Buffer Sharing Schemes for Continuous-Media Systems*

Dwight J. Makaroff and Raymond T. Ng
Department of Computer Science
University of British Columbia
Vancouver, B.C., V6T 1Z4
Canada

Abstract

Buffer management in continuous-media systems is a frequently studied topic. One of the most interesting recent proposals is the idea of buffer sharing for concurrent streams. As analyzed in [8], by taking advantage of the temporal behaviour of concurrent streams, buffer sharing can lead to a 50% savings in total buffer space. In this paper, we study how to actually implement buffer sharing. To this end, we develop the CES Buffer Sharing scheme that is very efficient to implement, and that permits savings asymptotically very close to the ideal savings predicted by the analysis in [8]. We show that the CES scheme can operate effectively under varying degrees of disk utilizations, and during transition periods when the number of concurrent streams changes. We also demonstrate how the scheme can be further improved, particularly for situations when the number of concurrent streams is small. In ongoing work, we will integrate the proposed scheme into a distributed continuous-media file system which is under development at the University of British Columbia.

1 Introduction

The advances in networking and storage technologies in the past decade have made multimedia computing possible. Providing effective multimedia support in information systems has naturally become a topic of great interest and practical value. For a multimedia information system to work well, however, it must deal with two major properties of or challenges presented by multimedia data. First, audio and video data are delay-sensitive. As recording and playback of video and audio data are continuous operations, once an information system starts displaying audio or video data, it must guarantee that enough resources are

*Research partially sponsored by NSERC Grants OGP0138055 and STR0134419, IRIS-2 Grants HMI-5 and IC-5, and CITR Grant on "Distributed Continuous-Media File Systems."

allocated so that the *continuity* and real time requirements are not violated. Second, (even compressed) audio and video data consume large amounts of system resources – primarily storage space and bandwidth.

Many excellent studies regarding the storage and retrieval of audio and video data have been conducted, such as those reported in [1, 2, 3, 4, 8, 9, 10, 12, 13, 15]. In particular, [1, 3, 4, 9, 13, 15] have studied, among other issues, the buffer space requirements for multiple multimedia streams. Most of these analyses are based on the buffer space needed by each stream individually. In other words, if $S_1, \ldots, S_n$ are the n streams running simultaneously in the system, the total amount of buffer space needed is $\sum_{i=1}^{n} B_i$, where B_i is the buffer space required by Stream S_i. Recent work [8, 13] has shown that by taking advantage of the temporal behaviour of the concurrent streams, buffers can be shared among these streams (cf. more details given in Section 2). The analysis and simulation in [8] indicate that buffer sharing can reduce the total buffer requirements to as little as $1/2 * \sum_{i=1}^{n} B_i$, achieving a 50% savings. Thus, sharing provides more efficient use of buffers and often improves the response times to queries, namely by increasing the number of concurrent streams that can be supported by a fixed amount of buffer space.

Given the benefits of buffer sharing, in this paper we study how to implement this idea (as [8] and [13] only give analyses and simulation results). A crucial issue which has not been examined elsewhere is buffer addressing. That is to say, after knowing how much buffer space to allocate to a collection of concurrent streams, how can the system decide exactly which buffers (i.e., blocks of memory) to use to contain the data read for each stream? Similarly, how can the playback process of each stream know where to find its data? As will be demonstrated in Section 3.1, the answers to these questions are complicated by the temporal aspects of the concurrent streams in that the buffer locations for each stream change over time.

To some extent, the problem of buffer addressing we are trying to solve here is similar to the problem of allocating blocks of memory to processes in a multi-programming operating system environment. However, a key difference between the two problems is a difference in time granularity. In operating system memory allocation, most changes in (virtual) memory allocation are made in the time granularity of process lifetimes, which are typically in seconds or minutes. In contrast, the time granularity for changes in buffer locations is on the order of milliseconds, assuming a display rate of 30 frames per second. Thus, we are looking for buffer addressing schemes that are with as little overhead as possible. To this end, the contributions of this paper are as follows.

- We will report the development of the CES Buffer Sharing Scheme that is very efficient to implement (i.e., constant-time computation). The scheme permits savings in buffer space asymptotically very close to the "ideal" savings predicted by the analysis given in [8].

- We will show how this scheme can be further improved, particularly for situations when the total number of concurrent streams is small, say less than 10.

- We will also demonstrate how the scheme operates under varying degrees of disk utilizations.

- As noted in [8, 9], when the number of concurrent streams changes (normally due to a completion/termination of a stream, or the admission of a new stream), the system enters into a transition period. We will show how to make the CES Buffer Sharing Scheme work during such periods of transition.

The organization of the paper is as follows. Section 2 presents several preliminary concepts and formulas, and gives an analysis on the benefit of buffer sharing. Section 3 introduces the CES Buffer Sharing Scheme and analyzes its behaviour with full disk utilization. Section 4 shows how the CES scheme can be further optimized by a more careful reuse of buffers. Section 5 analyzes the behaviour of the CES scheme when the disk utilization is less than full. Finally, Section 6 shows how to make the CES scheme work during periods of transitions.

2 Background: An Analysis of Buffer Sharing

2.1 Periodic Retrieval of Multiple Streams with No Buffer Sharing

Following the framework established in [4], we associate with each stream S_i a consumption rate p_i, which is the rate the data obtained from disks are consumed. For an uncompressed stream, its consumption rate is the same as its playback rate. To support n streams, $S_1, \ldots, S_n$, it is necessary that the data transfer rate R from disk is greater than the total consumption rate $\sum_{i=1}^{n} p_i$ of all the streams, so that the continuity requirements of all the streams will not be violated. More specifically, the disk [1] multiplexes itself by reading data for each stream in a periodic or cyclic fashion. Within a period, the disk spends t_i time units to read for S_i. Thus, if $s_{i,j}$ denotes the seek/switching time from Streams S_i to S_j, we have:

$$t_1 + \ldots + t_n + s_{1,2} + \ldots + s_{n,1} \quad \leq \quad T \tag{1}$$

where T denotes the total length of the period. To simplify notations, let $s = s_{1,2} + \ldots + s_{n,1}$. Then the disk utilization for the concurrent streams is given by:

$$\delta \quad = \quad \frac{t_1 + \ldots + t_n + s}{T} \tag{2}$$

[1] For ease of presentation here, we only consider the situation when there is only one disk. As far as buffer sharing is concerned, extending from one disk to multiple disks is straightforward.

