A Pipelining Mechanism to Minimize the Latency Time in Hierarchical Multimedia Storage Managers[*]

Shahram Ghandeharizadeh, Ali Dashti, Cyrus Shahabi

Department of Computer Science
University of Southern California
Los Angeles, California 90089

May 26, 1998

Abstract

An emerging area of database system research is to provide support for continuous media data types (digital audio and video). These data types are expected to play a major role in applications such as library information systems, scientific databases, entertainment technology, etc. They require both a high volume of storage and a high bandwidth requirement for their continuous display. The storage organization of systems that support these data types is expected to be hierarchical, consisting of one or more tertiary storage devices, several disk drives, and some memory. The database resides permanently on the tertiary storage device. The disk drives store a number of frequently accessed objects, while the memory is used to stage a small fraction of a referenced object for immediate display.

When a user references an object that is tertiary resident, if the system elects to materialize the object on the disk drives in its entirety before initiating its display then the user would observe a high latency time. This paper describes a general purpose pipelining mechanism that overlaps the display of an object with its materialization on the disk drives in order to minimize the latency time of the system. The pipelining mechanism is novel because it ensures a continuous retrieval of an object to a display station (in order to support its continuous display).

1 Introduction

During the past few years, the information technology has evolved to store and retrieve digital audio and video data types, termed continuous media data types [MWS93]. Systems that support this data type are commonly referred to as multimedia information systems. These systems utilize a variety of human senses to provide an effective means of conveying information, and play a

[*]This research was supported in part by the National Science Foundation under grants IRI-9203389, IRI-9258362 (NYI award), and CDA-9216321, a Hewlett-Packard research grant, a TRW research grant, and a research grant from AT&T/NCR/Teradata.

major role in educational applications, library information systems, etc. A challenging task when implementing these systems is to support the sustained bandwidth required to display continuous media objects [GRAQ91, SAD$^+$93, MWS93]. This is due to the low I/O bandwidth of the current disk technology, the high bandwidth requirement of continuous media data types, and the large size of their objects that almost always requires them to be disk resident. For example, a one minute uncompressed video object based on HDTV is approximately six gigabytes in size and requires 800 megabit per second (mbps) bandwidth to support its continuous display. In the presence of the low bandwidth of the current disk technology, one may employ either compression (lossless or lossy, see [Fox91] for an overview), declustering [GS93], striping [TPBG93], or a combination of these techniques [BGMJ94] to support a continuous retrieval of multimedia objects.

The storage organization of systems that support multimedia applications is expected to be hierarchical, consisting of: a tertiary storage device, a group of disk drives, and some memory [GS93, MWS93, BGMJ94]. The database resides permanently on the tertiary storage device and its objects are materialized on the disk drives on demand (and deleted from the disk drives when the disk storage capacity is exhausted). A small fraction of a referenced object is staged in memory to support its display. The reason for expecting hierarchical storage managers is the cost of storage. At the time of this writing, the approximate cost per megabyte of memory is $35, disk storage is $0.6, and tertiary storage is less than $0.1. It is economical to stage the data at the different levels of hierarchy in the following manner: a small fraction of an object in memory for immediate display, a number of frequently accessed objects on the disk drives, and the remaining objects on the tertiary storage device.

One might be tempted to replace the magnetic disk drives with the tertiary storage devices in order to reduce the cost further. This is not appropriate for the frequently referenced objects that require a fraction of a second transfer initiation delays, i.e., the time elapsed from when a device is activated until it starts to produce data. This delay is determined by the time required for a device to reposition its read head to the physical location containing the referenced data; this time is significantly longer for tertiary storage device (ranges from several seconds to minutes) as compared to that for a magnetic disk drive (ranges from 10 to 30 milliseconds). Similarly, the tertiary storage device should not be replaced by magnetic disk drives because: 1) the cost of storage increases, and 2) it might be acceptable for some applications to incur a high latency time for infrequently referenced objects.

The current technology trends in the area of tertiary storage devices is: 1) a rapid increase in both storage capacity and sustained bandwidth, 2) a rapid decline in $/megabyte of storage, and 3) a modest improvement in their head reposition time with the distance of travel determining the duration of this time [Sch93]. For example, the 8mm helical tape drives introduced in the late 1980's can store five gigabytes of data at a cost of \$0.42/megabyte and support a 4 mbps sustained transfer rate[1]. One version of this technology supports a rack of tapes to provide 60 gigabytes of storage at a cost of \$0.07/megabyte (this organization increases the time required to reposition the read head, by introducing switching time from one storage media to another, in favor of both a larger storage capacity and a lower \$/megabyte of storage). Recently, Digital introduced the DLT4000 tape drives based on the linear tape technology. It provides 20 gigabye of storage and supports a sustained transfer rate of 12 mbps at a cost less than \$0.3/megabyte. Similarly, METRUM has introduced a tertiary device based on helical scan technology (named RSS-600b). It provides 10 terabyte of storage and supports a sustained transfer rate of 16 mbps at a cost less than \$0.1/megabyte. Various tertiary devices based on Peregrine can support sustained bandwidths ranging from 76 to 253 mbps [Sch93]. The near future calls for tertiary devices that store hundreds of terabyte (if not petabyte) of data and support sustained transfer rates ranging from 80 to 800 mbps.

When a request references an object, we term the time elapsed from when the request arrives until the onset of its display as the latency time incurred by this request. With hierarchical storage organization, when a request references an object that in not disk resident, one approach might materialize the object on the disk drives in its entirety before initiating its display. In this case, assuming a zero system load, the latency time of the system is determined by: the time for the tertiary to reposition its read head to the starting address of the referenced object, the bandwidth of the tertiary storage device, and the size of the referenced object. Assuming that the referenced object is continuous media (e.g., audio, video) and requires a sequential retrieval to support its display, a superior alternative is to use pipelining in order to minimize the latency time. Briefly, the pipelining mechanism splits an object into s logical slices (S_1, S_2, S_3, ..., S_s) such that the display time of S_1 overlaps the time required to materialize S_2, the display time of S_2 overlaps the time to materialize S_3, so on and so forth. This ensures a continuous display while reducing the latency time because the system initiates the display of an object once a fraction of it (i.e., S_1) becomes disk resident.

Another advantage of pipelining is that it enhances the $useful$ utilization of resources when a user decides to abort the display of a referenced object. To illustrate, consider a user that requests an obscure (i.e., tertiary resident) 30 minute video object and decides that it is not of interest after a few minutes of display. With pipelining, the display of object is overlapped with its materialization and, once the user aborts the display, the system can abort the pipeline avoiding the tertiary from materializing the remaining slices of the referenced object (instead, tertiary can be used to service

[1]The transfer rates quoted in this paper assume a sequential read of the referenced data (no reposition times). The maximum transfer rate of tertiary devices is typically double their sustained transfer rate. Due to the continuous bandwidth requirement of video and audio objects, we use the sustained bandwidths provided by the vendors.

Figure 1: Example 1: Three alternative dataflow paradigms

some other request). Without pipelining, in addition to forcing the user to wait for a longer interval of time, the system would have had to use the tertiary for a longer interval time in order to stage the entire object on the disk drives.

