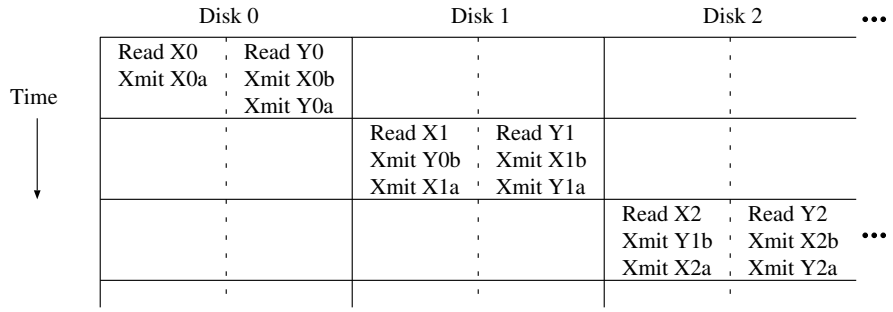


Figure 4: Staggered striping with low bandwidth objects

subobject X_0 is read, and the first half of X_0 (represented as X0a) is transmitted using pipelining. The second half of subobject X_0, X0b, is buffered for transmission during the second half of the time interval. In the second half of the first time interval, subobject Y_0 is read and both Y0a and X0b (from the buffer) are transmitted. Y0b now needs to be buffered for transmission during the first half of the second time interval. This process continues until the objects complete transmission.

This scheme effectively divides each disk into two logical disks of approximately one half the bandwidth of the original disk. This scheme can also be beneficial in reducing the overhead due to use of an integral number of disks. In effect, the request is now that an integral number of logical disks be allocated to a request. For example, an object that has $B_{Display} = 3/2\ B_{Disk}$ can be exactly accommodated with no loss due to rounding up the number of disks. In general, the waste due to rounding is reduced.

2.2.3 Fragmentation

Staggered striping may cause a fraction of the disk drives to remain idle even though there are requests waiting to be serviced. This occurs when the idle disk drives are not adjacent due to the display of other objects. This limitation is termed *bandwidth fragmentation*. To illustrate, consider the assignment of X, Y, and Z in Figure 3. Assume that an additional object (say W) with a degree of declustering of 4 is disk resident (M_W=4). Suppose the placement of W_0 starts with disk 2 ($W_{0.0}$ is stored on the same disk drive containing $X_{0.0}$). If the system is busy servicing three requests referencing objects X, Y, and Z, then there are three disks that are idle. Assume that a new request arrives referencing object W. It would have to wait because the number of idle disks (3) is less that that required to display W (4). If the display of object X now completes then there are six idle disks. However, the system is still unable to display object W because the available disk drives are not adjacent to each other (they are in groups of three separated by the display of Y and Z). The system cannot service displays requiring more than three disks until the display of either Y or Z completes.

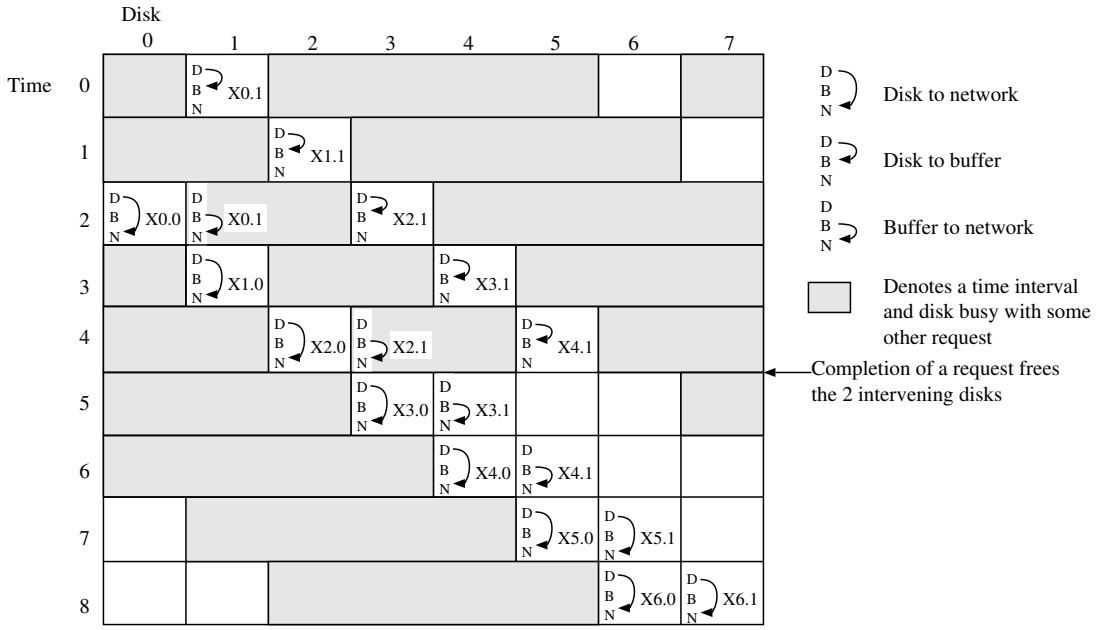


Figure 5: Utilizing fragmented disks

Bandwidth fragmentation can be alleviated by careful scheduling of jobs, but cannot be completely eliminated. However, with additional memory for buffer space and additional network capacity, the bandwidth fragmentation problem can be solved. To accomplish this, assume that a fragment can be read from the disk into a buffer in one time interval and, in a subsequent time interval, the same processor node can concurrently transmit to the network both (a) the previously buffered fragment, and (b) a disk resident fragment (using the pipelining scheme outlined earlier). In this section, we show how to use buffers to utilize a set of disks that are not adjacent to deliver an object.

Figure 5 illustrates an example of how our approach works. In the figure, the white reqions indicate which disks were available for serving new requests while the shaded regions are disks busy serving other requests. A request arrives at time 0 for an object X with a degree of declustering equal to 2. Further the stride is 1 in this example and the initial subobject X0 is stored on disks 0 and 1. There are 2 free disks, but they are not consecutive; there are 2 intervening busy disks. Disk 1 is free and is in position to read fragment X0.1, but disk 6 which is also free will not be in position to read fragment X0.0 until time interval 2. In order to support bandwidth fragmented delivery of object X, disk 1 can keep fragment X0.1 in memory until time 2 when it can be delivered along with fragment X0.0. Thus at time 2, fragment X0.0 is pipelined directly from disk 0 to the network, while node 1 transmits fragment X0.1 from its buffers (while disk 1 is concurrently servicing another request). Similarly disk 2 reads fragment X1.1 at time 1, and buffers it until time interval 3 when

both X1.0 and X1.1 can be delivered.

It is interesting that it is also possible to dynamically *coalesce* the use of the disks for the fragmented request as intervening busy disks become available. Figure 5 illustrates how fragmented requests can be dynamically coalesced. Suppose at time interval 5, the 2 intervening disks have completed their service and become free. coalesced so that the disks supporting the transmission of object X are adjacent (depending on how many disks become free, a bandwidth fragmented request may be only partially coalesced). By the start of time interval 5, fragments X3.1 and X4.1 are already buffered, and have to be delivered before reading recommences. During time intervals 5 and 6, fragments X3.1 and X4.1 are delivered from buffers while fragments X3.0 and X4.0 are delivered directly from disk. Starting at time 7, the coalescing has been completed and the 2 consecutive disks pipeline the fragments directly from the disk to the network.