Intuitively, it measures the degree to which the disk has been dedicated to the concurrent streams. The disk utilization is 1 when the disk is fully dedicated. The disk utilization is less than 1 when there is some time within each cycle that the disk is not serving the concurrent streams. We refer to this as the idle time within a cycle. Any value can be selected as the length of the reading period, keeping in mind that buffer utilization is directly proportional to that time value. If a stream needs 1.5 Mbps (as in MPEG-1 video), then 5 streams can be serviced in a period of length 10 seconds, if 15 Mbits are read for each stream. The total amount to be read is 75 Mbits or approximately 9 MBytes of buffer space would be required, assuming all of the data needs buffer space at some point.

In order for each stream to satisfy its continuity requirements, it is necessary to read sufficient data of S_i in time t_i so as to cover the (continuous) consumption of S_i for time T, that is,

$$t_i * R \geq T * p_i \tag{3}$$

In order to reduce the number of buffers used for each stream, however, we have:

$$t_i * R = T * p_i \tag{4}$$

From Equation (4), it is easy to see that $\frac{t_i}{t_j} = \frac{p_i}{p_j}$. In other words, to minimize buffer consumption, the reading time for each stream should be proportional to its consumption rate. Let P denote the total consumption rate, i.e., $P = p_1 + \ldots + p_n$. Then by combining Equations (2) and (4), t_i can be determined by:

$$t_i = (T * \delta - s) * \frac{p_i}{P} \tag{5}$$

By combining Equation (3) with the above equation, we can establish a lower bound on T:

$$T \geq \frac{s * R}{R * \delta - P} \tag{6}$$

Since the data transfer rate R is greater than the consumption rate p_i of each individual stream, buffers are needed for each stream. In particular, the maximum number of buffers is needed right after S_i has just finished reading. Thus, the number of buffers required by S_i is: $B_i = t_i * R - t_i * p_i$. By substituting Equation (5) into the above, we get:

$$B_i = p_i * (R - p_i) * \frac{T * \delta - s}{P} \tag{7}$$

Thus, the total buffer requirements for the n streams is:

$$B = \sum_{i=1}^{n} B_i = \frac{T * \delta - s}{P} * \sum_{i=1}^{n} p_i * (R - p_i) \tag{8}$$

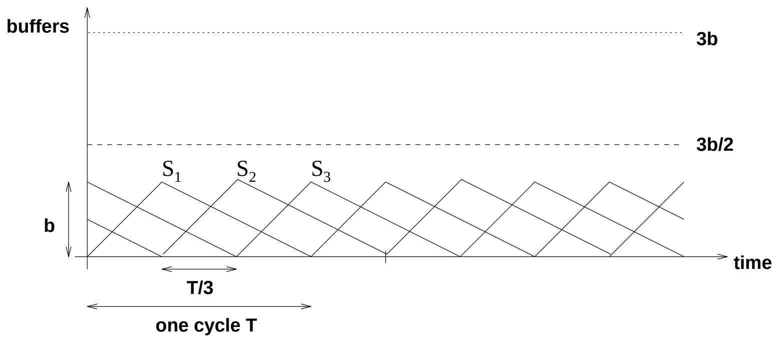

Figure 1: Buffer Sharing for 3 Streams with Identical Consumption Rates

If B_{max} is the maximum number of buffers available in the system, it is necessary that $B \leq B_{max}$. By substituting Equation (8) into $B \leq B_{max}$, we get an upper bound of the cycle length T:

$$T \leq \frac{B_{max} * P}{\delta * \sum_{i=1}^{n} p_i * (R - p_i)} + \frac{s}{\delta} \quad (9)$$

This equation can be combined with Equation (6) to decide whether it is possible to accept a new stream S_{n+1}. In particular, the admission controller computes the two equations by including the characteristics of S_{n+1} (i.e., sum to $n+1$). If the range defined by the two equations is empty (i.e., the right-hand-side of Equation (6) is strictly greater than that of Equation (9)), admitting S_{n+1} is not feasible. Otherwise, S_{n+1} can be admitted, and any value within the range can be picked as the value of T.

The analysis above assumes that apart from the seek required for switching from S_{i-1} to S_i, no extra seek is needed throughout time t_i when S_i is being read. This can be achieved by using the technique of storing data in clusters as proposed in [3], or by storing data contiguously (e.g. such as in a spiral optical disk). [8] discusses how to handle other situations of data placement.

2.2 The Benefit of Buffer Sharing

Many studies, such as [1, 3, 4, 9, 13, 15], have analyzed, among other issues, the buffer space requirements for multiple concurrent multimedia streams. However, nearly all the analyses are based on the buffer space needed by each stream individually. In other words, the total buffer space needed by the n streams is $\sum_{i=1}^{n} B_i$. Figure 1 shows clearly, however, that S_i does not need all B_i buffers at all times. In fact, S_i's buffer requirement can be less than B_i, for example when Stream S_{i+1} requires its maximum number of buffers. This fact is independently observed in [13]. Thus, a simple way to minimize total buffer consumption and thus to maximize buffer utilization is to allow the n streams to share buffers.

Figure 1 shows a simple situation when there are 3 streams S_1, S_2, S_3 in the cycle, each of which has the same consumption rate, but their data consumption cycles are offset in time with respect to each other, as in [13]. Thus, by Equation (5), each stream has an equal amount of reading time, i.e., same t_i. Since the cycle length T is normally much larger than the total switching time s, Figure 1 shows the simplified situation when $t_i = T/3$. Let us consider the total buffer requirement at time $4T/3$, at which point S_1 has just finished reading and requires b buffers, the maximum number of buffers that it ever needs. S_2, which is about to start reading, has run out of data. Thus, the buffer requirement of S_2 is 0. As for S_3, there were b buffers at time T, but at time $5T/3$, all the data in those buffers will be consumed. Thus, at the current time $4T/3$, S_3 needs $b/2$ buffers. Hence, the total number of buffers required by all 3 streams is $b + 0 + b/2 = 3b/2$. Note that if all the streams have identical consumption rates, their total buffer requirement does not change with time. Thus, $3b/2$ buffers are all the 3 streams need. However, without buffer sharing, $3b$ buffers are required. Thus, buffer sharing gives a 50% reduction in total buffer consumption.