The contribution of this paper is the design of[2] a general-purpose pipelining mechanism for continuous media that minimizes the latency time of the system while ensuring a continuous retrieval of the referenced object. It can be used with a tertiary storage device whose bandwidth is either equal to, higher or lower than the bandwidth required to display an object. The pipelining mechanism is described assuming an architecture that consists of some memory, several disk drives, and a tertiary storage device. We consider two alternative organization of these components: 1) memory serves as an intermediate staging area between the tertiary storage device, the disk drives and the display stations, and 2) the tertiary storage device is visible only to the disk drives via a fixed size memory. With the first organization, the system may elect to display an object from the tertiary storage device by using the memory as an intermediate staging area. With the second organization, the data must first be staged on the disk drives before it can be displayed. We capture these two organizations using three alternative paradigms for the flow of data among the different components:

- Sequential Data Flow (SDF): The data flows from tertiary to memory (STREAM 1 of Figure 1), from memory to the disk drives (STREAM 2), from the disk drives back to memory (STREAM

[2]Neither simulated nor implemented.

3), and finally from memory to the display station referencing the object (STREAM 4).

- Parallel Data Flow (PDF): The data flows from the tertiary to memory (STREAM 1), and from memory to both the disk drives and the display station in order to materialize (STREAM 2) and display (STREAM 4) the object simultaneously. (PDF eliminates STREAM 3.)
- Incomplete Data Flow (IDF): The data flows from tertiary to memory (STREAM 1) and from memory to the display station (STREAM 4) to support a continuous retrieval of the referenced object. (IDF eliminates both STREAM 2 and 3.)

Figure 1 models the second architecture (tertiary storage is accessible only to the disk drives) by partitioning the available memory into two regions: one region serves as an intermediate staging area between tertiary and disk drives (used by STREAM 1 and 2) while the second serves as a staging area between the disk drives and the display stations (used by STREAM 3 and 4). SDF can be used with both architectures. However, neither PDF nor IDF is appropriate for the second architecture because the tertiary is accessible only to the disk drives. When the bandwidth of the tertiary storage device is lower than the bandwidth required by an object, SDF is more appropriate than both PDF and IDF because it minimizes the amount of memory required to support a continuous display of an object. IDF is ideal for cases where the expected future access to the referenced object is so low that it should not become disk resident (i.e., IDF avoids this object from replacing other disk resident objects).

The rest of this paper is organized as follows. In section 2, we describe how our target environment supports a continuous display of an object. Section 3 uses this framework to describe the pipelining mechanism assuming that the database consists of a single media type (with a fixed bandwidth requirement $B_{Display}$). We describe our technique from the perspective of a tertiary storage device whose bandwidth is either higher or lower than the bandwidth requirement of a media type $B_{Display}(X)$. Extension of the pipelining mechanism to a database that consists of a mix of media type (each with a different bandwidth requirement) is described in [GDS94]. Our conclusion and future research directions are described in Section 4.

2 Overview

Given a system that consists of D disk drives and a database that consists of objects that belong to a single media type (with bandwidth requirement $B_{Display}$), we utilize the aggregate bandwidth of d disk drives to support a continuous display of an object. This is achieved as follows. First, the D

disk drives in the system are partitioned into R clusters where $R = \lfloor \frac{D}{d} \rfloor$. Next, each object in the database (say X) is striped [SGM86] into n equi-sized subobjects (X_1, X_2, ..., X_n). Each subobject X_i represents a continuous 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. The primary reason for a round-robin assignment of the subobjects to the clusters is to distribute the workload imposed by the display of an object evenly across the clusters. This avoids a cluster from becoming the bottleneck for the system, maximizing its processing capability. In a cluster, a subobject is declustered [RE78, LKB87, GDQ92] into d pieces (termed **fragments**), with each fragment assigned to a different disk in the cluster. The value of d is chosen such that the bandwidth of a cluster is greater than or equal to the bandwidth required to display an object.

Assuming that the bandwidth of each cluster is high enough that it can be multiplexed between $U_{Cluster}$ requests, the memory used to display objects is partitioned into $R \times (U_{Cluster} + 1)$ frames[3]. To ensure a continuous display of an object, the system maintains a **time cycle** for each cluster. A time cycle consists of $U_{Cluster}$ **time intervals** (also termed slots). A time interval is the time required for a cluster to reposition its disk head(s) and transfer a subobject into a memory frame. Given a request for an object X that consists of n subobjects, the system reserves n time intervals on behalf of this request, one per time cycle of the system. Relative in time, the distance between any two time slots reserved on behalf of a request is $U_{Cluster} - 1$. The cluster employed in the first time cycle is the one containing X_0 (say C_i). The display of X starts once X_0 is staged in memory. In the second cycle, cluster $C_{(i+1) \bmod R}$ is employed to read X_1. The organization of both the cycles and intervals is such that X_1 is memory resident immediately before the display of X_0 completes (this explains why the memory is partitioned into $R \times (U_{Cluster} + 1)$ frames). This is achieved by setting the duration of a time cycle to be equivalent to the display time of a subobject:

$$TimeCycle = \frac{size(subobject)}{B_{Display}} \tag{1}$$

The system switches from the memory frame containing X_0 to X_1 in order to support a continuous display of X. The system iterates over the clusters and memory frames until X is displayed in its entirety employing a single cluster in each time cycle.

Example 1: Assume a system that consists of 3 disk clusters. Moreover, assume that the bandwidth of each cluster is twice the bandwidth required to display an object ($U_{Cluster} = 2$). Let the following three objects reside on the disk clusters: X, Y, and Z. The size of a subobject of each of these objects is identical. X is striped into 8 subobjects while each of Y and Z is striped into 5

[3]This equation is explained in the following description.

Figure 2: A schedule for servicing three requests

subobjects (X is larger in size than both Y and Z). These objects are assigned to the clusters in a round-robin manner starting with a different cluster for each object (say cluster 0 for X, cluster 1 for Y, and cluster 2 for Z; the assignment of subobjects to the different clusters is shown in Table 1). Assume that three requests are issued, each referencing a different object, in the following order of arrival: X, Y, followed by Z. Figure 2 demonstrates the scheduling of the time intervals to display the different subobjects as a function of time. This schedule is possible due to the assignment of Table 1, allowing the system to display X, Y, and Z in the same time cycle. Note that X_i employs the same time interval in each time cycle (interval 1). Moreover, relative in time, the slots reserved on behalf of a display (say Y) are one slot ($U_{Cluster} - 1$) apart. Thus, Equation 1 ensures that Y_{i+1} is memory resident before the display of Y_i completes. $\Box$

Each of STREAM 2 and 3 in Figure 1 requires the use of time intervals. As demonstrated by Example 1, STREAM 3 requires a single time interval. The number of time intervals required by STREAM 2 depends on the bandwidth of the tertiary storage device.

Term	Definition
D	Number of disk drives in the system
$B_{Display}$	Bandwidth required to display an object
$B_{Tertiary}$	Bandwidth of the tertiary storage device
R	Number of disk clusters in the system
$size(X)$	Size of object X
$T_{Reposition}$	Time required for the tertiary to reposition its read head
n	Number of subobjects that constitute an object
PCR	Production Consumption Ratio, $\frac{B_{Tertiary}}{B_{Display}}$

Table 2: List of terms used repeatedly in this paper and their respective definitions

When materializing an object from tertiary, if the bandwidth of tertiary is lower that the bandwidth required to display the object then the tertiary cannot produce an entire subobject during each time interval to be flushed to a disk cluster. This is not a problem for a system that consists of a single cluster (R=1). However, this is a problem when $R > 1$ because the layout of an object across the disk drives is not sequential (a fragment does not represent a contiguous portion of an object, see [GRAQ91, BGMJ94]). Consequently, when materializing object X, the tertiary produces $\frac{B_{Tertiary}}{B_{Display}}$ of X_0 during the first time cycle. During the second time cycle, if the system forces the tertiary storage device to reposition its read head in order to produce $\frac{B_{Tertiary}}{B_{Display}}$ of X_1, the tertiary may incur an unacceptable overhead. One approach to resolve this mismatch is to write the data on the tape in the same order as it expected to be delivered to the disks. For example, in a system that consists of 3 disk clusters, if $\frac{B_{Tertiary}}{B_{Display}} = 0.8$ then the subobjects could be stored on the tertiary as follows: $0.8X_0$, $0.8X_1$, $0.8X_2$, $(0.2X_0, 0.6X_3)$, $(0.2X_1, 0.6X_4)$, $(0.2X_2, 0.6X_5)$, etc. (see Example 3 for a completion of this illustration). This would allow the system to read X sequentially from tertiary while ensuring a round-robin assignment of its subobjects to the disk clusters. We assume this approach in this paper.