3 Fault Tolerance

To exhibit reasonable performance through economy of scale, the multimedia server should contain a large number of disks; something on the order of 1000 drives would not be uncommon. A large number of disks would provide the bandwidth required to service many requests simultaneously and the storage space needed to insure that the probability of fetching an object from tertiary store is small. Although a single disk can be fairly reliable, given such a large number of disks, the probability that one of them fails is quite high. For example, if the mean time to failure (MTTF) of a single disk is $100,000$ hours, then the MTTF of some disk in a 1000 disk system is on the order of 100 hours. This can be unacceptable for traditional database systems, which require a high degree of reliability and availability of data. Due to the real-time constraint, the reliability and availability requirements of a multimedia system are even more stringent. Given that a copy of the entire database resides on tertiary storage, a disk failure does not result in data loss. However, it can result in interruption of requests in progress. If some of the data for a currently displayed object is on a disk that fails, a significant degradation in performance is incurred, since the portion of objects lost due to the failure must be retrieved from tertiary storage. Therefore, without some form of fault tolerance, such a system would not be acceptable. To improve the reliability and availability of the system some fraction of the disk space must be used to store redundant information. One way to introduce redundancy into the system is to use parity based schemes which construct a parity block for every d data blocks. We assume that the reader is familiar with parity schemes [PGK88] as they are used in disk arrays, and, in the remainder of this section, discuss reliability and availability issues in that context.

As a first step in adapting the parity approach to our multimedia storage manager, we must make the following observations.

Observation 1: one should not mix data blocks of different objects in the same parity group, because constructing parity groups in this manner does not allow continuity in the delivery of data in real-time when a disk failure occurs.

To illustrate this observation, consider a parity scheme depicted in Figure 6, where, for the sake of storage and bandwidth efficiency, there is only one parity disk corresponding to all three clusters[4]. Subobjects $X0, X1$, and $Y0$ are protected by parity $P0$; subobjects $Y1, Y2$, and $Z0$ are protected

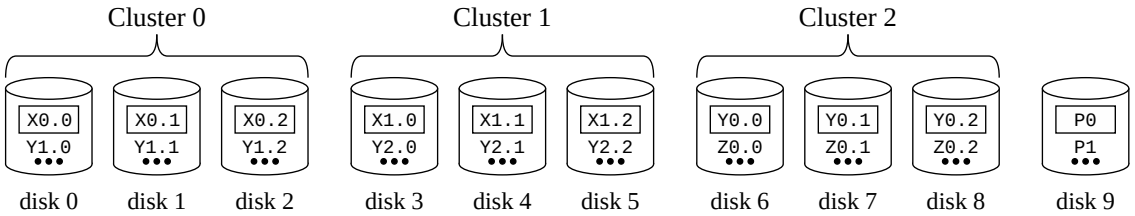


Figure 6: Invalid Parity Scheme

by parity $P1$, etc. Suppose that disk 0 fails and consequently $X0.0$ must be reconstructed. To compute the missing data, $X0.1$, $X0.2$, $X1$, $Y0$, and $P0$ must be read. If clusters 1 and 2 are busy and are scheduled to read $X1$ and $Z0$, respectively, then $X0.1, X0.2$ and $X1$ must be buffered until $Y0$ can be read from cluster 2. Clearly, this does not guarantee continuous delivery for object X, since we can not guarantee when reading of $Y0$ can be fit into the schedule. Therefore, in our system we must associate parity with *objects* rather than with *physical* disk blocks.

Observation 2: to avoid a hiccup in object delivery when a failure occurs, the first subobject in a parity group cannot be scheduled for transmission over the network until the entire parity group has been read from disk.

To illustrate this observation, consider the example parity scheme[5] of Figure 7, where: 1) there are up to three subobjects in a single parity group, 2) disk 4 has failed at time $t-1$, and 3) $X1$ is scheduled to be read at time t. It is clear that by time t, we have not read sufficient information to recover $X1.1$, i.e., parity computation is not possible until time $t+1$ when $X2$ can be read; this results in a "hiccup" in data delivery at time t. By offsetting the reading of a subobject from disks from its transmission over the network we can insure that the entire parity group is read before its

[4]Normally, in disk array systems such as RAID-5, the parity is distributed over all the disks in the system; this is done to avoid creating a bottleneck at the parity disk which can occur due to random writes. In our system, this is not a concern, since writes occur only when an object is retrieved from tertiary storage. Hence, for ease of illustration, we assume a dedicated parity disk.

[5]In this scheme, information on the failed data disk is recovered by: a) reading an appropriate fragment from the parity disks whenever the failed disk is scheduled to read a fragment and b) performing parity computation.

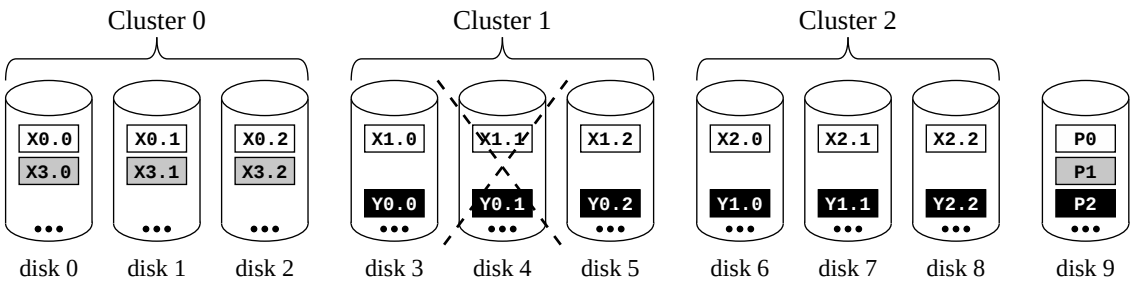


Figure 7: Multiple Clusters Per Parity Group Scheme

first subobject is scheduled for display and thus avoid hiccups. In the particular example of Figure 7, the reading of a subobject should precede its transmission by two time intervals. A consequence of this observation is that buffer space is necessary to store subobjects until the entire parity group has been read.

After making these two observations, we are ready to consider parity schemes for multimedia storage managers. It is natural to first attempt to provide fault tolerance by using a RAID system[6] [PGK88], e.g., a RAID-5. For example, assume that the D-disk storage system is a collection of RAIDs, each with $d+1$ disks, where d is the parity group size. The data layout and scheduling techniques described in Section 2 can still be used, with one exception; due to Observation 1, it is not possible to allocate fractions of a disk array's bandwidth, i.e., use the "low bandwidth objects" techniques. In other words, for each object in the system, we are forced to read the data off the disks at the rate of $\text{tfr} * d$ bytes/sec, which can be *significantly* larger than the required delivery rate of an object. The data that is read off the disks but can not (yet) be delivered to the display station must be buffered. To control buffer growth one can insert "idle" time intervals into an object's reading scheduling. For instance, if each disk array consists of $9+1$ disks and object X requires the bandwidth of a single disk, then object X can be scheduled for reading only once every 9 time intervals. During a "busy" time interval, subobjects X_i-X_{i+8} are read and X_i is delivered; during the 8 "idle" time intervals, there are no disk reads and X_{i+1}-X_{i+8} are transmitted to the display station, one per time interval.