An analysis of buffer sharing for the general case when there are n streams with heterogeneous consumption rates $p_1, \ldots, p_n$ involves finding the time point within a period when the total buffer requirement reaches the maximum. This is necessary because this maximum is no longer constant when $p_1, \ldots, p_n$ are not all the same. See [8] for more details. But here we will only consider the case when there are n streams with identical consumption rates. Since the consumption rates are the same, then by Equation (7), the individual buffer requirement B_i is the same, which we define to be b. This is equal to $pT(\dfrac{n-1}{n})$ when the disk is fully utilized and servicing n streams. The amount that must be displayed is pT and pT/n is what has been displayed while reading. Similarly, the reading time t_i for each stream is the same, say equal to t_0. Now let us consider the time when S_n has just finished reading. The following table shows the buffer requirement of each stream at that point.

Streams	S_1	S_2	S_3	$\ldots$	S_n
Buffers needed	0	$\frac{1}{n-1}b$	$\frac{2}{n-1}b$	$\ldots$	$\frac{n-1}{n-1}b$

First, S_n has just finished reading, thus requiring all b buffers. S_1 is about to start reading. Thus, it has 0 buffers of data at this point. S_2, at an earlier point in time, had b buffers of data which are supposed to cover the consumption of S_2 for a period of $(n-1) * t_0$. At the point when S_n has just finished reading, $(n-2) * t_0$ has elapsed from the time S_2 started consuming its data, or alternatively, S_2 will run out of data t_0 seconds later. Thus, the current level of buffered data for S_2 is $\dfrac{t_0}{(n-1) * t_0} b = \dfrac{1}{n-1} b$. Similarly, it is not difficult to see that the current level of buffered data for S_3 is $\dfrac{2}{n-1} b$. Hence, the total number of buffers needed is:

$$B_{shar} = \sum_{i=1}^{n} \frac{i-1}{n-1} b = \frac{n}{2} b \tag{10}$$

In this case, without buffer sharing, the total number of buffers required is $B = nb$. Thus, buffer sharing reduces total buffer consumption by 50%.

Example 1 Consider a homogeneous set of streams whose consumption rate is 240KB per second. (This is based on 24 frames per second where each frame is JPEG compressed to 10KB [11].) Given a disk whose maximum reading rate is 1000KB per second, 4 streams can be supported simultaneously, provided that there are enough buffers. If the length of the period for the 4 streams is 2.5 seconds, the buffer requirement for each stream is $b = 456$KB. Thus, without buffer sharing, about 2MB of buffer space is needed. But with buffer sharing, only 1MB is needed. Alternatively, if the system only has 1MB of buffer space, the number of streams that can be supported simultaneously without buffer sharing is only 2. With buffer sharing, the system can double the throughput and support all 4 streams. □

The above analysis assumes that the disk utilization δ is equal to 1. To take variations of disk utilization into account, we generalize the above table that shows the buffer requirement of each stream at the point after S_n has finished reading to become:

Streams	S_1	S_2	S_3	...	S_n
Buffers needed	cb	$(c + \frac{\delta}{n-\delta})b$	$(c + \frac{2\delta}{n-\delta})b$	...	$(c + \frac{(n-1)\delta}{n-\delta})b$

where $c = \frac{1-\delta}{1-\delta/n}$. For more information regarding these calculations, see [14]. A simple summation yields:

$$B_{shar} = \frac{2n - n\delta - \delta}{2(n-\delta)} * nb \tag{11}$$

Thus, in general, the percentage savings in buffers is given by:

$$\%\text{savings} = 100 * (1 - \frac{2n - n\delta - \delta}{2(n-\delta)}) = 100 * \frac{(n-1)\delta}{2(n-\delta)} \tag{12}$$

The above quantity is the largest when the disk utilization $\delta = 1$, in which case the percentage savings is 50%.

All the analyses presented so far are based on a fixed reading order of streams within a cycle. The benefit of allowing the reading order to change from one period to another is explored in [2, 3]. The gain is a reduction in total seek time, whereas the price to pay may be a doubling of buffer requirements. In ongoing work, we are studying whether we can get the best of both worlds by integrating buffer sharing with variable reading orders.

3 CES: A Buffer Addressing Scheme Based on Slots

Since buffer sharing can lead to considerable savings, the aim of this paper is to devise effective schemes to implement the idea. The main issue involved is buffer addressing. In other words, after deciding *how much* buffer space is needed by the concurrent streams, the system must decide *where* to put the data of each stream at *what time*. In this section, we will first present a naive implementation scheme which requires very costly bookkeeping. We will then present a buffer addressing scheme based on the notion of *slots*. We will show that the addressing scheme is easy to implement/maintain, and that the scheme permits savings in buffer space asymptotically very close to the "ideal" savings predicted by Equation (12).

3.1 Observations from a Naive Implementation

The following example demonstrates that with the minimum amount of buffer space (via buffer sharing), it is not possible to allocate contiguous blocks of memory space to the streams.

Example 2 Suppose we have 4 streams: A, B, C and D, each having the same consumption rate p. Consider the case when the disk utilization is 1. Then if the period is of length T, each stream will read for time $\frac{T}{4}$ within each period. Thus, it is easy to see that the buffer space needed for each stream is given by $b = p(T - \frac{T}{4}) = pT\frac{(4-1)}{4} = \frac{3}{4}pT$. Thus, by Equation (10), the total amount of space needed is $2b = 6p\frac{T}{4}$. For ease of illustration, let us divide this amount of memory space into 6 equal portions, each of size $p\frac{T}{4}$. Hereafter, we call these portions "slots." The following shows how these 6 slots will be used over time.

- At time $t = 0$, no portions are in use. Immediately after this moment, the disk starts to read for Stream A, and begins to use the space in Slots 1, 2, 3 and 4.

