On-Demand Data Elevation in a Hierarchical Multimedia Storage Server

Peter Triantafillou*
Multimedia Systems Institute of Crete and
Department of Computer Engineering
Technical University of Crete
Chania, Crete, Greece
peter@mhl.tuc.gr

Thomas Papadakis†
Department of Computer Science
York University
North York, Ontario M3J 1P3
Canada
tom@cs.yorku.ca

Abstract

Given the present cost of memories and the very large storage and bandwidth requirements of large-scale multimedia databases, hierarchical storage servers (which consist of RAM, disk storage, and robot-based tertiary libraries) are becoming increasingly popular. However, related research is scarce and employs tertiary storage for storage augmentation purposes only. This work, exploiting the ever-increasing performance offered by (particularly) modern tape library products, aims to utilize tertiary storage in order to augment the system's performance. We consider the issue of elevating continuous data from its permanent place in tertiary for display purposes. Our primary goals are to save on the secondary storage bandwidth that traditional techniques require for the display of continuous objects, while requiring no additional RAM buffer space. To this end we develop algorithms for sharing the responsibility for the playback between the secondary and tertiary devices and for placing the blocks of continuous objects on tapes, and show how they achieve the above goals. We study these issues for different commercial tape library products with different bandwidth and tape capacity and in environments with and without the multiplexing of tape libraries.

1 Introduction

Future multimedia information systems are likely to contain large collections of delay sensitive data objects (e.g. audio and video) with various lengths and display/play(back) requirements. Some of the main data characteristics which present serious challenges when building multimedia servers are: their large size, their delay-sensitivity, and their high display bandwidth requirements.

Multimedia objects can be very large in size: As one example, 100 minutes of MPEG-2 video may require up to 6 GB of storage. As another example, the annual storage requirements for typical hospital information systems (for data such as echographic images, computerized tomography, digital radiography, and digital angiography) approximate 300 GB. Thus, the storage requirements of future multimedia servers for many different applications will easily exceed several terabytes.

The present cost of memory strongly depends on the type of memory being employed and ranges from \$8.0 per MB of RAM, to \$0.2 per MB of magnetic disk storage, and \$0.004 per MB of magnetic tape storage. On the other hand, the access speeds of memory units point to a different direction. Tertiary storage is characterized by very slow access times, in the order of tens of seconds (or minutes) for optical disk

* Research supported by the European Community under the ESPRIT Long Term Research Project HERMES no. 9141.

† Research supported by the Canadian government under NSERC grant number 0155218, and by the 1996–97 Going Global - STEP program.

Proceedings of the 23rd VLDB Conference
Athens, Greece, 1997

or magnetic tape libraries. Magnetic disks have access times in the order of tens of milliseconds, whereas RAM memories' access times are in the order of tens of nanoseconds. The above are strong arguments for employing a hierarchy of memory technologies including RAM-based *Primary Storage* (PS), magnetic disk-based *Secondary Storage* (SS), and tape- or CDROM-based *Tertiary Storage* (TS). A key idea is to store all objects in the inexpensive tertiary storage, from where they will be elevated to the SS and PS levels before being accessed. This solves the problems owing to the large storage space requirements of multimedia servers and the costs of memory units. A complementary idea is to use the higher levels as a cache for the levels below; all objects will reside permanently on TS, while popular objects, for example, will reside for long periods of time in SS and portions of them will reside for long periods of time in PS. This idea addresses the problems owing to the memories' access times, and to the delay-sensitivity and high display bandwidth requirements of many multimedia objects.

1.1 The Problem

This paper assumes the above framework of a multimedia server based on a hierarchical storage manager. The research reported here addresses the important problem of delay-sensitive (continuous) data elevation, from its permanent place in TS to the higher levels [CT95]. In particular, we concentrate on *on-demand elevation* (i.e., elevation occurs when a display request for the object arrives). On-demand elevation is a non-trivial problem since it must deal with the *bandwidth mismatch* problem: firstly, the display bandwidth requirements of objects are different from the bandwidths of TS and SS devices; and secondly, the sustained (available, effective) TS and SS bandwidths vary with time, depending on the requirements of the current workload (e.g., whenever multiplexing is employed).

One may assume that, at a steady state, the popular objects will be residing in SS, with perhaps portions of them residing in PS. In this framework, the key resources which may limit the throughput of the system are the SS bandwidth and the PS buffer space. The key high-level goal of this work is to harness the (ever-)increasing bandwidth offered by modern tape library products in order to improve the overall system performance. This is not an easy task because, despite the very high transfer rates of modern tape drives, the average access cost in tape libraries remains very high due to the high costs for robotic movements and head positioning delays.

We focus on requests for delay-sensitive data elevation from TS and SS to higher levels, with the goals of achieving SS bandwidth savings, hiccup-free displays of streams (i.e., continuous objects), and low start-up latencies. The saved SS bandwidth can be used to accept and serve additional continuous or discrete (i.e., non-delay sensitive) data requests, improving thus the system throughput for continuous requests and/or the response time of discrete requests. At the same time, we handle the bandwidth mismatch problem during data elevation, in a manner that allows the above savings in SS bandwidth while not requiring any extra PS buffer space. The same issues are also addressed for the cases with and without the multiplexing of the tape library.