A system that operates in this manner has the following disadvantages. Firstly, it has large buffer space requirements, i.e., up to $\frac{D*(d-1)*\text{size(fragment)}}{d+1}$ bytes of buffer space for a storage system with D disks. Secondly, it has a large bandwidth overhead, i.e., one out of every $d+1$ disks is not used for delivery of objects under normal operation. Trying to reduce the bandwidth overhead by increasing the parity group size, increases the buffer space requirements; trying to

[6]Not that by using RAIDs, we impose the following constraints on the system: 1) parity groups can not cross disk array boundaries and 2) all objects must use the same parity group size.

reduce the buffer space requirements by decreasing the parity group size improves reliability but increases the bandwidth overhead. Lastly, assuming multiple object types, scheduling and data layout algorithms in this system are more complex than those described in Section 2. Since the rate at which data is read is dictated by the value of d, it is the same for all objects but their bandwidth requirements are not. Thus the ratio of busy to idle time intervals is different for each type of object. For example, consider object Y whose bandwidth requirements can be satisfied by three disks[7]. Given the example described above, reading of object Y would alternate between 1 busy time interval and 2 idle ones; at the same time object X might have to alternate between 1 busy time interval and 8 idle intervals. In view of the disadvantages, alternatives to the RAID-5 scheme should be considered.

A variety of parity-based fault tolerance schemes and a detailed discussion on design tradeoffs can be found in [BGM94]. Several redundancy schemes are presented for providing reliability and availability in a multimedia on-demand server and their tradeoffs with respect to storage overhead, wasted bandwidth and buffer space. Our goal in this section is to describe one scheme for providing high reliability and availability for multimedia servers and to illustrate the design tradeoffs associated with building a fault tolerant storage manager.

How much redundant information we store and how we place it on the disks will determine: a) the fraction of storage available for "real" data, b) the fraction of disk bandwidth available for delivering this data, and c) the resiliency of the system to disk failure, i.e., how many failures can the system withstand before degradation of service occurs or data is lost[8]. *Degradation of service* denotes the situation in which the system is unable to continue servicing all requests that are active when the failure occurs. This may be necessary due to a lack of available bandwidth (as opposed to data loss). In this situation, one or more requests have to be rescheduled at a later time. In addition, when designing a redundancy scheme, one should also consider how system behavior changes: a) due to disk failure and b) during the rebuild process (i.e., when rebuilding a failed disk on a spare one). In general, the more redundant information is stored, the lower is the probability that a failure results in data loss and a subsequent access of tertiary storage. On the other hand, the less redundant information is stored, the more space is available for storing objects and possibly the more bandwidth is available for displaying them. Therefore improvements in reliability must be balanced against degradation in performance.

Due to space limitations we describe only one of the schemes from [BGM94]. Consider the parity

[7] Bandwidth requirements that are not divisors of 9 would complicate matters further.

[8] The data is not really lost, since there is a copy on tertiary storage, but part of an object that was in disk storage is no longer there.

scheme illustrated in Figure 8, where each parity group is constructed using multiple consecutive[9] subobjects. Each parity fragment is placed on the disk immediately following the group of disks holding the data fragments of that parity group. For instance, subobjects $X0$ and $X1$ make up one parity group and reside on disks 0-5; the corresponding parity fragment, $X01.p$ resides on disk 6. The stride in this scheme must be large enough to insure that data fragments belonging to the

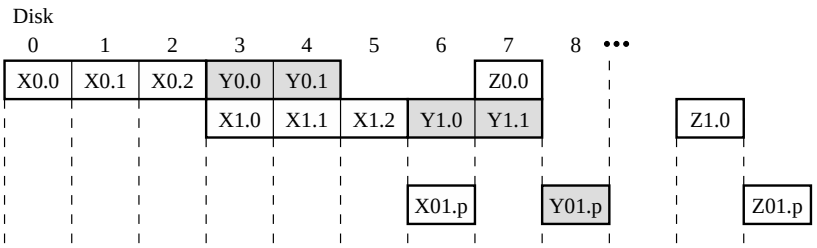


Figure 8: Large Stride Scheme

same parity group do not share disks. If this condition is not satisfied, one disk failure could result in the loss of several fragments from the same parity group; in this case, even a single failure will lead to data loss.

Under failure it is necessary make up for the missing data fragment by reading an appropriate parity fragment. For instance, if in Figure 8 $X1$ is scheduled for reading at time t and disk 5 is down, then instead of reading $X1.2$ from disk 5, we should read $X01.p$ from disk 6. If $Y1$ is also scheduled to be read at time t, then it's data fragment residing on disk 6, $Y1.0$, becomes inaccessible (i.e., from the point of view of object Y, disk 6 has effectively failed), and hence it also must perform a "shift to the right" by reading $Y01.p$ from disk 8 instead. This shift must propagate until an idle disk is encountered; if there is no idle disk in the system, then one of the active requests must be dropped, since there is insufficient bandwidth to continue servicing all active requests. Specific rules for performing the "shift to the right" are given in [BGM94]. Due to space limitations we omit these rules from this discussion but rather illustrate them through the following example. Consider again the system of Figure 8 when, for instance, disk 1 fails before time t. In that case, X and Y can still be delivered as follows (reading and delivery of Z are not shown since it is not affected by the failure):

time t: read $X0.0, X0.2$, and $X1.0$
read $Y0.1$ and $Y1.0$

[9] We would like these to be consecutive due to buffering considerations mentioned in the discussion on Observation 2; the less time it takes to read an entire parity group, the smaller are the buffer space requirements.

time $t+1$: read $X1.1$ and $X1.2$
do *not* read $X1.0$
read $X01.p$
read $Y1.1$
do *not* read $Y1.0$
read $Y01.p$

As already mentioned in Observation 2, this scheme can suffer from hiccups when a failure occurs, because an entire parity group is not read in a single time interval. Again, this problem can be solved by (sufficiently) offsetting the delivery of an object to insure that the entire parity group is read before its first subobject is delivered; this results in a need for buffer space, which is a function of the parity group size and the number of time intervals required to read a parity group.

Note that this parity scheme has several important advantages over the RAID scheme, discussed earlier in this section. It is flexible enough to allow different parity group sizes, because with this scheme it is neither necessary to have a constant number of fragments nor a constant number of subobjects in a parity group; recall, that in the RAID scheme it was necessary to have a constant number of data fragments in a parity group. Thus, decisions about parity group sizes can be made on per object basis, which allows better control over buffer space requirements as well as storage and bandwidth overhead. In addition, this scheme is able to use all available bandwidth during normal operation (discarding the bandwidth reduction due to seeks, latency, etc.). This scheme, as well as other approaches, and a discussion on design tradeoffs and costs for providing high degrees of reliability and availability in multimedia storage managers can be found in [BGM94].

4 Materialization

When a request references a non-disk resident object, the system may elect to materialize the object on the disk drives before initiating its display. The ratio between the production rate of the tertiary store, $B_{Tertiary}$, and the consumption rate at a display station, $B_{Display}$, is termed **Production Consumption Ratio** ($PCR(X) = \frac{B_{Tertiary}}{B_{Display}(X)}$). We assume that memory serves as an intermediate staging area between the tertiary and the disk-clusters. The number of disk drives allocated to the object materialization dictates the consumption rate of disk drives for object X.