- At time $t = \frac{T}{4}$, Stream A occupies b buffers. In fact, more than b buffers worth of data of A has been read, because while reading takes place, some of the buffers being filled have already been consumed (i.e., Slot 1). Slot 1 is empty because the amount of data displayed during the reading period for A ($\frac{T}{4}$) is exactly equal to the size of slot 1. Thus, the buffer space can be diagrammed as follows:

1	2	3	4	5	6
free	A	A	A	free	free

  Immediately after the moment $t = T/4$, the disk starts to read for Stream B, and begins by using Slot 5.

- At time $t = 2T/4$, Stream A has just finished consuming its data in Slot 2. Meanwhile, Stream B now requires b buffers (i.e., 3 slots). Thus, the buffer space looks like:

1	2	3	4	5	6
B	B	A	A	free	B

- At time $t = 2T/4 + T/16$, Stream C has just finished filling its first (of 4) slots, namely Slot 5. At this moment, the buffer space looks like:

1	2	3	4	5	6
B	B	A*	A	C*	B*

The asterisks under Slots 3, 5 and 6 indicate that only parts (in fact, one quarter) of these slots are empty. Then the question is where to put C's data to be read next. The unfortunate answer is that C's data will have to be interleaved within the space in Slots 3, 5 and 6.

- At time $t = 3T/4$, Stream C has finished reading, and the buffer space diagram becomes:

1	2	3	4	5	6
B	B	C+	A	C+	C+

The + signs under Slots 3, 5 and 6 denote that the data in these slots are interleaved and not in contiguous order.

- After time $t = 3T/4$, reading for Stream D begins. Unfortunately, the fragmentation of D becomes even worse than that of C above. The fragmentation escalates even further when the next cycle begins. Eventually, data of all streams are totally interleaved.

□

The above example illustrates that with the minimum amount of buffer space (as predicted by Equation (10)), interleaving of data streams in buffers occurs rampantly. In fact, such interleaving is unavoidable. This is because at the steady state when the buffers are all full, having the minimum amount of buffer space implies that every buffer released after consumption by every stream must be filled *immediately* by the stream that is reading data at that time. More importantly, those locations will not always be contiguous. Even if a special attempt has been made to make sure that those locations are contiguous in one cycle, the previous example shows that the locations will no longer be contiguous in the next cycle. In sum, a key observation is that with the minimum amount of buffer space, interleaving is unavoidable.

The next question to ask then is how to record the interleaving, so that the system knows where to find the i^{th} piece of a data stream in the buffers. One straightforward way is by using a linked list or some indexing structure, one for each stream. When a data block is read from disk, the contents would be copied to the buffer and the start address copied to the indexing structure. Similarly, when a data block is consumed, the indexing structure would be accessed to find the start address, and then the data would be transferred to the display hardware. However, the disadvantages of this naive approach are:

- As illustrated by the above example, the degree that the data of a stream is interleaved or fragmented is very high. This implies that there would be many, many small contiguous pieces whose start addresses need to be recorded. Thus, the amount of storage space required by the indexing structure may not be small. Moreover, the indexing structure may use up memory space which would otherwise be available as buffer space for streams.

- Each read operation from and write operation to buffers require accessing or updating the indexing structure. As a data stream is highly fragmented, the frequency of accesses to the indexing structure would be very high. This would undoubtedly lead to a major performance loss.

So far, we have observed that with the minimum amount of buffer space, a high degree of interleaving is unavoidable. And we have just seen that dealing with such a high degree of interleaving is very costly. Furthermore, if we reflect on the situation without buffer sharing, this would correspond to the other extreme with a maximum amount of buffer space but a zero degree of interleaving (as each stream has its own buffer space). Thus, the amount of buffer space and the degree of interleaving are two opposing factors. The goal then is to find a suitable point in between the two extremes described above.

3.2 Buffer Allocation by Slots

Let us take a closer look at the situation described in Example 2. Rampant interleaving begins immediately after time $2T/4 + T/16$, which is the time when every buffer released (e.g., in Slots 3, 5 and 6) must be reused immediately to contain data of the reading stream (e.g., Stream C). Thus, to avoid rampant interleaving, our approach is to design a scheme that satisfies the following condition.

> *The reading stream must read its data to slots that are completely empty when the stream starts its reading period.*

Hereafter, we will refer to the above condition as Condition CES ("completely empty slots"). Obviously, for any scheme to obey the given condition, it must have enough buffer space at the beginning of each reading period to contain all the data of the reading stream to be read in that period. Then it is not difficult to see that the total buffer space needed is more than the "ideal" amount given by Equation (10). In the remainder of this section, we present a scheme called the CES Buffer Sharing Scheme that permits buffer savings asymptotically very close to the ideal, and that satisfies the above condition.

We begin the description of the CES scheme by re-visiting the concept of slots which we have used rather informally in Example 2. Recall that there is some amount of data of the reading stream that will be consumed before the reading stream finishes its reading in the cycle. For example, if we have 4 concurrent streams, each cycle is broken up into 4 reading periods, one for each stream. Thus, the amount of data that are consumed in one reading period is $p * \frac{T}{4}$, where p is the consumption rate of the streams, and T is the cycle length. This assumes that the disk utilization is 1 and that the total switching time among streams is small compared with the reading times. Section 5 will consider the situation when the disk utilization is less than 1. In general, for n concurrent streams, an amount of $\frac{pT}{n}$ of each stream is consumed in every reading period. Thus, we can divide the data to be consumed

	Reading Periods														
Slot	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	A1	B1	C1	A1	B1	C1	A1	B1	C1	A1	B1	C1	A1	B1	C1
1	A2		C2		B2		A2		C2		B2		A2		C2
2		B2		A2		C2		B2		A2		C2		B2	
3	A3	A3		A3	A3		A3	A3		A3	A3		A3	A3	
4		B3	B3		B3	B3		B3	B3		B3	B3		B3	B3
5			C3	C3		C3	C3		C3	C3		C3	C3		C3

Table 1: Slot Allocations for 3 Streams Under the CES Scheme

for a given stream in a cycle into *portions* of size $\frac{pT}{n}$. That is to say, the first portion of a stream is consumed during the reading period of that stream, while the second and further portions are consumed during the reading periods of other concurrent streams. A *slot* is a block of buffer space that can contain exactly one portion. Thus, the slot size is also $\frac{pT}{n}$.