The remainder of the paper is structured as follows. In Section 2 we will look closely at *Hierarchical Storage Management Systems* (HSMS), identifying their key properties and technologies and refer to related work. In Section 3 we will present "Alternate Play", a novel algorithm for on-demand data elevation through the levels of the hierarchy and we will discuss its benefits with respect to SS bandwidth savings and its compromises with respect to PS buffer space. Subsequently, we will contribute a novel technique which places the blocks of a delay-sensitive multimedia object on the tertiary storage media in a manner which alleviates the need for extra PS buffer space, while still achieving the same SS bandwidth savings as in the Alternate Play algorithm. Then we will contribute the notion of strips of streams, the use of which makes our techniques applicable even when the display bandwidth requirements are larger than the available TS bandwidth (e.g., for low-end tape drives). In Section 4 we revisit these issues, only now assuming that the tape library is multiplexed across streams (to avoid experiencing unacceptably-long start-up latencies). In Section 5 we revisit the same issues, only now the assumption of a fixed and known, a priori, multiprogramming degree is removed. Finally, in Section 6 we will present the conclusions of this work.

2 Hierarchical Storage Management Systems

In this section we discuss HSMSs, paying attention to their specifications and their functionality in order to gain the relevant insights and put the contributions of this work in context. A summary of the HSMS's relevant parameters and some typical values is given in Table 1. Recall that HSMSs consist of a storage hierarchy of three different levels: PS, SS, and TS.

An appropriate candidate for SS appears to be arrays of magnetic disks [PGK88]. Given the high bandwidth requirements of many multimedia objects and the typical, relatively low, effective bandwidth of magnetic disks (nominal bandwidths are in the range of

Table 1: Storage and display parameters

symbol	explanation	typical values
b_D	display bandwidth (consumption rate)	0.2 – 1 MB/sec (eg, MPEG2)
B	# of blocks in an object	varies; user-defined
s	block size	varies; user-defined
d	"time unit" = display time for 1 block	varies; depends on s
—	SS bandwidth	3 – 40 MB/sec
—	disk capacity	1 – 20 GB
—	SS capacity	4 – 400 GB
b_T	tape drive bandwidth	0.5 – 20 MB/sec
—	# of tape drives	1 – 64
e	exchange time	6 – 30 sec
c	switch time = exchange (if needed) + search	1 – 60 sec
—	# of tape cartridges	30 – 48,000
—	tape capacity	5 – 150 GB
—	TS capacity	0.5 – 190 TB
r	b_T/b_D	0.5 – 100
j	# of jobs multiplexed	varies
t	# of blocks in a "time slice" (in multiplexing)	varies

4–7 MB/sec), striping techniques [SGM86, BMGJ94, TF97] are likely to prove beneficial, since, when striping objects across D disks, the effective SS bandwidth for these objects is D times the effective bandwidth of a single disk. Currently, average seek and rotational delays are approximately 10-15 and 4-8 milliseconds, respectively.

With respect to TS, technological developments have led to the emergence of robot-based magnetic-tape and optical-disk libraries, making them the best candidate for the TS devices. Tape libraries consist of a series of shelves (storing a number of tape cartridges), a number of tape drives onto which the tapes must be loaded before their objects can be accessed, and a number of robot arms (usually 1) which are responsible for loading and unloading the tapes to and from the drives. Despite TS's high data transfer rates (see Table 1) these devices remain comparatively slower than magnetic disks due to the high exchange costs (unload a tape, put it on the shelf, get another tape, load it) and to the cost of searching within a tape (proceeding at a pace of less than 1 GB/sec).[1]

The conceptual model of a HSMS is illustrated in Figure 1. The conceptual model shows the three levels of the HSMS. Data is *elevated* one level at a time. In the rest of this paper, the term "elevate" has been reserved for "elevate from TS to SS".

Figure 2 illustrates the physical architectural view of the HSMS. The key feature that must be noted here is that PS serves as an intermediate staging area between the TS and the SS device. This fact implies that an object is not required to be SS-resident in order to be displayed; it can also be displayed from TS [KDST95].

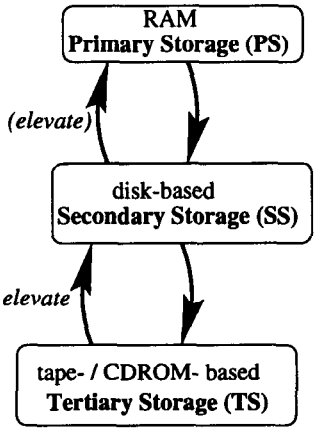

Figure 1: Conceptual model of a Hierarchical Storage Management System (HSMS)

However, note that the bandwidth of TS tape drives is typically significantly greater than the display bandwidth of objects. Thus, playing objects directly from TS can create serious PS buffer space problems. As a result, a much more preferable choice is to first elevate the object from TS to SS and then start the playback procedure issuing *retrieval* requests to SS for the blocks to be displayed [GS94, KDST95]. Furthermore, given that very few users (if any) would tolerate high response times, the playback process must be started

[1] Optical disk libraries (also called jukeboxes) have a similar architecture.