When the system consists of a single disk cluster ($R = 1$) and $U_{Cluster} = 1$, the system can either display or materialize a single object (either STREAM 2 or 3). In this case, the pipelining mechanism cannot be employed. In this paper, we assume that the product of $U_{Cluster}$ and R is greater than one ($U_{Cluster} \times R > 1$).

3 Pipelining

In this section we assume that all the objects belong to a single media type and have the same bandwidth requirement. We describe the pipelining mechanism for two possible cases: the bandwidth

Figure 3: Non-Pipelining Approach: Single Disk Cluster

of the tertiary is either 1) lower or 2) higher than the bandwidth required to display an object[4]. The ratio between the production rate of tertiary and the consumption rate at a display station is termed Production Consumption Ratio ($PCR = \frac{B_{Tertiary}}{B_{Display}}$). When $PCR < 1$ ($PCR > 1$), the production rate of tertiary is lower (higher) than the consumption rate at both a display station (STREAM 4 of Figure 1) and the bandwidth simulated by using a single time interval per time cycle (STREAM 2 of Figure 1).

3.1 $PCR < 1$

In this case, the time required to materialize an object is greater than its display time. Neither PDF nor IDF is appropriate because the bandwidth of tertiary cannot support a continuous display of the referenced object (assuming that the size of the first slice exceeds the size of memory). We start by describing SDF for a system that consists of a single disk cluster with $U_{Cluster} > 1$. Subsequently, we extend the discussion to a system that consists of R disk clusters.

Let an object X consists of n subobjects. The time required to materialize X is $\lceil \frac{n}{PCR} \rceil$ time cycles while its display requires n time cycles. If X is tertiary resident, without pipelining the latency time incurred to display X is $\lceil \frac{n}{PCR} \rceil + 1$ time cycles (see Figure 3). (Plus one because an additional time cycle is needed to both flush the last subobject to the disk cluster and allow the first subobject to be staged in the memory buffer for display, e.g., time cycle 11 in Figure 3.) To reduce this latency time, a portion of the time required to materialize X can be overlapped with its display time. This is achieved as follows. An object X is split into s logical slices ($S_{X,1}$, $S_{X,2}$, ...,

[4] The discussion for the case when the bandwidth of tertiary is equivalent to the display is a special case of item (2).

Figure 4: Pipelining Mechanism: Single Disk Cluster

$S_{X,s}$), such that the display time of $S_{X,1}$ ($T_{Display}(S_{X,1})$) overlaps the time required to materialize $S_{X,2}$ ($T_{Materialize}(S_{X,2})$), $T_{Display}(S_{X,2})$ overlaps $T_{Materialize}(S_{X,3})$, 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 with SDF is as follows:

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

STEP 2: For $i = 2$ to s do

(a) Initiate the materialization of $S_{X,i}$ from tertiary device onto disks[5].

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

STEP 3: Display the last subobject[6].

The duration of STEP 1 determines the latency time of the system. During STEP 2, while the subsequent slices are materialized from the tertiary device, the disk resident slices are being displayed. STEP 3 displays the last subobject materialized on the disk clusters.

STEP 1 requires a single memory frame and a single time interval per time cycle, while STEP 2 requires two memory frames and two time intervals per cycle (additional resources should not be allocated as they cannot be utilized), and STEP 3 requires one memory frame and time slot to display the last subobject. The tertiary storage device is fully utilized during both STEP 1 and 2, and it is not required during STEP 3. In STEP 1, during the first time cycle, the system reads PCR

[5]During this step, at least a portion of (not necessarily all) of $S_{X,i}$ is materialized on the disk cluster.

[6]The last subobject has already been staged in the memory frame.

of a subobject into a memory frame. In each of its subsequent time cycles, it flushes the partially full frame onto a disk cluster and continues to read PCR of a subobject from the tertiary storage device. In STEP 2, during each time cycle, the system uses one memory frame and time interval to repeat the procedure outlined for STEP 1 (to accomplish STEP 2.a), and a second memory frame and time interval to support the display of subobjects that constitute a previously materialized slice $S_{X,i-1}$. Finally, in STEP 3 the last subobject is displayed using a single memory frame. Note that both STEP 1 and 2.a waste 1-PCR of a time interval because they write partially full buffer frames.

Example 2: Assume a system with a single disk cluster ($R = 1$) as in Figure 4. Each time cycle consists of two time intervals ($U_{Cluster} = 2$). Assume that object X consists of 8 subobjects. If $PCR = 0.8$, then the time required to materialize the object is 10 time cycles ($\lceil \frac{n}{PCR} \rceil$). Figure 3 shows the materialization of object from the tertiary device followed by its display without the use of the pipelining mechanism, yielding a latency time of 11 time cycles. Figure 4 shows how the pipelining mechanism overlaps the display of X with its materialization. In this case, the incurred latency time is 4 time cycles while the system ensures a continuous display of X. $\square$

The number of time cycles required for each step is computed as follows. From Figure 4 it is obvious that STEP 3 requires no time slots. Furthermore, the duration of STEP 2 corresponds to the display time of $n - 1$ subobjects that requires n time cycles. To compute the number of time cycles for STEP 1 ($TC_{STEP1}(X)$), we subtract the total time required for STEP 2 (n time cycles) from the time required to flush X to the disk clusters in its entirety ($\lceil \frac{n}{PCR} \rceil + 1$ time cycles):

$$TC_{STEP1}(X) = \lceil \frac{n}{PCR} \rceil - n + 1 \tag{3}$$

During STEP 2, the portion of the object materialized by the pipelining mechanism is $\lfloor PCR * (n-1) \rfloor$. The remainder of the object must constitute $S_{X,1}$:

$$S_{X,1} = n - \lfloor PCR * (n-1) \rfloor \tag{4}$$

The granularity of $S_{X,1}$ is in terms of subobjects. The size of $S_{X,1}$ is important because it determines the latency incurred when X is referenced. The size of the slices are different and can be computed as a function of $S_{X,1}$. This is not presented because their size has no impact on the latency time of the system.

Now, we extend the discussion of the pipelining mechanism to a system that consists of R disk clusters. Recall that, the subobjects that constitute X are assigned to the disk clusters in a round robin manner. Furthermore, the physical layout of X on the tertiary device accommodates the round robin assignment of the subobjects and the time slots to the R disk clusters. To preserve the round robin assignment of the subobjects, the system may require more than $\lceil \frac{n}{PCR} \rceil$ time cycles to render object X disk resident. This is because the last subobject of X (i.e., X_n) might be memory resident, however, the cluster that should contain X_n might be busy servicing other requests. In the worst case, this cluster might be busy for $R-1$ additional time cycles before it is assigned to the materialization procedure, allowing the system to render X_n disk resident. If the memory containing X_n is accessible to the display stations, then the system can display X_n from memory in order to ensure a continuous display of X (using the PDF paradigm for the last subobject). Otherwise, the number of time cycles required for STEP 1 should be extended with the number of time cycles required to make X_n disk resident in order to ensure a continuous display. This increases the size of the first slice, resulting in a higher latency time.