Having introduced portions and slots, we now try to find the minimum number of slots necessary to satisfy Condition CES. On one hand, during each reading period, n completely empty slots are needed – one for consumption during the reading period, and $n-1$ for the other reading periods. On the other hand, at the end of each reading period, n slots are released – one for each concurrent stream, including the reading stream. Thus, if n slots are added to the minimum amount of buffer space, then the stream currently reading data can have n completely free slots, thereby satisfying Condition CES; and when the current reading period ends, the next reading stream will also have n completely free slots, because n slots have just been released.

Table 1 illustrates the situation when there are 3 concurrent streams. The columns of the table give the reading periods. For example, the first reading period is a period when Stream A is reading. Since there are 3 streams, 3 periods correspond to one cycle. Each row of the table specifies the portions that occupy the corresponding slot at the end of the reading periods. For instance, at the end of the first reading period, the second and third portions of A (i.e., A2 and A3) are in Slots 1 and 3 respectively. Notice that Slot 0 is actually empty at the end of each reading period, but the portion is included in Table 1 to indicate the stream that has just stopped reading in that period.

Let us consider the 7^{th} reading period in greater details, which is the steady-state version of the first reading period. Here Stream A has just finished reading, and Slots 1, 3 and 5 are occupied. For the next reading period, we need 3 slots to contain the data of Stream B. And there are exactly 3 empty slots available: Slot 0, 2 and 4. Thus, Condition CES is satisfied. At the end of the 8^{th} reading period, the second and third portions of B are in Slots 2 and 4 respectively. Meantime, the portions A2 and C3 have just been consumed. Thus, the empty slots are 1, 5 and 0, just enough for the next reading period. Here a key observation is that the 7^{th} reading period is identical to the 13^{th} period. The major implication is that by controlling the degree of interleaving, we can now know precisely where to get a certain portion of a stream at any point in time. In other words, no costly bookkeeping is necessary.

Table 2 in Section 3.4 later shows the slot allocations for 4 streams, and a table of this kind can be produced for any given n. In Section 3.4, we will explain in greater details the meaning of the specific cyclic patterns, and will present a formula computing the slot address of a portion. This is one of the two major questions remain to be answered. The other remaining question is how much space this scheme requires, which we will answer immediately below.

3.3 Space Requirement

As shown above, the proposed scheme satisfies Condition CES, and successfully prevents rampant interleaving from occurring. However, as argued in Section 3.1, we must pay a price in doing so, namely by using more space than the "ideal" case. To analyze how much more space we need, we make two observations from our previous description. First, at the end of each reading period, the total space of the *occupied* slots is equal to the minimum amount of buffer space needed. For example, for 3 streams, by Equation (10), the minimum amount of space needed is $3b/2$, where $b = 2p * T/3$, where p is the consumption rate and T is the cycle length. This amount is equal to the space of 3 slots, each of size $p * T/3$. As shown in Table 1, the number of occupied slots at the end of each reading period is always 3.

The second observation from the previous description is that the proposed scheme needs exactly n empty slots to work, which is the minimum number of slots required by Condition CES. Thus, the space requirement of the proposed scheme is the minimum buffer space obtained from Equation (10) plus n extra slots. Expressed in terms of b, the space requirement is given by:

$$\frac{n}{2}b + n * \frac{p * T}{n} \quad = \quad \frac{n}{2}b + \frac{n}{n-1}b \;=\; \frac{n+1}{n-1} * \frac{n}{2}b \tag{13}$$

as $pT = (\frac{n}{n-1})b$ from Section 2. Recall from Equation (10) that ideal buffer sharing requires $\frac{nb}{2}$ buffer space. Thus, the proposed scheme is only a factor $\frac{n+1}{n-1}$ off. And the larger the value of n, the closer is the space requirement of the proposed scheme to the ideal minimum.

Figure 2 shows the benefits of buffer sharing given by the proposed scheme. The x-axis is the number of concurrent streams, and the y-axis is the percentage of the amount of buffer space without buffer sharing (i.e., nb). The curve with asterisks corresponds to the proposed scheme based on Equation (13), and the 50% horizontal line corresponds to the ideal minimum based on Equation (10). The two curves become closer to each other as the

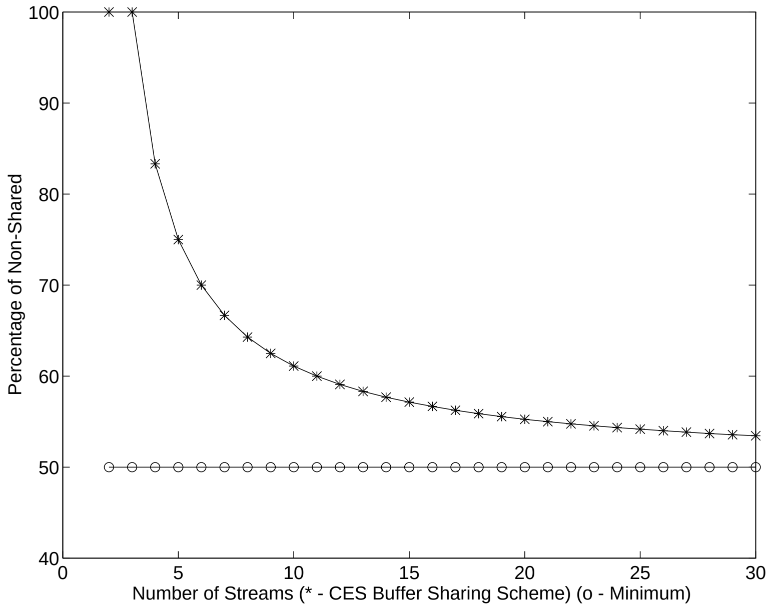

Figure 2: Percentage Savings in Buffer Space by the CES Scheme

number of streams grow. In other words, as n grows, the percentage savings in buffer space created by the proposed scheme is almost 50%, which corresponds to an enormous amount of buffer space in absolute terms.