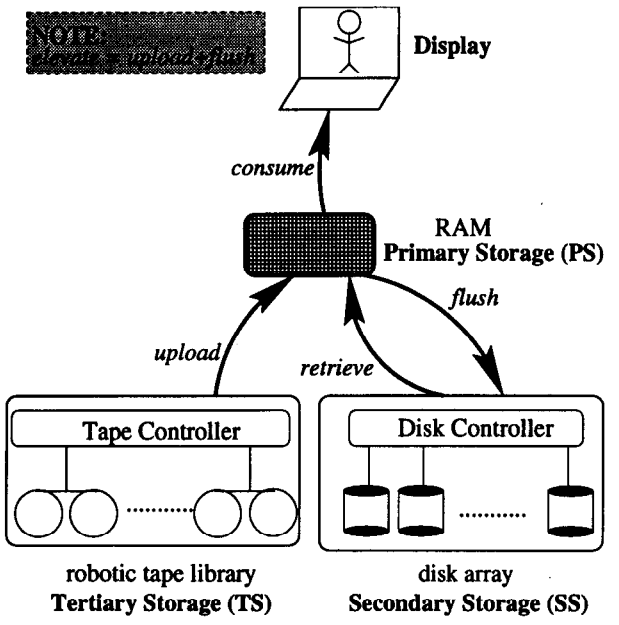

Figure 2: Physical model of a Hierarchical Storage Management System (HSMS)

immediately after enough data has been elevated to SS; hence, typically, the retrieval operations executing on behalf of the playback process are executed in parallel with the elevate operations which move the future blocks of the object to SS [GDS95].

3 On-demand Data Elevation Without Multiplexing

In this section we first discuss the conventional method for data elevation and play, and then we present three novel elevation methods.

3.1 Play-from-TS and Conventional Play

As discussed in the last section, an object can be played either from TS or from SS. One may *Play-from-TS* an object by issuing *upload* requests to TS. The uploaded blocks of the object are placed into PS buffers, from where they are subsequently *consumed*:[2] either they are transmitted over a network to a remote client, or they are displayed to a local client.

Recall that, (i) playing an object from TS can create serious PS buffer space problems; (ii) a much more preferable choice, thus, is to first elevate the object from TS to SS (that is, physically *upload* it from TS to PS, and then *flush* it from PS to SS), and subsequently *Play-from-SS* the object, by issuing *retrieval* requests to SS; and (iii) in order to decrease start-up latencies, the retrieval requests issued by the display process are executed in parallel with the elevation of future blocks of the object from TS to SS.

Let us refer to the above procedure (i.e., elevate from TS to SS, and simultaneously play from SS) which represents the state of the art, as the *Conventional Play* method. The obvious advantage of the Conventional Play method is that it requires no PS buffer space.[3] On the other hand, its obvious deficiency, is that SS is additionally taxed as a result of the parallel execution of flush and retrieval operations, reducing thus the available SS bandwidth significantly. Since in our target environment we expect SS bandwidth to be one of the (two) scarcest resources, the above observation must be noted seriously. In the following, we will present some approaches aiming to alleviate this problem.[4]

3.2 Alternate Play

An initial attempt to overcome the shortcomings of the conventional method centers on the following idea: uploaded blocks from TS can be maintained in PS buffers and made available to the playback process from them. This can save significant SS bandwidth since neither a flush to, nor a retrieval from, SS is required.

Of course, blocks belonging to popular objects may still be flushed to SS, in addition to maintaining them in PS buffers from where they will be consumed. This alteration of the Play-from-TS method (i.e., Play-from-TS, and then flush to SS), when compared with the Conventional Play, saves SS bandwidth for the current request (since no retrievals from SS are needed), and both SS and TS bandwidth for future requests for the same object (since no uploads or flushes for the SS-resident blocks are needed).

Despite these unquestionable benefits, there is an obvious concern regarding the amount of PS buffer space which is needed to realize the aforementioned bandwidth savings. For example, if $r = 2$, the PS space requirements are 50% of the size of the entire object to be displayed (since when the last block is uploaded, only half of the blocks will have been consumed). In general, the PS space requirements are $(r-1)/r$ of the object's size. Given that PS buffer space is another scarce system resource, care must be taken to use it wisely. In an effort to reconcile this trade-off between the PS space requirements and the SS bandwidth savings, we can use hybrid techniques,

[2]The terms "consume", "display", "playback" and "play" will be used interchangeably in this paper.

[3]Throughout the paper, "PS buffer space requirements" will refer to the *maximum* number of PS buffers (each holding 1 block) required at any instance during the display of an object. I/O buffers are excluded.

[4]Throughout the paper, "Algorithm X achieves x SS bandwidth savings" will mean "the total SS bandwidth (traffic) required for the display of an object initiated by Algorithm X, is x of the SS bandwidth required by the Conventional Play algorithm".

so that for some blocks the Conventional Play method — essentially: Play-from-SS — is employed, while for other blocks a Play-from-TS method is followed. We refer to the newly-proposed method as the *Alternate Play* method.

To illustrate the Alternate Play method, we give pseudo-code for it (Algorithm 1), for the $r = 2$ special case. If a block does not exist (e.g., if $r = 2$ and $B = 12$), an "empty block" is read off the tape. `time unit` refers to the time required to display one block. The given algorithm uses routines to `upload` an indicated block to a PS buffer, or to `elevate` (i.e., upload and then flush) an indicated block to SS. The `parbegin/parend` construct is intended to indicate that the display (`consume/Play-from-SS`) of a block occurs in parallel with the readings (`elevate` and `upload`) of the next two blocks from TS. The algorithm terminates when all blocks have been read off the TS. The display of the requested object continues by consuming all remaining blocks in strict alternation from PS and SS. At the end, half of the blocks will have been played from TS and half from SS.