To compute the exact number of time cycles required before the entire object becomes disk resident with R disk clusters, the system can employ the following equation:

$$TC_{Materialization}(X) = (\lfloor \frac{l}{R} \rfloor + \lceil \frac{\lfloor \frac{l \bmod R}{\max(1, n \bmod R)} \rfloor}{R} \rceil) \times R + (n \bmod R); where\ l = \lceil \frac{n}{PCR} \rceil \quad (5)$$

This equation compensates for the delay associated with the preservation of a round-robin assignment of the subobjects to the disk clusters. The last subobject may not become disk resident immediately after $l = \lceil \frac{n}{PCR} \rceil$ time cycles. Number of complete round-robin cycles[7] required for the object materialization onto the disk clusters is: $\lfloor \frac{l}{R} \rfloor + \lceil \frac{\lfloor \frac{l \bmod R}{\max(1, n \bmod R)} \rfloor}{R} \rceil$. The last subobject, X_n, becomes disk resident after: number of complete round-robin cycles$\times R$ + location of X_n in the round-robin assignment[8].

Substituting Equation 5 in place of $\lceil \frac{n}{PCR} \rceil$ in Equation 3, the number of cycles required for STEP 1 is:

$$TC_{STEP1}(X) = TC_{Materialize}(X) - n + 1 \quad (6)$$

The system compensates for the newly introduced delays by expanding the duration of STEP 1 (increasing the size of the first slice) to ensure a continuous display. Hence, the number of time cycles in STEP 2 and STEP 3 remain unchanged[9]. Consequently, after completing the retrieval of the object, the tertiary is free to service other requests. The last portion of X continues to be memory resident until it is flushed to a disk cluster. This may avoid the tertiary from servicing another request if the available memory is exhausted.

[7]A round-robin cycle consists of R time cycles.
[8]This is captured by: $(n \bmod R)$.
[9]STEP2 may either partially employ or not employ $R-1$ time slots allocated to it, see Example 3 for illustration.

Figure 5: Pipelining Mechanism: 3 Disk Clusters

Example 3: Assume a system with 3 disk clusters as in Figure 5 and the same values for n, PCR, and $U_{Cluster}$ as in Example 2. If object X consists of 8 subobjects, its subobjects would be stored on the tertiary as follows: $0.8X_0$; $0.8X_1$; $0.8X_2$; $(0.2X_0, 0.6X_3)$; $(0.2X_1, 0.6X_4)$; $(0.2X_2, 0.6X_5)$; $(0.4X_3, 0.4X_6)$; $(0.4X_4, 0.4X_7)$; $(0.4X_5, 0.4X_6)$; $(0.2X_6, 0.6X_7)$. The semicolon separates the portions that become memory resident during each time cycle. Figure 5 shows how each portion is flushed to the disk cluster to attain the round-robin assignment of X as shown in Table 1. The first portion of X $(0.8X_0)$ is read into the memory during time cycle 1. During time cycle 2, this portion is flushed to cluster 0 and $0.8X_1$ is read from tertiary. This process is repeated for the remaining portions of X. The last portion $(0.2X_6, 0.6X_7)$ becomes memory resident at the end of the tenth time cycle. The portion corresponding to $0.2X_6$ is flushed to cluster 0 during the eleventh time cycle (wasting 80% of a time interval). The portion corresponding to $0.6X_7$ must be stored on cluster 1 (X_7 must reside on cluster 1 to ensure a round-robin assignment, see Table 1). If the tertiary abides by the round-robin usage of clusters then $0.6X_7$ is flushed during the twelfth time cycle (see Figure 5). Equation 5 and 6 increase the size of the first slice accordingly to ensure that this discrepancy does not disrupt the continuous retrieval of[10] X, resulting in a higher latency as the display of X starts during time cycle 5 (instead of 4). If tertiary is allowed to flush $0.6X_7$ to cluster 1 during time cycle 11, then the display of X could start during time cycle 4 (as before). However, note that this violation may interfere with the display of another object that requires a different subobject that resides on cluster 1 during time cycle 11.

If the memory used as the staging area between tertiary and disk clusters is available to the display stations, the system may employ PDF for the last subobject. In this case the display of

[10]The display of X is disrupted when the system attempts to display a subobject that is not disk resident.

X starts during time cycle 4. The last portion, i.e., $0.6X_7$, is displayed and flushed simultaneously during time cycle 12. □

Using the above equations a scheduler can determine the number of time slots and memory frames required for the pipelining mechanism at each time cycle, and the number of subobjects required to be disk resident ($S_{X,1}$) before the display of object X can start.

3.2 $PCR > 1$

In this case, the bandwidth of tertiary exceeds the bandwidth required to display an object. Therefore, Equation 2 is satisfied when each slice of an object consists of a single subobject. Moreover, the layout of each object on the tertiary storage device is sequential and device independent because a complete subobject can be materialized during each time cycle.

Two alternative approaches can be employed to compensate for the fast production rate: either 1) multiplex the bandwidth of tertiary among several requests referencing different objects, or 2) increase the consumption rate of an object by reserving more time intervals per time cycle to render that object disk resident. The first approach wastes the tertiary bandwidth because the device is required to reposition its read head multiple times. The second approach utilizes more resources in order to avoid the tertiary device from repositioning its read head. The resources required per time cycle on behalf of an approach can be variant. In the following two sections, we consider each approach in turn. We develop analytical models that determine the combination of resources required for each approach.

3.2.1 Multiplexing

One approach to compensate for the high bandwidth of tertiary is to multiplex its bandwidth among several requests, providing each request referencing an object X with a bandwidth equal to $B_{Display}(X)$. We start this section by describing this approach for PDF paradigm. Subsequently, we demonstrate how the utilization of the tertiary storage device can be maximized using a combination of both PDF and IDF. We conclude by describing this approach for SDF.

Assuming that the tertiary storage device is multiplexed among j distinct requests, and a new request arrives increasing the total number of requests to k (i.e. $j + 1$). The following shows the PDF paradigm for the newly arrived request:

STEP 1: Materialize the first portion of the object referenced by the newly arrived request into memory buffers (a portion consists of one or more subobjects).

STEP 2:

(a) Each memory buffer is displayed and flushed to a disk cluster at the same time.

(b) Next portions of each of the k referenced objects are transfered to the memory buffers.

STEP 3: If the materialization of one of the k objects (say X) completes then the tertiary either sits idle and waits for the arrival of a new request or services another request based on the availability of its resources and the bandwidth requirement of the object referenced by the new requests.