3.4 Slot Addressing

In the rest of this section, we address the issue of how to compute the location of a portion of a stream at any given time. To do so, we must understand more about the patterns exhibited by the kind of slot allocation described in tables like Table 1. To better illustrate the patterns, we use Table 2 which describes the slot allocations for 4 streams. Note that the rows of the table are divided into 4 groups (marked by horizontal lines) based on the portions of the streams. More specifically, only the first portions of the streams (e.g., A1, B1) appear in the first group (i.e., Slot 0), and similarly only the second portions of the streams (e.g., A2, B2) appear in the second group (i.e., Slots 1 and 2). Observe that for a given stream, its lower-numbered portions occupy buffer space for a smaller amount of time than its higher-numbered portions. More specifically, its i$^{\text{th}}$ portion stays in the buffers one more reading period than its (i-1)$^{\text{st}}$ portion. Thus, slots containing higher-numbered portions have a smaller chance to be reused, and thus we need more slots for higher-numbered portions than for smaller-numbered ones. In the extreme case, the first portion of a stream is consumed at the end of the its reading period. Thus, we need only one slot for all the first portions: A1, B1, etc.

As for the second portions of the streams, observe that at the end of each reading period,

	Reading Periods															
Slot	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	A1	B1	C1	D1	A1	B1	C1	D1	A1	B1	C1	D1	A1	B1	C1	D1
1	A2		C2		A2		C2		A2		C2		A2		C2	
2		B2		D2		B2		D2		B2		D2		B2		D2
3	A3	A3		D3	D3		C3	C3		B3	B3		A3	A3		D3
4		B3	B3		A3	A3		D3	D3		C3	C3		B3	B3	
5			C3	C3		B3	B3		A3	A3		D3	D3		C3	C3
6	A4	A4	A4		A4	A4	A4		A4	A4	A4		A4	A4	A4	
7		B4	B4	B4		B4	B4	B4		B4	B4	B4		B4	B4	B4
8			C4	C4	C4		C4	C4	C4		C4	C4	C4		C4	C4
9				D4	D4	D4		D4	D4	D4		D4	D4	D4		D4

Table 2: Slot Allocations for 4 Streams Under the CES Scheme

there can only be one second portion remaining in the buffers – namely the second portion of the stream that has just finished reading (e.g., A2 at the end of the fifth reading period). Recall that to satisfy Condition CES, one more empty slot is needed to contain the second portion of the next reading stream (e.g., Stream B). This explains why all the second portions require only two slots (e.g., Slots 1 and 2). It also explains the cyclic pattern exhibited by all the second portions of the streams.

The situation for the third portions is very similar to that for the second portions. The key difference is that the third portions stay one reading period longer than their corresponding second portions. Consequently, in the steady state, there are always two third portions contained in the buffers (e.g., A3 and D3 at the end of the fifth reading period). Thus, 3 slots are needed to handle all the third portions. The portions use the three slots in an obvious, cyclic fashion.

In general, for n concurrent streams, k slots are needed for the k^{th} portion. A special case occurs for the last portion (i.e., $k = n$). Since n slots are allocated for the n last portions, each of those portions has its own slot. For instance, as shown in Table 2, A4, B4, C4 and D4 have exclusive uses of Slots 6, 7, 8 and 9 respectively. Incidentally, it is obvious from the above analysis that the total number of slots needed for n streams is given by $\sum_{k=1}^{n} k$. Thus, the total space needed is given by $\frac{n(n+1)}{2} * \frac{p*T}{n}$, where $\frac{p*T}{n}$ is the slot size. This amount, without surprise, is exactly the same as the figure computed by Equation (13).

We are now in a position to develop a formula to compute the slot address that contains a certain portion of a stream at any given point in time. In particular, the formula is of the form f(stream, portion, cycle number) = slot address/number. As an example, Table 3 gives the slot numbers that should be computed by the formula for 4 streams. The columns of

4 Streams	Portions											
Cycle No.	A2	B2	C2	D2	A3	B3	C3	D3	A4	B4	C4	D4
1	1	2	1	2	3	4	5	3	6	7	8	9
2	1	2	1	2	4	5	3	4	6	7	8	9
3	1	2	1	2	5	3	4	5	6	7	8	9
4	1	2	1	2	3	4	5	3	6	7	8	9

Table 3: Slot Addresses for 4 Streams Under the CES Scheme

the table correspond to the different portions of the 4 streams. Since the first portions of all streams are consumed within the same reading period they are read, they do not require any slot and are not included in the table. The rows of the table correspond to the cycles. Since each cycle consists of 4 reading periods (i.e., given 4 streams), the first cycle corresponds to the first 4 reading periods in Table 2, the second cycle the next 4 reading periods, and so on. Notice from Table 2 that A2 occupies only Slot 1 regardless of which cycle, this corresponds to the column of 1's for A2 in Table 3. Similarly, a previous discussion has explained the columns of 6's, 7's, 8's and 9's in Table 3 for A4, B4, C4 and D4 respectively.

A slightly more interesting situation occurs for the third portions of the streams. Look at Table 2 again. For A3, it occupies Slot 3 in the first cycle, Slot 4 in the second, and Slot 5 in the third. These correspond to the values given in Table 3 under the column for A3. While the situations for B3 and C3 are very similar, the situation for D3 is slightly more complicated. Observe from Table 2 that D3 stays in buffers across cycle boundaries (e.g., the 4^{th} and 5^{th} reading periods). Thus, strictly speaking, D3 has two different slot addresses in the second cycle: Slot 3 at the end of the 5^{th} period, and Slot 4 at the end of the 8^{th} period. However, to simplify the computation (i.e., to make slot address computation truly functional), we adopt the convention that a cycle number is only valid after the given stream has started its reading period in that cycle. For instance, as far as Stream D is concerned, the second cycle only begins at the 8^{th} reading period. In other words, the slot address at the end of the 5^{th} period corresponds to the first cycle. Thus, the column for D3 in Table 3 gives the (unique) value of 3 for the first cycle and 4 for the second cycle.