Algorithm 1 Alternate Play example ($r = 2$)

INPUT: Object's blocks in TS

```
next_upld_blk := B1
next_displ_blk := B1
upload(next_upld_blk++)
for time unit i:= 1 to ⌈(B-1)/2⌉ do
    parbegin
        1. if i = odd
               then consume(next_displ_blk++)
               else Play-from-SS(next_displ_blk++)
        2. elevate(next_upld_blk++)
           upload(next_upld_blk++)
    parend
```

An example of the algorithm's action on a 13-block object is given in Table 2. Newly arriving blocks have been underlined.

Two features of the Alternate Play algorithm must be noted. First, during its execution (i.e., during the first $\lceil \frac{B-1}{2} \rceil \simeq B/2$ time units of the object's display) the SS bandwidth requirements of even time units are twice the SS bandwidth requirements of odd time units; for the remaining $\lfloor \frac{B+1}{2} \rfloor \simeq B/2$ time units, the SS bandwidth requirements are equal to those of the odd time units. Second, during the algorithm's execution, every two time units the number of PS buffers (needed to hold the Play-from-TS blocks) increases by 1; at the end of the algorithm's execution, half of the $\simeq B/2$ Play-from-TS blocks will still be in PS. It thus follows that the PS space requirements are 25% of the object's size.

Table 2: Example of Alternate Play ($r=2$; $B=13$)

time unit	(elevated) to SS	(uploaded) to PS	block displayed
0	—	B1	—
1	B2	B3	B1
2	B4	B3 B5	B2
3	B4 B6	B5 B7	B3
4	B6 B8	B5 B7 B9	B4
5	B6 B8 B10	B7 B9 B11	B5
6	B8 B10 B12	B7 B9 B11 B13	B6

3.3 Generalizing Alternate Play

Algorithm 1 can be generalized to an Alternate Play algorithm, handling arbitrary r ratios and offering space-performance trade-offs as follows: In every time unit, r blocks are read from TS. From these, k blocks are uploaded to PS and the rest $r-k$ blocks are elevated to SS, for some (arbitrarily/appropriately chosen) $k = 1, 2, \ldots r-1$. This gives us a family of Alternate Play algorithms, one for each different value of k. The impact of the choice of k on the PS space requirements and on the SS bandwidth savings is examined in [TP97].

The issue of a non-integer r is also examined in [TP97].

In addition to the above described spectrum of Alternate Play algorithms (one algorithm for each value of k), we will contribute other techniques which can be used to further reconcile the conflicting goals of high SS bandwidth savings and low additional PS buffer requirements.

3.4 Alternate Play With A Twist

First, note that the high PS buffer requirements of the Alternate Play method are due to the long time that a PS buffer is dedicated to holding a particular Play-from-TS block. Next, observe that this long time is dependent on r: if r is large, then the Play-from-TS blocks will occupy a PS buffer for a long time (since they will come into a PS buffer too far ahead of their consumption times). In trying to attain further reductions in the PS space requirements, our key idea is that, by altering the order with which the blocks of an object are recorded on the TS media, large values of r (i.e., bandwidth mismatches between the TS and the display) can be accounted for in a way that reduces the occupancy time of PS buffers by Play-from-TS blocks.

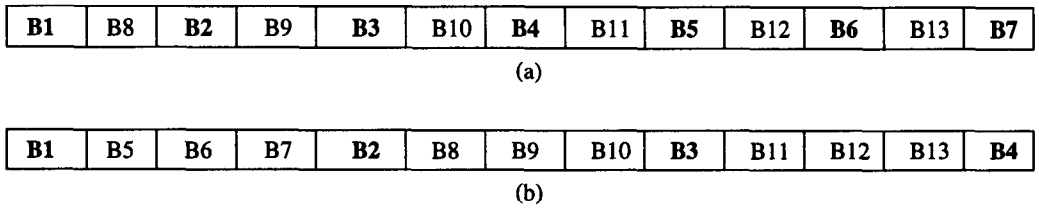

Figure 3: Twisted Placements of a 13-block object when (a) $r=2$ and (b) $r=4$

Algorithm 2 is a placement algorithm, which, when given as inputs B and r, determines a *twisted*

Algorithm 2 Twisted Placement

INPUT: B, r
OUTPUT: Twisted sequence of object's blocks (to be placed in TS)

$i := 1$
$n := 1$
$N[i++] := n++$
$f := \lceil B/r \rceil + 1$
while $(i \leq B)$ **do**
  for j:=1 **to** r-1 **do**
    $N[i++] := f++$
  $N[i++] := n++$

placement/ordering of an object's blocks {Block[1], Block[2], ..., Block[B]} as {Block[N[1]], Block[N[2]], ..., Block[N[B]]} on a tape. (If a block does not exist, an "empty block" is placed in the twisted sequence.) Examples of the twisted placements when $B=13$ and $r=2$ or $r=4$ are given in Figure 3.