The number of objects that can be multiplexed simultaneously (k) depends on $B_{Tertiary}$ and the time required to reposition the read head of tertiary among these requests. Assume that a fixed size memory is allocated to each of the k requests, where the size of memory is a multiple of the subobject size, say $z \times size(subobject)$ where z is an integer. The time required to display the memory buffer for each object X (i.e., $\frac{z \times size(subobject)}{B_{Display}}$) should be greater than or equal to the total time required to: 1) multiplex the tertiary among $k-1$ other requests (i.e., $(k-1) \times \frac{z \times size(subobject)}{B_{Tertiary}}$), 2) materialize the next portion of X (i.e., $\frac{z \times size(subobject)}{B_{Tertiary}}$), and 3) the time required for the tertiary to reposition its head among the k requests (i.e., $\sum_{i=1}^{k} T_{Reposition}(o_i)$). Thus, multiplexing must satisfy the following constraint on behalf of each of the k requests:

$$\frac{z \times size(subobject)}{B_{Display}} \geq k \times \frac{z \times size(subobject)}{B_{Tertiary}} + \sum_{i=1}^{k} T_{Reposition}(o_i) \tag{7}$$

If this constraint is violated then PDF cannot guarantee a continuous display of an object. In this case, the system may employ the SDF paradigm as described in section 3.1 because each of the k requests observes a PCR less than 1. For the rest of this section, we describe PDF assuming that the constraint posited in Equation 7 is satisfied.

Equation 7 assumes that k time slots are reserved per time cycle. Moreover, the k reserved time slots should be positioned such that the system flushes the subobjects to the clusters in the same manner (round-robin) as it would have been read if the object was disk resident (each of the z subobjects read on behalf of a request is flushed and displayed simultaneously, one per time cycle). The upper bound on the number of required time slots is $\lfloor PCR \rfloor$; no additional time slots should be allocated as they cannot be used.

An interesting property of Equation 7 is the effect of memory buffers. By increasing z, more data is transfered every time the read head of the tertiary is repositioned. This enables the system to multiplex requests that require the read head of tertiary to travel a longer distance and incur a higher repositioning time.

Parameter	Value
$B_{Tertiary}$	50 mbps
$B_{Display}$	20 mbps
$size(o_x)$	20 megabyte
$size(o_y)$	10 megabyte
$size(subobject)$	1.25 megabyte
k	2
$\sum_{i=1}^{k} T_{Reposition}(o_i)$	1 second

Table 3: System parameters for the example

z	Maximum $\sum_{i=1}^{k} T_{Reposition}(o_i)$ tolerated by Equation 7
1	1 Second
2	2 Seconds
3	3 Seconds

Table 4: The effect of memory on repositioning time

Example 4: Consider the system parameters shown in Table 3. Assuming one frame of memory is dedicated for each object to be multiplexed ($z = 1$) and two time slots are available ($k = 2$), the constraint posited in Equation 7 is satisfied. Table 4 shows the effect of increasing the amount of memory (z) on the feasible repositioning times. By increasing z, Equation 7 is satisfied for higher repositioning time (depending on the distance between the k multiplexed objects). □

A static scheduler can determine if a new requests can be added to the k currently multiplexed requests based on the available resources (time slots and memory) and the idle time of the tertiary device. To multiplex k objects using PDF, k time slots and $k \times z$ memory frames are required. The idle time of the tertiary when multiplexed among k requests (derived using Equation 7) is:

$$IdleTime = \frac{z \times size(subobject)}{B_{Display}} - \sum_{i=1}^{k} T_{Reposition}(o_i) - \frac{z \times k \times size(subobject)}{B_{Tertiary}} \tag{8}$$

If the constraint posited in Equation 7 is satisfied on behalf of each multiplexed request then $IdleTime$ will be a number either greater than or equal to zero. the minimum value for k is one. In this case, the tertiary is used for $\frac{z \times size(subobject)}{B_{Tertiary}}$ and sits idle for the duration of time computed by Equation 8. This process is repeated $\frac{n}{z}$ times. This shows that in the worst case the system can employ the PDF paradigm with one time slot and one memory frame (the required resources to display an object). Upon the arrival of a new request, the scheduler may encounter alternative scenarios: 1) the additional reposition time required to service the new request renders the idle time of the tertiary to be a negative number (this can also happen when the new request results in a k that is greater than $\lfloor PCR \rfloor$), 2) there is insufficient memory to service the new requests, 3) there are insufficient time slots to materialize the referenced object, and 4) a combination of the first three scenarios. In the first two cases (and any combination that includes these two case), the system has no choice and must schedule the new requests to be served at some point in the future when one of the k active requests completes.

For the third scenario, the scheduler may employ the IDF paradigm (in combination with PDF) to harness the full bandwidth of the tertiary storage device without materializing the object on the disk drives. the IDF paradigm is specially useful for requests that reference objects whose frequency of access is so low that the system elects not to materialize them on the disk drives. Assuming that the system services m requests using the IDF paradigm, the multiplexing approach must guarantee the following constraint on behalf of each $(k+m)$ requests:

$$\frac{z \times size(subobject)}{B_{Display}} \geq (k+m) \times \frac{z \times size(subobject)}{B_{Tertiary}} + \sum_{i=1}^{k+m} T_{Reposition}(o_i) \tag{9}$$

where $k+m \leq \lfloor PCR \rfloor$. The system is pure PDF (IDF) when $m = 0 (k = 0)$.

With multiplexing, the SDF paradigm is almost identical to the PDF paradigm. The major differences are as follows. First, SDF requires more resources as compared to PDF. To multiplex k objects using SDF, the system must allocate:

- $2 \times k$ time slots: k time slots to transfer the different subobjects from tertiary to disk cluster, another k time slots to display the disk resident subobjects.
- $(k \times z) + k$ frames of memory: $k \times z$ frames serve as an intermediate staging area between tertiary and multi-disk to materialize k objects, while k memory frames are used to initiate the display of k disk resident subobjects.

PDF requires only k time slots and $k \times z$ memory frames because it eliminates STREAM 3 of SDF. Note that SDF cannot be applied unless two time intervals are allocated on behalf of a request: one for STREAM 2 and a second for STREAM 3. Second, the latency time is higher with SDF as compared to PDF. With PDF, the latency time is equivalent to one time cycle (for STREAM 1). With SDF, the latency time incurred by a request is dependent on the assignment of its two time slots. If the slots are assigned horizontally (i.e., the slots are from a single cluster C_i), its latency time is two time cycles: one for STREAM 1 and second to perform STREAM 2 and 3 concurrently. However, if the two slots are assigned vertically, one from cluster C_i and second from cluster C_j (say

$i < j$), then the latency time is the sum of: 1) one time cycle for STREAM 1, 2) one time cycle for STREAM 2 (subobject flushed to C_i), and 3) $j - i$ time cycle delay until STREAM 3 can be initiated ($j - i$ is the number of time cycles required for the interval corresponding to C_i to reach the position of time interval corresponding to C_j that contains the first subobject of[11] X). Third, PDF enforces k to be lower than $\lfloor PCR \rfloor$. This constraint can be violated with the SDF paradigm, resulting in a higher latency time on behalf of each request (i.e., when $k < \lfloor PCR \rfloor$ the first slice consists of a single subobject, when $k > \lfloor PCR \rfloor$ the number of subobjects for the first slice is determined by the discussion of Section 3.1). Note that when the system services m requests using IDF, it allocates $m \times z$ memory frames to support their display.

3.2.2 Non-Multiplexing

The second approach accommodates the high bandwidth of tertiary storage device by increasing the consumption rate of the system. This is achieved by dedicating additional time slots per time cycle to materialize the referenced object (increasing the rate of data flow in STREAM 2), with one time slot dedicated to display the object (STREAM 3). This technique is appropriate for neither PDF nor IDF because both techniques require the production rate of the tertiary (STREAM 1) to be approximately the same as $B_{Display}$ (STREAM 4)[12]. However, it is appropriate for SDF because it uses the disk space as a staging area to materialize the object (STREAM 2) at a rate of $B_{Tertiary}$ while retrieving (STREAM 3) and displaying (STREAM 4) it at a rate of $B_{Display}$, eliminating the constraint enforced by both PDF and IDF. With SDF, the display of an object is initiated once the first subobject becomes disk resident.