The alternative cases to consider are $PCR = 1$, $PCR < 1$, and $PCR > 1$. For each case, in [GDS94a] we describe alternative materialization techniques. For each technique we described: (1) the physical layout of the object on the tertiary storage medium, (2) the algorithm that computes the time required to materialize an object from tertiary onto disks, $T_{Materialize}$, and the number of memory fragments required during each time interval. When $PCR = 1$, the tertiary store produces

a single subobject per time interval. Thus, the layout of an object on tertiary is sequential and its materialization time is equivalent to its display time. In this section, we describe the object materialization when $PCR < 1$ (and refer the reader to [GDS94a] for $PCR > 1$ due to lack of space).

With $PCR < 1$, the tertiary cannot produce an entire subobject during each time interval. Instead, it produces a fraction of a subobject: $PCR \times size(subobject)$. If an object is stored in a sequential manner on the tertiary storage device, then there will be a mismatch between the order in which data is read from the tertiary storage device and the order in which it is to be written to the disks: recall from Section 2 that the organization of an object on the disk drives is not sequential. Consequently, when materializing object X, the tertiary store produces $PCR(X)$ of X_0 during the first time interval. During the second time interval, if the system forces the materialization process to shift k disks to the right (in a round-robin manner, similar to the display of an object) then the tertiary store would be required to skip some data and produce $PCR(X)$ of X_1. This would require the tertiary storage device to reposition its read head, incurring overhead that might not be acceptable. Alternatively, the system can treat the materialization process as a non-real-time activity and allow it to 1) continue to read X_0 during the second time interval avoiding a tertiary seek, 2) flush the fragments to the individual disks selectively based on the availability of resources during a time interval. In the worst case, the amount of memory required by this approach is $\Theta(M_X \times D)$ memory fragments, for the duration of object materialization. This memory requirement can be minimized by flushing a portion of a fragment as it becomes memory resident depending on the availability of the disk drives that should contain it. This process can further be optimized to flush portions of different fragments during a single time interval, as long as its total service time does not exceed the duration of a single time interval. Assuming that an object consists of n subobjects, in the worst case, the number of time intervals required to materialize an object is $\lceil \frac{n}{PCR} \rceil + \frac{D \times M_X}{T}$ time intervals (assuming T is the average number of disk drives employed during each time interval). In [GDS94a], we devise as algorithm that computes the exact number of memory fragments required per each time interval and the number of time intervals required to complete the object materialization.

An alternative approach is to write the data on the tape in the same order as it is expected to be delivered to the disk drives and treat the materialization process similar to a display that requires $\frac{B_{Tertiary}}{B_{Disk}}$ disk drives. With this approach, the physical layout of an object on tertiary is dependent on the bandwidth required to display an object, as well as the bandwidth of both disk and tertiary. To illustrate, assume that object X requires a 60 mbps (see Figure 3) bandwidth and the disk bandwidth is 20 mbps ($M_X = 3$). If the bandwidth of the tertiary storage is 40 mbps, then the fragments of

X could be stored on tertiary based on the organization of the fragments across the disk drives in the following manner: $X_{0.0}, X_{0.1}, X_{1.0}, X_{1.1}, X_{2.0}, X_{2.1}, ..., X_{n-1.0}, X_{n-1.1}, X_{0.2}, X_{1.2}, X_{2.2}, ..., X_{n-1.2}$. During the first time interval the first two fragments of subobject X_0 are read from tertiary and flushed onto the disk drives ($X_{0.0}, X_{0.1}$), while during the second time interval the first two fragments of subobject X_1 are flushed to the disk drives ($X_{1.0}, X_{1.1}$), etc. This approach ensures that the expected fragments are rendered disk resident during each time interval, in most part. However, since the object layout does not take into account the number of disk drives in the system (D), there will still be some mismatch. In the worst case, this mismatch requires the use of $T \times D$ (assuming T is the number of disk drives dedicated per time interval) memory frames as a staging area in the worst case. Thus, in the worst case, the number of time intervals required is $\lceil \frac{n}{PCR} \rceil + D$. However, note that this causes the layout of an object to be dependent on B_{Disk}, $B_{Display}$, and $B_{Tertiary}$. This dependency might be a limitation due to the current advancements in the tertiary bandwidth [Sch93]. Every time that the bandwidth of the tertiary store increases and the system is upgraded, the data on tertiary needs to be re-recorded (re-organization of data).

5 Pipelining

With a hierarchical storage organization, when a request references an object that is not disk resident, one approach might materialize the object on the disk drives in its entirety before initiating its display. In this case, the latency time of the system is determined by the bandwidth of the tertiary storage device and the size of the referenced object. With continuous media (e.g., audio, video) that require a sequential retrieval to support its display, a better alternative is to use a pipelining mechanism ([GDS94b]) that overlaps the display of an object with its materialization, in order to minimize the latency time.

With pipelining, a portion of the time required to materialize X can be overlapped with its display. This is achieved by grouping the subobjects of X into s logical slices ($S_{X,1}$, $S_{X,2}$, $S_{X,3}$, ..., $S_{X,s}$), such that $T_{Display}(S_{X,1})$ eclipses $T_{Materialize}(S_{X,2})$, $T_{Display}(S_{X,2})$ eclipses $T_{Materialize}(S_{X,3})$, etc. Thus:

$$T_{Display}(S_{X,i}) \geq T_{Materialize}(S_{X,i+1}) \; for \; 1 \; \leq \; i \; < \; s \tag{2}$$

Upon the retrieval of a tertiary resident object X, the pipelining mechanism is as follows:

1: Materialize the subobject(s) that constitute $S_{X,1}$ on the disk drive.

2: For $i = 2$ to s do

   a. Initiate the materialization of $S_{X,i}$ from tertiary onto the disk.

   b. Initiate the display of $S_{X,i-1}$.

Figure 9: The pipelining mechanism

3: Display the last slice ($S_{X,s}$).

The duration of Step 1 determines the latency time of the system. Its duration is equivalent to $T_{Materialize}(X) - T_{Display}(X)+$ (one Time Interval). Step 3 displays the last slice materialized on the disk drive. In order to minimize the latency time[10], $S_{X,s}$ should consist of a single subobject. To illustrate this, assume $PCR < 1$ and consider Figure 9. If the last slice consists of more than one subobject then the duration of the overlap is minimized, resulting in a longer duration of time for Step 1.

6 Management of the Disk Space

The staggered striping technique is scalable and appropriate for both a high end system that consist of thousands of disk drives and service many users and a low end system that services a single user at a time (e.g., a workstation that consists of several disk drives and a tertiary storage device [Com94]). However, a disk space management technique appropriate for a single user system may not be appropriate for a multi user system and vice versa. We have investigated each environment and, in the following, provide a summary of our proposed techniques and research results for each case.