One may immediately see that the produced twisted placement is such that when the object's blocks are read off the tape sequentially, every r-th block will be uploaded into PS precisely before the time it has to be consumed.[5] These B/r blocks can thus be played-from-TS without any additional PS buffer space requirements; the remaining blocks will be elevated to SS, and will be played-from-SS when needed. Algorithm 3 materializes the above idea. The notational conventions of Algorithm 1 have been used here, too. APWAT algorithm terminates when all blocks have been read off the TS. The display of the requested object continues by consuming one more block from PS, and all remaining blocks exclusively from SS. At the

[5] Actually, any placement sequence having the 1st, 2nd, 3rd, 4th,... blocks of an object in locations 1, r+1, 2r+1, 3r+1,... of a new sequence, will be as good as the twisted sequence produced by Algorithm 2. For example, in the two given twisted placement examples, it would be sufficient to keep the bold-faced blocks in the given locations; the rest of the blocks could be arbitrarily intermixed.

Algorithm 3 Alternate Play With A Twist (APWAT) for $r = 2$

INPUT: Twisted placement of object's blocks in TS

next_upld_blk := B1
next_displ_blk := B1
upload(next_upld_blk++)
for time unit i:= 1 **to** $\lceil \frac{B-1}{2} \rceil$ **do**
  parbegin
    1. consume(next_displ_blk++)
    2. elevate(next_upld_blk++)
       upload(next_upld_blk++)
  parend

Table 3: Example of Alternate Play With a Twist ($r=2$; $B=13$)

time unit	(elevated) to SS	(uploaded) to PS	block displayed
0	—	B1	—
1	B8	B2	B1
2	B8 B9	B3	B2
3	B8 B10 B10	B4	B3
4	B8 B9 B10 B11	B5	B4
5	B8 B9 B10 B11 B12	B6	B5
6	B8 B9 B10 B11 B12 B13	B7	B6

end, $1/r$ of the blocks will have been played from TS, and the remaining $(r-1)/r$ from SS.

An example of APWAT's action on the $r=2$ twisted placement of Figure 3(a), is shown in Table 3. Again, newly arriving blocks have been underlined.

One can immediately see that the SS bandwidth requirements remain constant (r–1 blocks per time unit) throughout the algorithm's execution; the display of the remaining blocks requires a constant bandwidth of 1 block per time unit. The total SS bandwidth savings of the APWAT method, for each stream, is $1/r$ (since $1/r$ of the blocks for each stream are played from TS). Clearly also, the additional PS buffer requirements are zero.

Table 4: Primary (PS) and Secondary (SS) Storage requirements

	PS requirements	SS bandwidth savings
Conventional	0	—
Alternate Play	$Bk(r-1)/r^2$	k/r
APWAT	0	$1/r$

The improvement achieved by of our newest Play algorithm is apparent from Table 4 (see [TP97] for details), summarizing the examined algorithms' characteristics. The case of non-integer r is again examined in [TP97].

3.5 Data Already on Disk & Strips of Streams

Although not mentioned explicitly so far, all previously described algorithms work only if $r \geq 1$. One may notice that a $r < 1$ situation may indeed arise (e.g., low-end products — see Table 1). One may also immediately see that the $r \geq 1$ restriction is obviously not a deficiency of our algorithms: if $r < 1$, and an object resides exclusively on TS, no algorithm can display it without hiccups, or without a long response time.

If $r < 1$, and the object resides in SS as well, one may use the ideas presented in the previous sections to display the object in a manner requiring less SS bandwidth than the obvious Play-from-SS algorithm, while still requiring zero PS buffer space. A *strip of a stream* is a subset of the blocks of a (stream) object. We define a $\frac{1}{r}$-*strip* (*of a stream*) to be the strip consisting of every $\frac{1}{r}$-th block of an object, except for the 1st block of the object. For example, the 2-strip is [B3, B5, B7, B9, ...], the 3-strip is [B4, B7, B10, B13, ...], etc. We store such a $\frac{1}{r}$-*strip* (as a partial replica of the object) permanently in TS.

One may now see that a continuous uploading of TS blocks results in each such block being brought into PS precisely before it has to be consumed. A trace of this Strips of Stream Play algorithm is given in Table 5. Note that the algorithm achieves the same bandwidth savings and has the same extra PS buffer space requirements as the APWAT algorithm. See [TP97] for more details.

4 Data Elevation With Known Multiplexing Degree

In many environments, consisting of high-end tape library products with very large bandwidth, the tertiary storage level of a multimedia server will be satisfying

Table 5: Example of Strips of Stream Play ($B = 12$; $r = 0.5$; 2-strip = [B3, B5, B7, B9, B11])

time unit	uploaded from TS strip	block displayed
1–2	B3	B1 B2
3–4	B5	B3 B4
5–6	B7	B5 B6
7–8	B9	B7 B8
9–10	B11	B9 B10
11-12	—	B11 B12

more than one requests simultaneously. In order to avoid unacceptably high start-up latencies, it is desirable for the TS to be multiplexed. When the TS bandwidth is multiplexed across several concurrent streams, the effective TS bandwidth $b_{T,\text{eff}}$ and the effective ratio r_{eff} are reduced. We will assume a Round Robin scheduling discipline for the multiplexed requests.