Assuming k is the number of time slots allocated per time cycle to materialize the object, the ideal case is when k is equal to $\lceil PCR \rceil$ because it renders the consumption rate of the disk cluster to be either higher or equivalent to $B_{Tertiary}$. If $k < PCR$, then some extra memory is required to temporarily buffer the portion that cannot be flushed to the clusters. In this case, the memory resident portion continues to accumulate as a function of time until the object becomes disk resident in its entirety. This section describes analytical models to determine the number of required time slots and memory frames used to materialize the subobjects on the disk clusters. We start by assuming that k is a constant for the total number of time cycles required to materialize the reference object. Subsequently, we relax this assumption and consider the case where the value of k fluctuates from

[11]If the first subobject was flushed to C_i instead of C_j (in item 2) then item 3 would incur a delay of $R - j + i$.The system can choose either C_i or C_j to decrease the delay ($Min(R - j + i, j - i)$).

[12]Multiplexing (as described in section 3.2.1) decreases rate of data flow in STREAM 1 in order to employ both PDF and IDF.

one cycle to another.

Assuming k time slots are available per time cycle, consider a vertical assignment of time slots across the clusters: the time slots are assigned among the clusters in a round-robin manner starting with an available cluster. Thus, h_c time slots are dedicated to cluster c where $\lfloor \frac{k}{R} \rfloor \leq h_c \leq \lceil \frac{k}{R} \rceil$ (i.e., the k time slots are divided equally among the R clusters). When $h_c > 0$, the clusters are adjacent due to round-robin assignment. Obviously, k should be $\sum_{c=1}^{R} h_c$ and the physical upper bound enforced on k is $R \times U_{Cluster}$. Assuming the vertical assignment, we now calculate the maximum amount of memory required to prevent overflow. During each time cycle, $Min(k, PCR)$ subobjects are transfered from memory to disk clusters. Assuming the object consists of n subobjects, then the number of time cycles required to render the object disk resident in its entirety is:

$$q = \lceil \frac{n}{Min(k, PCR)} \rceil \tag{10}$$

Moreover, the number of time cycles required to read the object from tertiary is $\frac{n}{PCR}$. If $k < PCR$ then the production rate will be higher than the consumption rate, requiring more cycles to flush the object onto the disk drives as compared to the number of cycles required to read it from tertiary. The difference between these two components $(\frac{n}{Min(k,PCR)} - \frac{n}{PCR})$ determines the portion of an object that becomes memory resident once tertiary completes servicing this request. To compute the exact number of subobjects that are memory resident, it is sufficient to multiply this difference by $Min(k, PCR)$:

$$MaxMem = n - \frac{n \times Min(k, PCR)}{PCR} \tag{11}$$

$MaxMem$ represents the maximum amount of memory required in one or more time cycles[13]. The amount of memory required for each time cycle is computed as follows: 1) compute the fraction of an object that is disk resident per time cycle, 2) compute the fraction that remains on the tertiary storage device per time cycle, 3) subtract the size of an object from the sum of item 1 and 2. The number of subobjects that are tertiary resident at cycle i, for each time cycle i where $i = 0, 1, ..., q$ is:

$$Ter_i = Max(n - (i \times PCR), 0) \tag{12}$$

where PCR is the number of subobjects retrieved from the tertiary per time cycle, i is the number of time cycles elapsed since the initiation of the materialization, and n is the number of subobjects that

[13]In computing $MaxMem$ we assumed that different fragments of different subobject are transfered from memory to disk clusters simultaneously. Therefore, the available memory should be managed in frames, where the size of a frame corresponds to the size of a fragment; otherwise, the available memory may become fragmented, reducing its utilization. Detailed discussion on memory management is part of a system scheduler and constitutes our future research direction.

Figure 6: Object layout on the tertiary

constitute the referenced object. Therefore, $i \times PCR$ is the number of subobjects that are retrieved from the tertiary thus far. By deducting this from n, we compute the number of subobjects that are still tertiary resident. The maximum function is used to avoid Equation 12 from producing a negative value once the tertiary has completed the retrieval of the object. When the materialization first starts (time cycle 0), $Ter_0 = n$.

The number of subobjects that reside on the disk clusters at time cycle i is:

$$Disk_i = Min(Disk_{i-1} + Min(k, PCR), n) \tag{13}$$

where $Disk_{i-1}$ represents the accumulated subobjects that reside on the disk clusters ate time cycle $i - 1$, and $Min(k, PCR)$ is the number of subobjects added to disk cluster at time cycle i. The minimum function is used to avoid Equation 13 from producing a value greater than n once the object has transfered to the disk clusters in its entirety. When the materialization first starts, $Disk_0$=0.

Hence, the number of subobjects that are memory resident at time cycle i is:

$$Mem_i = n - (Ter_i + Disk_i) \tag{14}$$

To illustrate the application of the above analytical models, and to discuss how this method supports the round-robin assignment of the subobjects to the disk clusters, consider the following examples.

Cluster 0	Cluster 1	Cluster 2
X_0	X_1	X_2
X_3	X_4	X_5
X_6	X_7	X_8
X_9	X_{10}	X_{11}

Table 6: Desired assignment of subobjects to a three cluster system

Time cycle (i)	0	1	2	3	4	5	6	7	8	9	10	11	12
Ter_i	12	10.5	9	7.5	6	4.5	3	1.5	0	0	0	0	0
$Disk_i$	0	1	2	3	4	5	6	7	8	9	10	11	12
Mem_i	0	0.5	1	1.5	2	2.5	3	3.5	4	3	2	1	0
Flushed to Disk	-	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}
Memory Resident	-	$\frac{1}{2}X_1$	X_2	$X_3, \frac{1}{2}X_4$	X_4 X_5	X_5, X_6 $\frac{1}{2}X_7$	X_6, X_7 X_8	X_7, X_8 $X_9, \frac{1}{2}X_{10}$	X_8, X_9 X_{10}, X_{11}	X_9, X_{10} X_{11}	X_{10} X_{11}	X_{11}	-

Table 7: Example five

Table 5 contains the parameters for all the examples. Fig 6 shows the layout of object on the tertiary storage device, and based on the PCR, the portion read per time cycle. Thus, it requires 8 time cycles to read X from tertiary. Round-robin assignment of X on the three cluster system is shown in Table 6.

Example 5: Assume $k = 1$ time slot is dedicated per time cycle to materialize X. Equation 10 determines that 12 ($q = 12$) time cycles are required before X becomes completely disk resident. Equation 11 determines that the maximum amount of required memory corresponds to the size of four subobjects ($MaxMem = 4$). Equation 12 to 14 are use to compute the amount of memory required during each time cycle (Table 7). The numbers in the first three rows of the table correspond to the number of subobjects that reside on either the tertiary, disk or memory. The last two rows of the Table 7 show the subobject(s) that become either memory or disk resident at each time cycle respectively. The schedule for allocating the clusters (to flush the subobjects onto disk drives) is identical to the one for displaying them (see Figure 2).