The formal statement of the problem is as follows. Assume that the database consists of m objects $\{\vee_1, \ldots, \vee_m\}$, with a predefined frequency of access denoted as $\text{HEAT}(\vee_j) \in (0,1)$ satisfying $\sum_{j=1}^{n} \text{HEAT}(\vee_j) = 1$, and sizes $\text{SIZE}(\vee_j) \in (0, C)$ for all $1 \leq j \leq n$. The size of the database exceeds the storage capacity of the disks (*i.e.*, $\sum_{j=1}^{n} \text{SIZE}(\vee_j) > C$ where C is the storage capacity of the disks). Assume a process that generates requests for objects in which object $\vee_j$ is requested with

[10] Maximize the length of the pipeline.

probability HEAT($\vee_j$) (all independent). We assume no advance knowledge of the possible permutation of requests for different objects. Once the storage capacity of the disk drives is exhausted, in the presence of a request referencing a tertiary resident object, the system should replace one or more objects to enable the referenced object to become disk resident. We have investigated three alternative replacement policies. The first replaces those objects with the lowest heat in their entirety (termed HBased). Assuming no dependencies between requests referencing objects, HBased approximates the LRU algorithm [OOW93]. The HEAT of an object is estimated by observing the requests referencing that object for a fixed duration of time. Hence, when the heat of a disk resident object decreases, HBased replaces it by an object that has a higher heat. An alternative approach is a *cost based* approach (termed CBased). CBased is identical to HBased, except that it chooses its victims based on both their *heat* and *size*, i.e., *cost*(X) = HEAT(X) $\times$ SIZE(X).

The third approach is a PartIal ReplAcement TEchnique (PIRATE) [GS94]. PIRATE replaces an object at the granularity of a subobject. Therefore, a number of subobjects that constitute an object X might be disk resident (denoted DISK(X)), while its remaining subobjects might be absent, termed ABSENT(X); ABSENT(X) = SIZE(X) $-$ DISK(X). (Note: the granularity of ABSENT(X), SIZE(X), and DISK(X) are in terms of subobjects.) PIRATE also deletes objects based on their HEAT, and similar to HBased, is a dynamic replacement policy.

The results of our analytical evaluations and simulation studies demonstrate that PIRATE is a superior technique in a single user environment as compared to both HBased and CBased. However, in the presence of multiple users, HBased outperforms both PIRATE and CBased.

6.1 Single User

We describe PIRATE in the context of pipelining. Assume a reference to object X. If the first slice of X ($S_{X,1}$) is disk resident, then the maximum latency time observed by a request referencing X is approximately zero (the worst seek and latency time). Otherwise, the user is required to wait for tertiary to materialize $S_{X,1}$ before the display of X can be initiated. Keeping the first slice of the frequently accessed objects disk resident and deleting the remainder of the object minimizes the average latency time observed by a single user. This explains the superiority of PIRATE to HBased (or any other approach) that maintains an object disk resident in its entirety. Note that it is possible that the available disk space is exhausted after storing the first slice of several objects. In this case if a new request arrives for an object whose first slice is not disk resident, the system must delete from the disk resident slice of other objects. There are two alternative approaches:

1. **Simple PIRATE** deletes the first slice of those objects with the lowest heat (following the greedy strategy suggested for *fractional knapsack problem* [CLR90]). **Moreover, it frees up only sufficient space to accommodate the pending request and no more than that.** (For example, if ABSENT(Z) is equivalent to $\frac{S_{X,1}}{10}$ and X is chosen as the victim then

Figure 10: Status of the **first slice** of objects

only the subobjects corresponding to the last $\frac{1}{10}$ of $S_{X,1}$ are deleted in order to render Z disk resident[11].) One might argue that a combination of HEAT and SIZE (including CBased) should be considered when choosing victims. In [GS94] we demonstrated the invalidity of this argument and proved the optimality of simple PIRATE to minimize the latency time of the system.

2. **Extended PIRATE** follows the same strategy as simple PIRATE, except that it maintains a minimum portion of an object (LEAST(X)) disk resident. In [GS94] we computed LEAST(X) as the amount of disk space an object X deserves: $\frac{\text{HEAT}(X)\times\text{SIZE}(S_{X,1})}{\sum_i^m \text{HEAT}(i)\times\text{SIZE}(S_{i,1})}\times C$ where m is the total number of objects in the database. Hence, it stops deleting X when DISK(X) $\leq$ LEAST(X), and it chooses another victim (say Y). As discussed above, if HEAT(Y) $>$ HEAT(X), keeping a portion of X disk resident at the expense of deleting from Y, results in a higher average latency time. Extended PIRATE degrades the average latency time proportional to the improvement in variance. In experiments conducted we used *Chebyshev formula* to demonstrate that extended PIRATE reduces the probability of a request observing a high latency time by at least 20% (and up to 40%) as compared to simple PIRATE.

To compare simple PIRATE with extended PIRATE consider Figure 10. In this figure, each vertical box corresponds to the first slice of an object. The shading represents the disk resident fraction of each slice. Simple PIRATE maintains the **first slice** of those objects with the highest heat ($\aleph_{simple}$) disk resident causing the other objects to compete for the remaining portion of the disk space (see Figure 10.a). However, extended PIRATE keeps a portion (LEAST(X)) of the first slice of the hot objects disk resident (see Figure 10.b). Therefore, in the long run the number of

[11] HBased deletes X in its entirety to render Z disk resident.

unique objects that have a fraction disk resident is higher with extended PIRATE ($\aleph_{extended}$) as compared to simple PIRATE ($\aleph_{simple}$). This results in the following trade off: On one hand, a request referencing an object Z has a higher probability to observe a *hit* with extended PIRATE as compared to simple PIRATE. On the other hand, a *hit* with simple PIRATE translates into a fraction of a second latency time, while with extended PIRATE, it results in a latency time of: $(\text{SIZE}(S_{Z,1}) - \text{LEAST}(Z)) \times \frac{size(subobject)}{B_{Tertiary}}$. This explains why extended PIRATE results in a longer average latency while improving the variance proportionally.

6.2 Multiple User

In the presence of multiple users, HBased outperforms PIRATE. This is because:

Cluster Availability: In staggered striping, assume a request arrives referencing Z. With HBased $\text{DISK}(Z) = 0$, hence Z can be materialized starting with a first **available** cluster. However, with PIRATE, if $0 < \text{DISK}(Z) \leq \text{SIZE}(Z)$, then the remainder of Z should be materialized starting with a specific disk cluster (C_i) to support the organization required by staggered striping. C_i might be busy servicing other requests. This results in a higher latency time because the materialization process is forced to wait until C_i becomes available. However, this latency in the worst case (heavy loaded system) is $R-1$ time intervals. If $\text{SIZE}(Z) >> R$ (the granularity of $\text{SIZE}(Z)$ is in subobjects), the latency in the worst case is negligible as compared to the display time of an object.

Tertiary Bottleneck: PIRATE maintains the first slice of the frequently accessed objects disk resident while HBased maintains a fewer number of frequently accessed objects disk resident in their entirety. This causes PIRATE to generate more requests to the tertiary where each request transfers data for a short duration of time. In the presence of multiple users, this may result in the formation of a queue at the tertiary. Although each access employs tertiary for a shorter duration of time, the requests in the end of the queue still observe the summation of service time of the requests to be serviced by tertiary (tertiary becomes the bottleneck). Moreover, the physical characteristics of tertiary devices dictate that they are more efficient in the presence of few requests where each request retrieves a large object in a sequential manner as compared to servicing many requests where each request retrieves a small portion of a different object sequentially. This is due to large seek time associated with these devices.