A moment's thought reveals that using, say, the APWAT algorithm on the twisted sequence for r, does not work. Consider, for example, the twisted sequence for $r = 4$, given in Figure 3(b). If the number of multiplexed jobs (multiplexing degree) is $j = 2$, i.e., if $r_{\text{eff}} = 2$ for each of the two multiplexed jobs,[6] the APWAT algorithm, with Round Robin scheduling of a time slice equal to d, will suffer from hiccups. For instance, while the B1's of the two objects are displayed, APWAT will elevate/upload the next 4 blocks of the first object (which will take 1 time unit) and it will then switch to elevate/upload the next 4 blocks of the second object (which will also take 1 time unit). Thus, it is apparent that the displays will starve.

Another approach, is to use the APWAT algorithm on the twisted sequence of Figure 3(a), instead of Figure 3(b). For our $r = 4$, $j = 2$, $r_{\text{eff}} = 2$ example, the APWAT algorithm, with Round Robin scheduling of a time slice equal to $d/2$, will display both objects without hiccups. It will elevate/upload a pair of blocks of the first object (which will take 1/2 time unit), and it will then switch to elevate/upload a pair of blocks of the second object (which will take another 1/2 time unit). This method (playing boldfaced blocks from TS, and the rest from SS) requires no extra PS buffers, and it achieves 50% SS bandwidth savings (since half of the blocks are played from TS).

A crucial factor to the last approach's success (besides the use of the twisted sequence for $r_{\text{eff}} = \frac{r}{j} = 2$) was the fact that its time slice was the time needed to upload 2 blocks. To see why this is indeed crucial, consider, for example, the extreme case of a time slice

[6] For now, for readability purposes, we assume no exchange overhead.

equal to the time needed to upload 12 blocks: the last approach could not achieve 50% SS bandwidth savings with no extra PS buffers (for the same sequence of $r_{\text{eff}} = 2$). This suggests that small time slices are better than big ones, and one should thus use time slices as small as possible.

Unfortunately, as we discussed earlier, the above approach is over-simplified, since there is a *switch cost* c (which includes, possibly, a tape exchange cost, plus a search within a tape) associated with each Round Robin's switch from one object to another. Therefore, the just derived minimum time slice will not be sufficient to display the objects without hiccups. Nevertheless, we are now in a position to describe a general approach and to derive the minimum time slice.

4.1 APWAT with Round Robin

The basic idea is to split each object's blocks into groups of t blocks each (called *t-tuples*), for some appropriate value of t, as follows: the first t-tuple will contain the blocks {B1, ..., Bt}; the second t-tuple will contain the blocks {B(1+t), ..., B(2t)}, etc. In order to process (upload, elevate, display) each t-tuple in the most efficient way, we will use a twisted placement of the blocks within each t-tuple for r. The desired twisting here is slightly different than the one suggested by Algorithm 2. The Twisted Placement, produced by Algorithm 2, places the first blocks of an object in locations 1, $r+1$, $2r+1$, $3r+1$, etc. The Twisted Placement that is useful in our case places the first blocks of a tuple in locations r, $2r$, $3r$,.... within the tuple. Such an arrangement of an object's blocks will be called a *(t,r)-organization*, or simply a *t-organization*.

An example is shown in Figure 4. The objects will be displayed by multiplexing them in a Round Robin fashion: each time slice will process one t-tuple by employing the APWAT algorithm. As an example, consider the (8,4)-organization of Figure 4 (implying that $r = 4$). While displaying B1 of the first object, the Round Robin algorithm will elevate [B4, B5, B6] and it will upload B2. While displaying B2, it will elevate [B7, B8, B9] and it will upload B3. It will then start displaying B3, and it will switch to a second object (processing its first 9 blocks), possibly then to a third object (also processing its first 9 blocks), etc. During the uploads/elevations of the blocks of the second, third, etc., objects, the display process of the first object will finish off consuming B3, and it will subsequently start playing B4, B5, B6, B7, B8, B9 from SS. To guarantee a play of all objects with no hiccups, Round Robin should return to the first object before B9 has been displayed.

The last remark can be used to derive the smallest allowable value of t, given r and j. As explained, the time required to read j t-tuples (one tuple from each object) should be less than or equal to the time needed to display the t blocks of the tuple read j time slices earlier. Since the time needed to upload a t-tuple is ts/b_T, or td/r, and the switch cost is c, it follows that the time required to read j t-tuples is $(td/r+c)j$, or $(ts/b_T + c)j$. Clearly also, the time needed to display t blocks (of one object) is td. Thus

$$td \geq \left(\frac{td}{r}+c\right) j = \left(\frac{ts}{b_T}+c\right) j \qquad (1)$$

should hold. This allows us to derive the minimum allowable value of t, given the bandwidth ratio r (a hardware characteristic), the time unit d — or equivalently (since $s = db_T/r$): given the block size s — and the number j of multiplexed jobs:

$$t_{\min}(j) = \left\lceil \frac{cjr}{d(r-j)} \right\rceil = \left\lceil \frac{cjb_T}{s(r-j)} \right\rceil \quad \text{if} \quad d > \frac{s}{b_T} j. \qquad (2)$$