Table 7 shows that the maximum memory required is 4 subobjects during time cycle eight (cycle $\lfloor \frac{n}{PCR} \rfloor$), and it is the same as that computed by Equation[14] 11. □

Example 6: By dedicating two time slots per cycle to materialize X ($k = 2$), the time required to materialize X is reduced to 8 time cycles ($q = 8$) and no extra memory is required ($MaxMem = 0$). Note that because $k > PCR, k - PCR$ (i.e., 0.5) of the disk bandwidth allocated during a time

[14]If n is not divisible by PCR, then the last portion of the object is less than $Min(k, PCR)$. This is not captured in Equation 11; however, Equation 14 compensates to compute the exact amount of required memory per cycle.

Figure 7: Schedule for example six

interval is wasted (see Table 8 and Figure 7). □

In Example 6 where $k > PCR$, once the pipelining mechanism employ k clusters during time cycle 1 (say clusters: $(i, i+1, ..., i+k-1)$ modulo R), in the second cycle it employs the k clusters starting with either cluster $((i+k-1)$ modulo $R)$ or cluster $((i+k)$ modulo $R)$. If cluster $((i+k-1)$ modulo $R)$ is employed as the starting cluster during time cycle 1, then there is a logical overlap between time cycle 1 and 2. To illustrate, in Example 6, there was an overlap between cycles 1 and 2 (cluster 2) and no overlap between cycles 2 and[15] 3. In general, when $PCR \times m = integer$, the first cycle uses the $((i+k)$ modulo $R)$ criteria, followed by $m-1$ cycles use $((i+k-1)$ modulo $R)$ criteria. This pattern repeats itself for the duration of materialization. To illustrate, if PCR was 1.25 then $m = 4$, and the employed clusters in the different time cycles are: $(1,2) \rightarrow (2,3) \rightarrow (3,1) \rightarrow (1,2) \rightarrow (3,1) \rightarrow (1,2) \rightarrow (2,3) \rightarrow$ etc. If there is no such m that $PCR \times m = integer$ then there is always an overlap between consecutive cycles (i.e., employ $((i+k-1)$ modulo $R)$ criteria). When $k < PCR$ there is no overlap between consecutive time cycles (see Example 5). Note that there is no constraint on the occupation of time slots during each time cycle. For example, $\frac{1}{2}X_1$ may occupy the second time slot of the first time cycle that corresponds to cluster 2 (Figure 7).

We now extend the discussion to consider the case where k is no longer a constant. The scheduler

[15]In Figure 7, the physical overlap between the second and third time cycles (i.e., cluster 2) is because of modulo R and is not a logical overlap.

may vary k for different time cycles to achieve the best memory/time-slot combination based on its perspective on the future status of resources. Obviously, Equations 10 and 11 are no longer valid. Furthermore, Equation 13 should be revised to:

$$Disk_i = Min(Disk_{i-1} + Min(k_i, PCR + Mem_{i-1}), n) \tag{15}$$

where k_i is the number of time slots available during time cycle i. To describe $Min(k_i, PCR + Mem_{i-1})$, note that at the $i$the time cycle, Mem_{i-1} subobjects are memory resident and ready for consumption. In addition, PCR subobjects are produced per time cycle by the tertiary. Thus, if enough time slots are available per cycle, all of these $(PCR+Mem_{i-1})$ can be consumed. Otherwise, the provided number of time slots in that cycle (k_i) determines the number of consumed subobjects. In Equation 13, since k is fixed in all q time cycles, if $k > PCR$ then $Mem_i = 0$ for all $i = 1, 2, ..., q$. This results in $Min(k, PCR + Mem_{i-1}) = PCR$. However, if $k \leq PCR$ then independent of the value of $Mem_{i-1}, Min(k, PCR + Mem_{i-1}) = k$. This explains why Mem_{i-1} does not appear in Equation 13.

In general, the number of available time slots per time cycle determines the amount of memory required per time cycle. This technique does not waste the bandwidth of the tertiary storage device, however, it requires more resources as compared to multiplexing[16]. Moreover, if k_i is greater than $PCR + Mem_{i-1}$ then $k_i - (PCR + Mem_{i-1})$ defines the fraction of a time interval wasted during time cycle i.

Example 7: Assume the system parameters presented in Table 5. Moreover, assume $k = 2$ for the fifth time cycle, $k = 3$ for the seventh time cycle, and $k = 1$ for the remaining cycles. Here the maximum amount of memory and the total period of materialization cannot be calculated directly from Equation 10 and 11. However, by using Equation 15 (instead of Equation 13), a similar table as Table 7 and 8 can be constructed (see Table 9). This table shows that the time required for X to become disk resident is reduced to 9 time cycles $(q = 9)$ as compared to the 12 time cycles required in Example 5. Moreover, only 2 subobjects become memory resident during the fourth and fifth time cycles $(MaxMem = 2)$. The schedule for flushing the subobjects onto the disk clusters is shown in Figure 8. $\Box$

The strategy used to determine the position of time slots in different time cycles is the same as described before. the only difference is that if $k_i > PCR + Mem_{i-1}$ then time cycle i has logical overlap with cycle $i + 1$.

[16]Scheduler may be forced to do multiplexing due to lack of resources.

Figure 9: Inputs and outputs of the algorithm

In Examples 5-7, the k available time slots were allocated vertically. However, the k time slots might be available horizontally: the k allocated time slots are provided in one time cycle and correspond to a single disk cluster ($k \leq U_{Cluster}$). To demonstrate, recall Example 6 and assume that the 2 available time slots are assigned horizontally. Table 10 shows the subobject(s) transfered to disk per time cycle. During the first time cycle, although two time slots are available, only one of them can be utilized. We do not assign $\frac{1}{2}X_1$ to cluster 0 in order to preserve the round-robin assignment of subobjects to the disk clusters. This portion should remain memory resident until the time slot corresponding to disk cluster 1 becomes available. With horizontal allocation of time slots, Equations 12 and 14 remain valid. However, Equations 13 and 15 need to be extended. The reason is that neither k nor k_i available time slots imply that either k or k_i subobjects are transfered to the disk clusters. Moreover, contrary to the vertical assignment, here the subobjects are not transfered to disk sequentially. For example, in Table 10, X_8 becomes disk resident before X_6. Therefore a closed form formula to compute the portion of the object that resides on disks per time cycle (similar to Equation 15) does not exist. This is because it is run-time dependent on the value of k_i and the sequence of the subobjects being transfered. In Appendix A an algorithm is provided to determine the portion of the subobject(s) transfered to disk for a given time cycle i and available time slots (k_i). Obviously, $Disk_i$ is the accumulated amount of these portions and the portions from the previous runs of algorithm for 1 to $i-1$. This computation can replace Equation 15 when time slots are reserved horizontally. Figure 9 shows inputs and outputs of the algorithm.

4 Conclusion and Future Directions

This paper presented the design of a pipelining mechanism that minimizes the latency time of systems based on a hierarchical storage structure. It is novel and different than the pipelining mechanism proposed for parallel relational systems because it ensures a continuous display of audio and video objects. It is general purpose because it can support: 1) different organization of memory, disk, and tertiary storage, 2) tertiary devices with various transfer rates, 3) objects that require different bandwidths, and 4) a system that allocates various resources (memory and disk bandwidth) dynamically based on its load and the availability of resources. It splits an object into s logical slices: (S_1, S_2, ..., S_s). In the best case, it can reduce the latency time of the system as compared to non-pipelining case by the following ratio: $\frac{size(X)-size(S_{X,1})}{size(X)}$. The precise reduction in the latency time is dependent on the architecture, the resources allocated to the mechanism, the bandwidth of the tertiary, and the bandwidth required to support a continuous display of the referenced object. We presented analytical models that compute this latency time based on these input parameters.