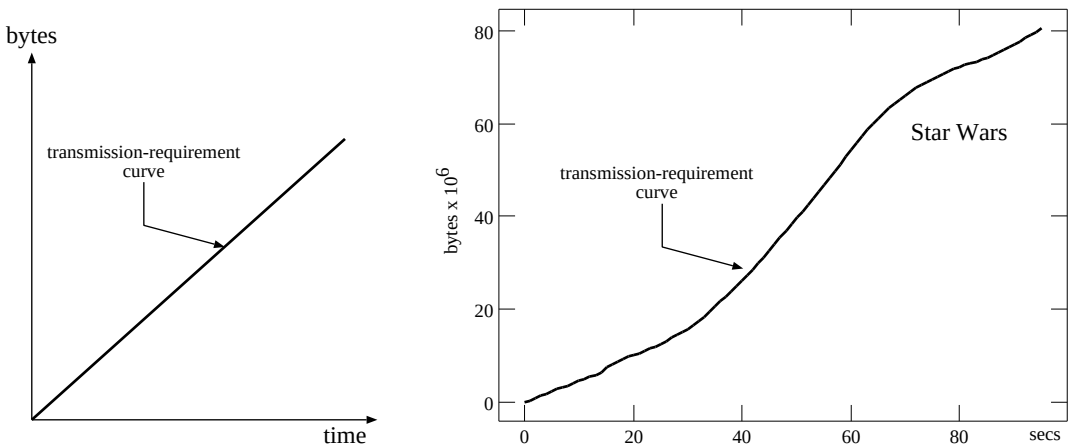
Comparing HBased with CBased, although both take advantage of the **Cluster Availability**, CBased suffers from **Tertiary Bottleneck**. This is because CBased might delete a smaller object with a higher heat (say X), instead of a larger object with a lower heat (say Y). While this decreases the duration of time the tertiary will be employed in future (because $\text{SIZE}(X) < \text{SIZE}(Y)$), the probability of future access to tertiary is increased (because $\text{HEAT}(X) > \text{HEAT}(Y)$). As discussed in the previous paragraph, this is not a good trade off given the physical characteristics of the current

tertiary storage devices.

If the target platform consists of several tertiary storage devices (say t), the performance of PIRATE might improve. Let A_{HBased} (A_{PIRATE}) be the number of requests competing simultaneously for the tertiary storage device with HBased (PIRATE). From the previous discussion we know $A_{HBased} < A_{PIRATE}$. If $A_{HBased} < t = A_{PIRATE}$, PIRATE can utilize all the tertiary devices while $t - A_{HBased}$ tertiary devices will sit idle with HBased. However, if $t < A_{HBased} < A_{PIRATE}$, the probability that tertiary devices become the bottleneck is higher with PIRATE as compared to HBased. Therefore, there is a trade off based on the average number of requests competing for the tertiary simultaneously and the number of available tertiary devices.

7 Compression

Compression techniques are an effective means of reducing both the size and the disk bandwidth requirements of continuous media data types. For example, using MPEG compression techniques [Gal91]; a typical (MPEG-I) compressed 100 minute movie requires on the order of 1.25GB of storage and (on the average) a 1.5 mbps transfer rate (MPEG-I results in approximately 1:26 ratio between compressed and uncompressed video). A side effect of compression is that it results in variant disk bandwidth requirements (as compared to a fixed bandwidth with uncompressed objects). To illustrate, Figure 11 depicts the number of bytes that must be retrieved for a video object (Star Wars) in both uncompressed and compressed form. In compressed form (see Figure 11(b)), the rate at which data must be delivered for an object varies with time.



(a) Constant BW-Req per Object (uncompressed data) (b) Variable BW-Req per Object (compressed data)

Figure 11: Object Bandwidth Requirements

Scheduling schemes in Section 2 rely on the "regularity" of the bandwidth requirement of each object. We can reap the benefits of the staggered striping method by extending it to work with

compressed data. If the data is not stored in compressed form, then the disk transmission rate (as well as the rate of the processor and network transmission) is simply matched to the delivery requirements of an object (see Section 2). On the other hand, if the data is stored in compressed form, then the disk transmission characteristic does not have to match exactly the characteristic of an object's delivery requirements, as long as the disks transmit at a sufficiently high rate to stay ahead of the transmission requirement. This is illustrated in Figure 12, where the disk transmission rate only "roughly" matches the object's delivery requirement; the data that has been transmitted by the disks but is not yet due at a display station, is buffered in memory. We assume that

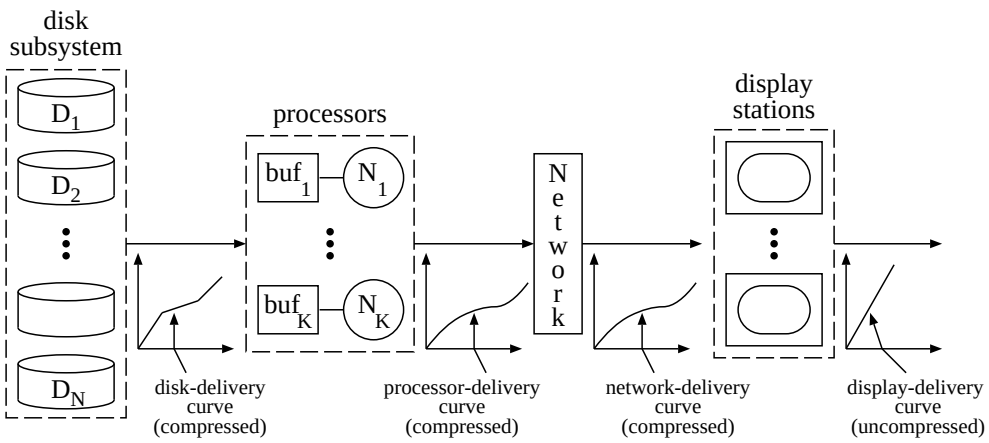


Figure 12: Transmission of Data

the data is transmitted through the network in compressed form to reduce the network bandwidth requirements and is only decompressed at the display station; in addition, we assume that a display station performs the decompression, and that it is not capable of buffering more than a few frames, i.e., any buffering required by the scheduling schemes must be provided by the storage manager. In this section, we focus on the data layout and the corresponding scheduling schemes for *disk* transmission, i.e., on the first step in Figure 12.

A taxonomy of schemes, presented in [EMGGM94] for extending the staggered striping method to handle delivery of compressed data, is shown in Figure 13. The two basic categories of schemes that can be used with compressed data are:

- *constant-rate* schemes (illustrated in Figure 14(a)), where *each* object is delivered at a constant rate, although the rate may vary between objects, and
- *variable-rate* schemes (illustrated in Figure 14(b)), where each object can have a different bandwidth requirement in each time interval.

Each scheme discussed in [EMGGM94] differs in its construction of the "disk-delivery" curve, i.e., the curve that represents the rate at which the data will actually be transmitted from the disks.