Example 1: Consider an object consisting of 12,000 blocks, each 0.5 MB, residing on a TS having bandwidth 20 MB/sec and exchange cost 10 sec. Furthermore, assume that the time needed to display one block is 1 sec. These figures imply that $r = 40$. Then, eqn. (2) says that the display of $j = 20$ such objects can be multiplexed using APWAT with Round Robin if each object is split into at most 30 t-tuples of at least 400 blocks each. ■

Inequality (1) may be seen under a different angle. If we assume that the size t of each tuple has been (somehow) fixed, it gives us the maximum number of jobs that can be multiplexed as

$$j_{\max}(t) = \left\lfloor \frac{rtd}{td + cr} \right\rfloor = \left\lfloor \frac{tsr}{ts + cb_T} \right\rfloor \qquad (3)$$

Example 2: Given the configuration described in Example 1, eqn. (3) says that *if* each object is split into 30 groups of 400 blocks each, then *up to* 20 such objects can be multiplexed. ■

In the APWAT with Round Robin algorithm, given a certain multiplexing degree j, if one chooses $t = t_{\min}$ as given by (2), then, clearly, the PS buffer requirements are zero (recall our earlier example of the (8,4)-organization). Clearly also, the SS bandwidth savings are $1/r$ for each of the j jobs. Thus, if each of the j objects consists of B blocks, then APWAT with Round Robin will result in SS bandwidth savings of jB/r blocks, when compared with the Conventional algorithm (elevating all blocks to SS, and playing them from there).

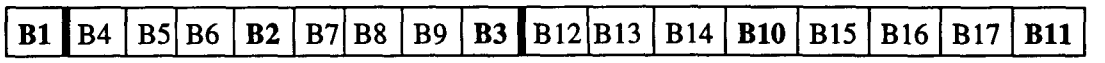

Figure 4: (8,4)-organization of a 17-block object

4.2 Start-Up Latency Considerations

In the algorithms of the previous section, the issue of the start-up latency was not addressed in detail since it was not important (we were assuming only one display request at a time: start-up latency was equal to the time needed to upload the 1st block of the object). However, in a multiplexed environment, this issue deserves more attention. Assuming that all requests for the j objects arrive simultaneously, the display of the 1st object will start with a delay of s/b_T (i.e., after B1 of the 1st object is uploaded), the display of the 2nd object will start with a delay of $s/b_T + ts/b_T + c$ (i.e., after the 1st t-tuple of the first object is uploaded, TS has switched to the beginning of the 2nd object, and B1 of the 2nd object is uploaded), etc. Therefore, the display of the j-th object will start with a delay of

$$\text{delay\_time} = \frac{s}{b_T}(1+(j-1)t) + c(j-1) \qquad (4)$$

The precise effect of the choice of t now becomes more clear. Given a certain multiplexing degree j, if one chooses $t > t_{\min}$, the start-up latency will be higher (as implied by (4)), but the demand of TS usage will be lower (since when the time slice of the 1st object arrives again, not all of its blocks, uploaded during its previous time slice, will have been consumed), allowing it to be used for other purposes. Nonetheless, the display of the object will be done correctly (i.e., without hiccups), the PS space requirement will still be zero, and the total SS bandwidth savings will still be jB/r for j jobs.

5 Data Elevation With Unknown Multiplexing Degree

The algorithms of the previous section work if the number j of multiplexed jobs in known in advance. In this section, we will briefly discuss methods of overcoming this limitation.

For objects residing exclusively on TS, and assuming of course that $r > 1$, (2) and (3) imply that if $j > r$, then no multiplexing of all j display requests is possible. We may thus assume that a known realizable upper bound on the maximum value of $j_{\max}$ for the system. Obviously also, $j \geq 2$ is also true.

In order to display j objects, for arbitrary $2 \leq j \leq j_{\max}$, it suffices to use the $t_{\min}(j_{\max})$-organization. This is so, because, according to (2), $t_{\min}(j_{\max}) \geq t_{\min}(j)$. The discussion now of the last paragraph of Section 4.2 implies that using the $t_{\min}(j_{\max})$-organization will work correctly, without changing the attained SS bandwidth savings of jB/r SS blocks or the required zero PS buffer space.

The only drawback of the last proposal is its increased start-up latency. If startup latency is of great importance, it can be traded-off with additional TS media storage space, as follows: Create and store a $t_{\min}(j_1)$-organization, a $t_{\min}(j_2)$-organization, etc., for various values of $j = j_1, j_2, \ldots$. At run time, given the current multiplexing degree j_{curr}, use the smallest such $t_{\min}(j_i)$ value which is larger than $t_{\min}(j_{curr})$. Each used $t_{\min}(j_i)$-organization is a *full* replica of an object in TS. Frequently, the workload can be guessed, using past experience, and this can be used as a guide to decide which replicas to create.

If $r < 1$ and the objects to be displayed are (partially) residing in SS, then strips of streams can prove useful, as discussed in Section 3.5. The idea is to make every t-tuple a $\frac{1}{r}$-strip. Given r, c and d (or equivalently: s), the $t_{\min}$ and $j_{\max}$, for arbitrary j, and t respectively, are given in [TP97].[7] Unlike the case of non-SS residency (discussed earlier in this section), if one uses Strips of Streams, the required space overhead is significantly smaller. For example, creating a 3-strip, a 6-strip, a 12-strip, a 24-strip, ..., has a total space requirement (for *all* strips) at most $\frac{B}{3}(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots) \leq \frac{2B}{3}$, i.e., at most two-thirds of the space of the entire object. Each of these strips achieves 1/3, 1/6, 1/12,... SS bandwidth savings, correspondingly.