	PCR < 1	PCR > 1	
		Multiplexing	Non-Multiplexing
SDF	YES	MAYBE	YES
PDF	MAYBE	YES	MAYBE
IDF	MAYBE	YES	MAYBE

Table 11: Suitability of the proposed paradigms

We have described three pipelining techniques: SDF, PDF, and IDF. While SDF and PDF render an object disk resident, IDF displays an object from the tertiary storage device without storing it on the disk clusters. IDF is appropriate for those objects whose expected future reference is so low that they should not replace other objects that are currently disk resident. However, when PCR $<$ 1, IDF (and PDF) would require the first slice of an object to become memory resident (prior to the display of an object). The value of PCR defines the size of the first slice of an object which in turn dictates the amount of required memory. If memory is a scarce resource then neither PDF nor IDF may be appropriate when PCR < 1. SDF might be more suitable because it minimizes the amount of required memory by using the disks as the staging area.

With PCR > 1, we considered two alternative execution paradigms: multiplexing and non-multiplexing. Multiplexing strives to render PCR $= 1$ with the rate of data delivery from tertiary matching that of display (by introducing tape seeks). In this case, both PDF and IDF are suitable because they require less disk bandwidth and memory as compared to SDF. Non-multiplexing retrieves an object from the tertiary as fast as possible (without introducing seeks). In this case, the object becomes memory resident faster than its consumption rate at a display station. Once again, SDF minimizes the amount of required memory by flushing the subobjects to the disk as soon as possible. PDF and IDF may not be appropriate because they may require substantial amount of memory (depending on the value of PCR).

We intend to extend this study by investigating the design of a scheduler that uses the pipelining mechanism. The scheduler should allocate resources (memory and disk bandwidth) to each request in a manner that maximizes the utilization of resources and minimizes the latency time of the system while ensuring the real-time constraint to support a continuous display. Moreover, it should analyze the system load when rendering decisions. The alternative dataflow paradigms presented in this paper provide the scheduler with a wide variety of choices to optimize the utilization of resources.

5 Acknowledgments

We would like to thank David DeWitt for bringing to our attention the future trends in the area of tertiary storage devices and providing us with the literature that supports these trends. In addition, we would like to thank the anonymous referees for their valuable comments.

References

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

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

[GDQ92] S. Ghandeharizadeh, D. DeWitt, and W. Qureshi. A performance analysis of alternative multi-attribute declustering strategies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, June 1992.

[GDS94] S. Ghandeharizadeh, A. Dashti, and C. Shahabi. A Pipelining mechanism to minimize the latency time in hierarchical multimedia storage managers. Technical Report USC-CS-94-584, University of Southern California, 1994.

[GRAQ91] S. Ghandeharizadeh, L. Ramos, Z. Asad, and W. Qureshi. Object Placement in Parallel Hypermedia Systems. In *Proceedings of the International Conference on 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.

[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.

[MWS93] D. Maier, J. Walpole, and R. Staehli. Storage System Architectures for Continuous Media Data. In *Proceedings of the Foundations of Data Organization and Algorithms (FODO) Conference*, october 1993.

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

[SAD$^+$93] M. Stonebraker, R. Agrawal, U. Dayal, E. Neuhold, and A. Reuter. DBMS Research at a Crossroads: The Vienna Update. In *Proceedings of the International Conference on Very Large Databases*, 1993.

[Sch93] T. Schwarz. High performance quarter-inch cartidge tape systems. *Third NASA GSFG conference on MASS Storage Systems and Technologies*, pages 13–25, February 1993.

[SGM86] K. Salem and H. Garcia-Molina. Disk striping. In *Proceedings of International Conference on Database Engineering*, February 1986.

[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.

Figure 11: Consumed: The main data structure of the algorithm

Appendix A: Algorithm for Horizontal Allocation Figure 10 shows an algorithm to compute the number of subobjects flushed to the disk drives (consumed) per time cycle for horizontal assignment of time slots. Figure 11 shows the main data structure of the algorithm, termed Consumed, when the system consists of 3 clusters and the referenced object consists of 12 subobjects (R=3 and n=12). Each element of this data structure corresponds to a portion of subobject that should be assigned to a cluster. When the algorithm is first invoked, all elements of this data structure are initialized to zero. The value that may occupy each element must be a real number between 0 and 1. It represents a fraction of a subobject that has become disk resident: 1 indicates that the subobject has been flushed in its entirety while 0 indicates that it is either tertiary or memory resident. A number between 0 and 1 indicates the fraction of a subobject flushed thus far. In Figure 11, the subobjects corresponding to each element is shown at the top left position of the element. The algorithm always flushes the first subobject (X_0) to the first disk-cluster. Modifying it to start from a specific disk cluster is trivial. The general steps of the algorithm are as follows:

STEP1: Locate the cluster (C_Cluster) such that during time cycle i, the k_i allocated time slots correspond to C_Cluster. Moreover, determine the smallest index (C_index) that satisfies the following constraint: Consumed[C_Cluster, C_index]< 1. During this step, the last subobject that correspond to C_Cluster that has been partially flushed is located.

STEP2: find the largest index (P_index) of the subobject that has been produced for C_Cluster. This step determines the last subobject that corresponds to C_Cluster that has become either partially or fully memory resident.

STEP3: Flush the subobjects Consumed[C_Cluster, j] where j varies from C_index to either P_index or C_index+k_i whichever is smaller. During this step, the subobjects that have become memory resident are flushed to the disk clusters. The number of flushed subobjects is determined by the number of either available time slots or subobjects that are produced by the tertiary storage device.

Shahram Ghandeharizadeh received the B.S., M.S., and Ph.D. degrees in Computer Sciences, in 1985, 1987, and 1990, respectively, from the University of Wisconsin, Madison.

In September 1990, he joined the Computer Science Department at the University of Southern California, where he is currently an Assistant Professor. One of his current research program involves the design and implementation of a highly parallel object-oriented database machine. This project is studying alternative implementations of next-generation database constructs, declustering algorithms for data intensive applications, and techniques to support applications with irregularly structured data (e.g., multimedia information systems, CAD/CAM, scientific databases). To support this research, the project has implemented the Omega database machine operational on a cluster of workstations (with a magneto optical jukebox serving as a tertiary storage device). One of his main research objective is to demonstrate the feasibility of the techniques presented in this paper using Omega.

Dr. Ghandeharizadeh is a member of ACM and IEEE Computer Society. He received an NSF Young Investigator Award in 1992.

Cyrus Shahabi is a Ph.D. Candidate in the Computer Science department at the University of Southern California. He received his B.S. degree in Computer Engineering from Sharif University of Technology, Tehran, Iran in 1989, and M.S. degree in Computer Science at the University of Southern California in 1993. He participated in the design and the implementation of the Omega database machine. His current research interests include the design and the implementation of multimedia information systems, and support for continuous display of non-linear presentations. He can be reached by e-mail: shahabi@perspolis.usc.edu.

Cyrus Shahabi is a member of ACM and IEEE computer Society.

Ali Dashti is a Ph.D. student in the Computer Science department at the University of Southern California, Los Angeles. He received his B.S. degree from the University Of the Pacific, Stockton, California in 1990, and his M.S. degree from the University of Southern California in 1993, both in Computer Engineering. His current research interests include the design and the implementation of multimedia information systems. He can be reached by e-mail: dashti@perspolis.usc.edu.