In the following, we give the procedure/formula that can be used to compute slot addresses. We assume that the concurrent streams are numbered $1, \ldots, n$, and we use i to denote a certain stream. We use the symbol K to denote the portion of a stream. Thus, the portion at time t of a stream is given by

$$K = \left\lceil \frac{t \mod T}{(\frac{T}{n})} \right\rceil \tag{14}$$

where T is the length of the cycle. We use CN to denote the cycle number which is given by

$$CN = 1 + (t \ div \ T) \tag{15}$$

Procedure ComputeSlotAddr
Input K, CN, i, and n.

1. If $K = 1$, output 0 and halt.
2. If $K = 2$:
   (a) If n is even, output the value of $(i + 1) \ mod \ 2 + 1$, and halt.
   (b) Otherwise, output the value of $(i + CN) \ mod \ 2 + 1$, and halt.
3. If $K = n$, output the value of $\frac{n(n-1)}{2} + ((i - 1) \ mod \ n)$ and halt.
4. Otherwise (i.e., $2 < K < n$), output the value of $\frac{K(K-1)}{2} + [(n - K)(CN - 1) + i - 1] \ mod \ K$. □

Example 3 Suppose we have 4 concurrent streams (i.e., $n = 4$). Suppose we want to find the slot address of the third portion of Stream D in the second cycle. From Table 3, Slot 4 should be the answer. Now for Procedure ComputeSlotAddr, the values of K, CN and i are 3, 2 and 4 respectively. In Step (4), the expression enumerated is $\frac{3(3-2)}{2} + [(4 - 3)(2 - 1) + 4 - 1] \ mod \ 3$, which is equal to 4. Similarly, if we are interested in the fourth portion of D, the answer computed is 9, the same as the value given in Table 3. □

Recall that all first portions of the streams are contained in Slot 0. Step (1) of Procedure ComputeSlotAddr gives exactly this value. As for the second portions, if there are an even number of concurrent streams (e.g., see Table 2), then all odd-numbered streams (e.g., Stream A) use Slot 1, and all even-numbered streams (e.g., Stream B) use Slot 2. If there are an odd number of streams, then the second portion of every stream uses Slot 1 and 2, depending on the cycle number. This explains Step (2) of Procedure ComputeSlotAddr. As for the last portions of the streams, recall that each stream has its own slot. This gives rise to the expression $((i - 1) \ mod \ n)$ in Step (3). The expression $\frac{n(n-1)}{2}$ corresponds to the total number of slots required to hold all but the last portions of all the streams (i.e., $\sum_{j=1}^{n-1} j$).

To understand the computation carried out in Step (4) for $2 < K < n$, it is important to note that during each successive cycle, the K$^{\text{th}}$ portion of a given stream must be placed in the next available slot within the group of slots allocated for all the K$^{\text{th}}$ portions. From one cycle to the next, there are $(n - K)$ slots used up by the streams that did not fit into the first allocation of those slots. Thus, in the the next cycle, stream i must be offset by an additional $(n - K)$ slots. If this causes a wrap-around, the calculation must be performed modulo K, to cycle back to the original beginning of that group of slots. As an example, consider $n = 4$ and $K = 3$. As shown in Table 3, each successive cycle moves the portion forward by 4 - 3 = 1 slot. The wrap-around is handled by the modulo 3 operation that puts each partition into the correct slot.

Note that the output obtained from Procedure ComputeSlotAddr only gives the slot number. But once the slot number is obtained, the exact address of a certain sub-portion of a stream can be easily computed. Also note that some of the expressions computed in Procedure ComputeSlotAddr may look a bit messy. But in fact, they can enumerated in constant time.

To summarize, in this section, we have first argued that using the minimum amount of buffer space in buffer sharing would introduce rampant interleaving which is too costly to keep track of. In response, we have proposed the CES Buffer Sharing scheme that is based on portions and slots. The scheme, in satisfying Condition CES, only allows interleaving to the level of slots. A key benefit then is that to locate any part of a stream at any given point in time, slot addresses can be computed in constant time, and no costly bookkeeping is required. Furthermore, we have shown that while the scheme needs extra buffer space, it gives buffer savings asymptotically equal to the ideal case described in Equations (10) and (12).

4 Further Improvement: a Hybrid Scheme

The curves in Figure 2 show that when the number of concurrent streams is small, the CES Buffer Sharing scheme does not give too much savings in buffer space. In fact, when there are only 3 streams, there is no savings at all (cf: 6 slots needed in Table 1). And for 4 streams, the savings is less than 20%, far from the 50% predicted by Equation (12). In this section, we show how this situation can be ameliorated by a more careful reuse of buffers.

4.1 The NCES Buffer Sharing Scheme

Let us consider the special case when there are only two streams. We can implement the following scheme that requires the minimum amount of buffers. The key is to interleave the reading locations and choose not to buffer the data which are both read and consumed in the same reading period. More specifically, Stream A can fill b buffers by reading its data in 2 parts – with every other read going directly to the process that consumes the data (e.g., display hardware), and the remaining reads being stored in the buffers. Data in the stored buffers are displayed after Stream A has finished its reading period. Now when Stream B reads data, it follows the same strategy, consuming one buffer and storing one buffer in the same slot as is freed by Stream A during that time. Note that for this scheme to work, the coordination between reading and consuming must be very precise, and no mismatches

	Reading Periods											
Slot	1	2	3	4	5	6	7	8	9	10	11	12
1	A2		C2		B2		A2		C2		B2	
2		B2		A2		C2		B2		A2		C2
3	A3	A3	C3	C3	B3	B3	A3	A3	C3	C3	B3	B3
4		B3	B3	A3	A3	C3	C3	B3	B3	A3	A3	C3

Table 4: Slot Allocations for 3 Streams Under the NCES Scheme

in timing can be tolerated. Also note that this scheme does not satisfy Condition CES in that not every slot used by the reading streams is completely empty at the beginning of the reading period. Hereafter, we refer to this scheme as the NCES Buffer Sharing scheme.