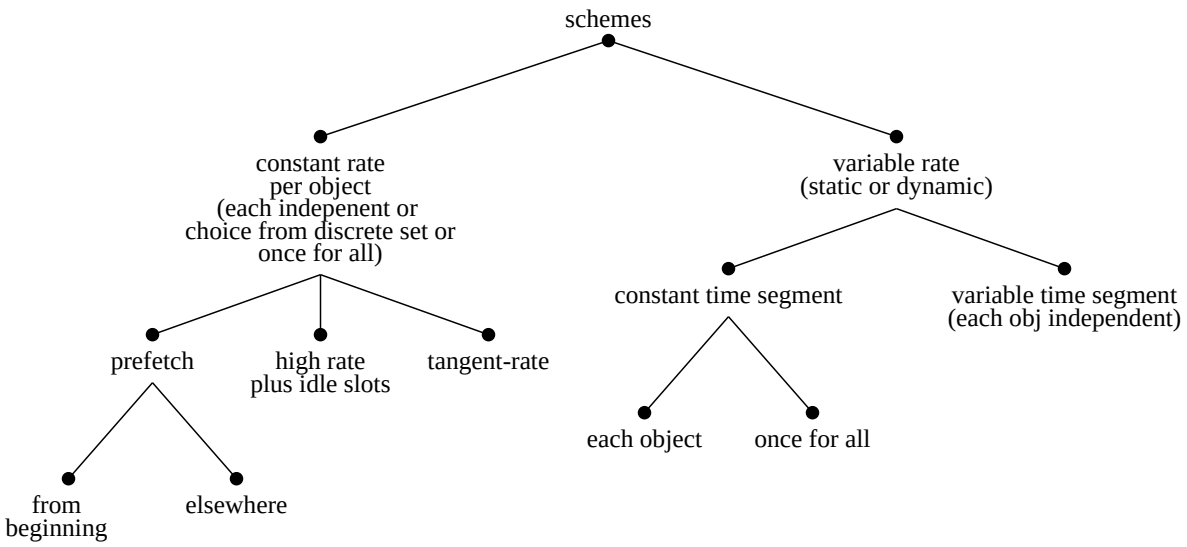


Figure 13: Taxonomy of Schemes

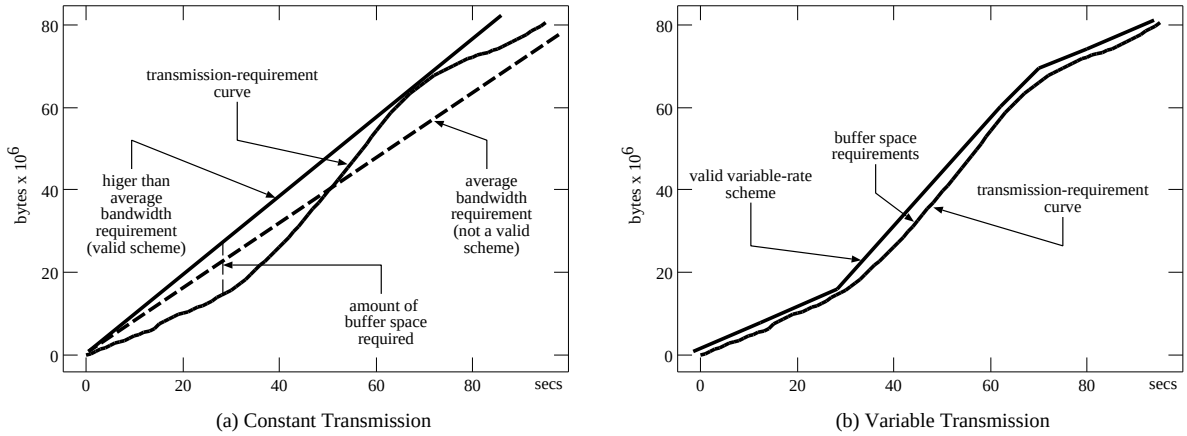


Figure 14: Disk Transmission

(This curve determines the bandwidth requirements, buffer space requirements, etc., discussed in the remainder of this section.) To insure the satisfaction of the real-time constraint, it is necessary for the "disk-delivery" curve to either coincide with or stay above the "transmission-requirement" curve. The difference between the required bandwidth and the bandwidth delivered by the disks results in buffer space requirements. The closer is the fit between the two curves, the smaller are the buffering requirements.

An advantage of using a constant-rate scheme is that we can make use of the *simple* scheduling and data layout techniques presented in Section 2, without modifications. A disadvantage of such a scheme is that it can result in high buffer space requirements, if the highest transmission rate required by an object is not representative of the rest of the transmission-requirement curve. Large

buffer space requirements could result in a restriction on the number of requests that can be serviced simultaneously, even when the bandwidth to service these requests is available, i.e., the buffer space limitations, rather than the maximum bandwidth of the system, might prove to be the bottleneck.

A variable-rate scheme would help remedy the buffering problem, since it gives more control over the buffer usage, as shown in Figure 14(b) where a piecewise-linear curve is constructed to better fit the bandwidth requirements of the object. However, such a scheme results in significant complications in the scheduling algorithms. In order to schedule a new request with a variable-rate scheme, it is not sufficient to consider the current bandwidth availability of the system. Instead, the availability of the necessary bandwidth for the entire duration of an object's display must be taken into consideration in order to decide where a new request "fits" into the current schedule. This can (potentially) increase the latency for starting the service of a request and result in disk bandwidth fragmentation problems, i.e., in wasted bandwidth due to inability to use the full bandwidth capacity of the system as a result of future, rather than current, bandwidth requirements of an object. In this case, we must balance a closer curve fit, which can potentially make the scheduling and fragmentation problems more complicated, against a rougher fit, with simpler scheduling but larger buffering requirements.

The following metrics are used in comparing schemes:

- *buffer space*: amount of buffer space required by the scheme, which is a function of how closely the "disk-delivery" curve matches the "transmission-requirement" curve
- *bandwidth*: amount of additional bandwidth, i.e., in addition to the transmission requirements, that is used by the scheme (this is also a function of how closely the "disk-delivery" curve matches the "transmission-requirement" curve)
- *variability of bandwidth requirements*: the number of different bandwidths that an object requires throughout the duration of its display

Note that in comparing schemes, it is important to consider not just the maximum amount of buffer space that a particular schemes requires, but also the *duration* for which these buffers are required, e.g., some schemes might require a lot of buffer space, but only at the beginning of an object's display. A scheme that occupies buffers for a long time, in effect has relatively high buffer space requirements. Similarly, it is important to consider not just the amount of additional bandwidth required by a scheme, but also the duration for which it is required, e.g., some schemes might require more bandwidth but for a shorter duration of time.

The purpose of considering the above "scheme-comparison" metrics is to try to predict how the various schemes will affect the performance of the multimedia server. The important performance characteristics to consider are as follows:

- *latency*: the delay for starting the service of an object
- *throughput*: the number of simultaneous displays that can be supported by the system. A scheme that either requires substantial amount of memory for a long duration of time or results in a high degree of disk bandwidth fragmentation (due to scheduling constraints) may limit the throughput of the system.

We intend to construct simulation models to investigate the alternative schemes further.

8 Related Work

The physical design of a storage manager that can support continuous media data types has been investigated in several earlier studies, including [AH91, RVR92, RV93, CL93, TPBG93, RW94]. All studies assume that the bandwidth required to display an object is lower than the bandwidth of a single disk drive. These studies investigate either disk scheduling [RW94, TPBG93] or the placement of data across the surface of a disk drive [CL93, RV93] to enhance the performance of the disk subsystem. They develop admission control policies that enable a system to guarantee a continuous display for each active request. While [TPBG93] assumes a RAID [Pat93] as its target hardware platform, the remaining studies assume an architecture that consists of a single disk drive and some memory. (Even in [TPBG93] the number of disks considered is no more than 4.) The design of the storage manager described in this paper is novel because it scales to large number of disks, employs a hierarchical storage manager and can support the display of those objects whose bandwidth exceeds the bandwidth of a single disk drive.