Depending on the tapes' capacities and on the number of drives, it may be possible to do the following: Place all 3-strips of the (say, 20) most popular movies on one 7GB tape (recall that 20 MPEG-2 200KB/sec, 90-minute, movies require about 21GB). Additionally, use slightly larger tape(s) to store additional k-strips. The "3-strips tape" may be "permanently" mounted on one of the drives; this way, the switch cost will only be attributed to the head positioning delays (e.g., about 3.5 sec on average for a 7GB tape searching at a 1 GB/sec speed), and thus the very expensive robotic movements can be avoided when using these strips to multiplex (some of) the most popular movies.

[7] Even when the objects reside in SS and $r > 1$, using the "strips of streams" partial replicas in TS and the above ideas we can still attain significant SS bandwidth savings, with zero additional PS buffer space [TP97].

6. Conclusions

In this paper we have addressed the problem of continuous data elevation in multimedia servers which are based on HSMSs; a problem which, in our view, has not received adequate attention.

We first contributed the notion of alternating the playback of delay-sensitive data between the TS and the SS and discussed how this idea can save significant SS bandwidth (but also pointed out that it requires non-zero buffer space). Subsequently, we contributed the Twisted Placement algorithm with a companion play algorithm, called Alternate Play With A Twist; the Placement algorithm determines the proper placement/recording order for the blocks of objects on the tapes so that the Play algorithm achieves the same SS bandwidth savings as before but with zero additional PS buffer space requirements this time.

Subsequently, we contributed the notion of strips of streams, which are special partial replicas of stream objects, residing permanently in TS, and which consist of the blocks which are to be played from TS. Strips of streams allow the previous contributions to be enjoyed even when (1) the bandwidth of TS is smaller than the display bandwidth of objects; and (2) the objects also reside in SS (which will be the case for the most popular objects).

Later we considered the subject of multiplexing and we derived algorithms which employ the previously-developed techniques to continue offering SS bandwidth savings, at no additional PS buffer space. We presented an algorithm showing how to store the stream on tapes so that a high multiplexing degree j is maintained. For each of the j jobs the algorithm continues to achieve the aforementioned savings; thus, the total savings are significantly greater.

For many multimedia servers we expect that the data objects will exhibit skewed access distributions (e.g., popular movies in a movie-on-demand application). We further expect that for a large majority of cases these objects will have been uploaded and will reside in SS by the time the next request arrives. In such scenarios, the bandwidth of SS is the most critical resource and the resource most likely to cause bottlenecks. In a sense, the aforementioned contributions suggest the collaboration of TS and SS in order to improve the system's throughput. The significance here is that the essence of our proposal aims at using the TS of a HSMS, not simply for storage augmentation reasons (as its traditional role would indicate) but for bandwidth augmentation reasons as well. This is the high level contribution of this research.

References

[BMGJ94] Steven Berson, Richard Muntz, Shahram Ghandeharizadeh, and Xiangyu Ju, *Staggered striping in multimedia information systems*, Proc. ACM SIGMOD Conference, Minneapolis, MN (ACM SIGMOD Record, 23(2), June 1994), May 1994, pp. 79–90.

[CT95] Stavros Christodoulakis and Peter Triantafillou, *Research and development issues for large-scale multimedia information systems*, ACM Computing Surveys **27** (1995), no. 4, 576–579.

[GDS95] Shahram Ghandeharizadeh, Ali Dashti, and Cyrus Shahabi, *A pipelining mechanism to minimize the latency time in hierarchical multimedia storage managers*, Computer communications **18** (1995), no. 3, 170–184.

[GS94] Shahram Ghandeharizadeh and Cyrus Shahabi, *On multimedia repositories, personal computers, and hierarchical storage systems*, Proc. ACM Multimedia Conferece, 1994.

[KDST95] Martin G. Kienzle, Asit Dan, Dinkar Sitaram, and William Tetzall, *Using tertiary storage in video-on-demand servers*, COMPCON'95, San Francisco, CA, IEEE Computer Society Press, March 1995, pp. 225–233.

[PGK88] David Patterson, Garth Gibson, and Randy H. Katz, *A case for redundant arrays of inexpensive disks (RAID)*, Proc. ACM SIGMOD Conference, Chicago, IL (ACM SIGMOD Record 17(3), Sept. 1988), June 1988, pp. 109–116.

[SGM86] Kenneth Salem and Hector Garcia-Molina, *Disk striping*, Proc. International Conference on Data Engineering, Los Angeles, CA, February 1986, pp. 336–342.

[TF97] Peter Triantafillou and Christos Faloutsos, *Overlay striping and optimal parallel I/O in modern applications*, *Parallel Computing Journal*, Special Issue on Parallel Data Servers and Applications, to appear. (Available from the `http://www.ced.tuc.gr/hermes` tech report series.), 1997.

[TP97] Peter Triantafillou and Thomas Papadakis, *On-demand data elevation in a hierarchical multimedia storage server*, Journal version, in preparation, 1997.