Of particular relevance to this paper is [TPBG93] because it assumes a multi-disk array in the target architecture. Briefly, assuming that the bandwidth required to display an object is fixed and lower than the bandwidth of a single disk drive, [TPBG93] ensures a continuous retrieval of multimedia objects by scheduling the disk I/O in cycles. In each cycle, each active object requests a fixed size fragment from each of the N_d disks. (Of course the object has been striped across the disks so that the next N_d fragments to be delivered are spread across the disks.) Using this scheme the load is evenly distributed over the disks. By choosing the size of the I/O requests and limiting the number of active objects, it can be guaranteed (to within some probability) that data for all active requests is delivered from the disks at the required rate. The paper does not report on systems greater than 4 disks, and even in these small systems the amount of memory required grows quite rapidly with the segment size; and one wants a large segment size to get most of the available disk bandwidth. The reason is that the size of the segment that is read from each disk for an object during each cycle, determines the effective disk bandwidth achieved and also the amount of buffer memory required.

Staggered striping is a superior alternative because it: 1) is scalable as it uses a fixed subset

of disk drives during each time interval (causing the fraction of wasted disk bandwidth to remain unchanged as a function of additional disk drives), 2) uses declustering to establish a multi-input pipeline between the activated disk drives and a display station in order to minimize the memory required to support a display.

9 Conclusion and Future Research Directions

In this paper, we provided an overview of techniques to construct a scalable multimedia storage manager. Its target hardware platform is hierarchical, consisting of memory, multiple disk drives and tertiary storage devices. We have conducted simulation studies to evaluate staggered striping and the alternative disk management policies [GS94, BGMJ94], and intend to extend these studies to analyze the alternative materialization techniques and the schemes proposed to support the variable disk bandwidth requirement of compressed objects. The latter entails an analysis of the characteristics of data expected to be stored in a multimedia storage manager (e.g., bandwidth requirement of compressed movies from a video server). Once these studies are complete, we intend to undertake an implementation of a system to demonstrate the feasibility of our techniques (and address any hidden limitations).

References

[AH91] D. Anderson and G. Homsy. A cotinuous media I/O server and its synchronization. *IEEE Computer*, October 1991.

[BGM94] S. Berson, L. Golubchik, and R. R. Muntz. A Fault Tolerant Design of a Multimedia Server. Technical Report CSD-940009, UCLA, February 1994.

[BGMJ94] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered Striping in Multimedia Information Systems. In *Proceedings of ACM SIGMOD*, 1994.

[CL93] H.J. Chen and T. Little. Physical Storage Organizations for Time-Dependent Multimedia Data. In *Proceedings of the Foundations of Data Organization and Algorithms (FODO) Conference*, October 1993.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, editors. *Introduction to Algorithms.* The MIT Press, and McGraw-Hill Book Company, 1990.

[Com94] R. Comerford. The Multimedia Drive. *IEEE Spectrum*, 31(4):77–84, April 1994.

[CP94] D. K. Campbell and K. Proehl. Optical Advances. *BYTE Magazine*, March 1994.

[Doz92] J. Dozier. Access to data in NASA's Earth observing system (Keynote Address). In *Proceedings of ACM SIGMOD*, June 1992.

[EMGGM94] M. L. Escobar-Molano, S. Ghandeharizadeh, L. Golubchik, and R. R. Muntz. A framework for the study of delivery of compressed data in multimed ia information systems. Technical report, UCLA, 1994.

[Fox91] E. A. Fox. Advances in Interactive Digital Multimedia Sytems. *IEEE Computer*, pages 9–21, October 1991.

[Gal91] D. Le Gall. MPEG: a video compression standard for multimedia applications. *Communications of the ACM*, April 1991.

[GD90] S. Ghandeharizadeh and D. DeWitt. A multiuser performance analysis of alternative declustering strategies. In *Proceedings of International Conference on Database Engineering*, 1990.

[GDS94a] S. Ghandeharizadeh, A. Dashti, and C. Shahabi. Object Materialization with Staggered Striping. Technical report, University of Southern California, 1994.

[GDS94b] S. Ghandeharizadeh, A. Dashti, and C. Shahabi. A Pipelining mechanism to minimize the latency time in hierarchical multimedia storage managers. Technical report, University of Southern California, 1994.

[GR93] S. Ghandeharizadeh and L. Ramos. Continuous retrieval of multimedia data using parallelism. *IEEE Transactions on Knowledge and Data Engineering*, 1(2), August 1993.

[GRAQ91] S. Ghandeharizadeh, L. Ramos, Z. Asad, and W. Qureshi. Object Placement in Parallel Hypermedia Systems. In *Proceedings of Very Large Databases*, 1991.

[GS93] S. Ghandeharizadeh and C. Shahabi. Management of Physical Replicas in Parallel Multimedia Information Systems. In *Proceedings of the Foundations of Data Organization and Algorithms (FODO) Conference*, October 1993.

[GS94] S. Ghandeharizadeh and C. Shahabi. On Multimedia Repositories, Personal Computers, and Hierarchical Storage Systems. In *Submitted to ACM Multimedia*, 1994.

[Has89] B. Haskell. International standards activities in image data compression. In *Proceedings of Scientific Data Compression Workshop*, pages 439–449, 1989. NASA conference Pub 3025, NASA Office of Management, Scientific and technical information division.

[Joh93] C. Johnson. Architectural Constructs of AMPEX DST. *Third NASA GSFC Conference on Mass Storage Systems and Technologies*, 1993.

[LKB87] M. Livny, S. Khoshafian, and H. Boral. Multi-Disk Management Algorithms. In *Proceedings of the 1987 ACM SIGMETRICS Int'l Conf. on Measurement and Modeling of Computer Systems*, May 1987.

[OOW93] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K Page Replacement Algorithm for Database Disk Buffering. In *Proceedings of ACM SIGMOD*, pages 413–417, 1993.

[Pat93] D. Patterson. Massive Parallelism and Massive Storage: Trends and Predictions for 1995 to 2000 (Keynote Speaker). In *Second International Conference on Parallel and Distributed Information Systems*, January 1993.

[PGK88] D. Patterson, G. Gibson, and R. Katz. A case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of ACM SIGMOD*, May 1988.

[RE78] D. Ries and R. Epstein. Evaluation of distribution criteria for distributed database systems. UCB/ERL Technical Report M78/22, UC Berkeley, May 1978.

[RV93] P. Rangan and H. Vin. Efficient Storage Techniques for Digital Continuous Media. *IEEE Transactions on Knowledge and Data Engineering*, 5(4), August 1993.

[RVR92] P. Rangan, H. Vin, and S. Ramanathan. Designing an On-Demand Multimeida Service. *IEEE Communications Magazine*, 30(7), July 1992.

[RW94] A. L. N. Reddy and J. C. Wyllie. I/O Issues in a Multimedia System. *IEEE Computer Magazine*, 27(3), March 1994.

[Sch93] T. Schwarz. High Performance Quarter-Inch Cartridge Tape Systems. *Third NASA GSFC Conference on Mass Storage Systems and Technologies*, 1993.

[TPBG93] F.A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID-A Disk Array Management System for Video Files. In *First ACM Conference on Multimedia*, August 1993.