The NCES scheme can be extended to situations with larger number of concurrent streams. Tables 4 show the slot allocations for 3 streams. As compared with Table 1, Slot 0 and 5 are eliminated. Slot 0 is not needed because data are sent directly to the process that consumes the data. While Tables 1 and 4 do not differ in the first two reading periods (modulo Slot 0), the differences between the two tables start to show in the third reading period. In the situation described in Table 1, the scheme obeys Condition CES, and thus C3 is read into Slot 5 which is completely empty at the beginning of the third period. In the situation described here, Portion C1 is sent directly to the consuming process, and C2 fills up Slot 1. By the time Slot 1 is filled, a considerable part of A3 has already been consumed (i.e., 2/3 to be exact). In other words, Slot 3 is almost empty. Thus, C3 is read into Slot 3. Even though the rate at which C3 is read into Slot 3 is three times as fast as the rate A3 relinquishes buffers in Slot 3, it is not difficult to see that the data of C3 will never catch up with the data of A3 in Slot 3. In fact, right after the last frame of A3 has been consumed, the space just freed is immediately filled with the last frame of C3. Similarly, in the next reading period (i.e., the 4^{th} period), A2 fills up Slot 2, and A3 occupies Slot 4 just as B3 frees up space in that slot. Again, contrast this with the situation shown in Table 1.

For the NCES scheme in general, it is easy to see that we can set up the kind of slot addresses tables like Table 3, and modify Procedure ComputeSlotAddr slightly to compute the slot addresses of the portions of the n streams. The only change needed is the case for the n^{th} portions of the streams. This is because there are now only $(n-1)$ slots to be shared among all the n^{th} portions of the streams. We omit the details here.

4.2 A Hybrid Scheme

From the above analysis, it is quite obvious that the NCES scheme requires 2 fewer slots than the CES scheme proposed in the previous section. As shown in Section 3.3, the CES scheme needs n more slots than the minimum amount of buffer space. Thus, the NCES scheme requires $(n-2)$ extra slots when compared with the minimum amount. Slot 0 is

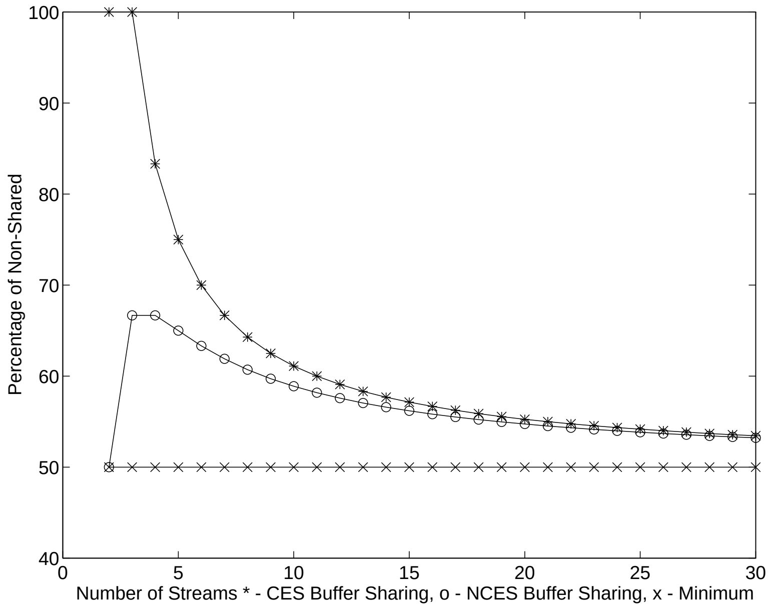

Figure 3: Percentage Savings in Buffer Space by the CES and NCES Schemes

not needed because data is displayed directly to the output device, while one slot in the last group is not needed, cince the final slot read into was not empty at the beginning of the cycle. Total buffer requirements become:

$$\frac{n}{2}b + (n-2) * \frac{p * T}{n} \quad = \quad \frac{n}{2}b + \frac{n-2}{n-1}b \; = \; (\frac{n-2}{n-1} + \frac{n}{2}) \; b \tag{16}$$

The curve with circles in Figure 3 shows the benefits of buffer sharing by the NCES scheme. When n is very small, the NCES scheme gives a better percentage savings in buffer space than the CES scheme. But when n increases, the difference between the two schemes becomes negligible.

Compared with the CES scheme, the NCES scheme has some shortcomings. First, as discussed above, the NCES scheme requires very precise coordination between data reading and consumption. It cannot handle any variation in effective disk access rate. In contrast, by using slightly more buffer space, the CES scheme is more resistant to unexpected changes in the hardware conditions. Moreover, recall that the NCES scheme requires reading data in two parts. This necessarily complicates the processing that is needed. Therefore, in light of these complications caused by the NCES scheme, we believe that it is not worthwhile to adopt the NCES scheme for all values of n. Instead, we propose to use the NCES scheme only when the number of concurrent streams is small, say n less than 10, but to use the CES scheme for larger values of n.

5 Variations of Disk Utilization

All the analyses and discussions presented in the last two sections assume that the disk utilization δ is equal to 1. That is, all disk activities are dedicated to the concurrent streams, and there is no idle period within a cycle. However, there are situations in which it would be necessary to operate a disk at a utilization level strictly below 1. In those situations, there would be an idle period within each cycle. As shown in [8], the idle period may enable the prefetching of data for streams that are in the waiting queue, and make it easier to increase the number of concurrent streams. In this section, we will examine how the presence of idle periods affect the ways that we can implement buffer sharing.

5.1 The Effect of Idle Periods

There are at least two different models of idle periods, depending on how the idle time is distributed within a cycle. Consider for example that there are 3 concurrent streams involved in a cycle. Suppose that the cycle length is 3 seconds, 0.3 seconds of which the disk is idle. One way to operate in this situation is to make the disk read for 0.9 seconds for the first stream, pause for 0.1 second, read for 0.9 seconds for the second stream, and so on. Another way is to make the disk read 0.9 seconds for each stream without pause, and leave all 0.3 seconds of idle time at the end of each cycle. We argue that the latter model is generally more useful than the former because by combining all small idle periods (e.g., 0.1 seconds each) into one large period (e.g., 0.3 seconds), operations such as prefetching can be supported more easily [8]. Thus, in this section, we will base our analysis on the latter model.

A natural question to ask at this point is what differences an idle period at the end of a cycle would make. A key difference is that when $\delta = 1$, the first stream starts to read (indicating the beginning of the next cycle) immediately after the last stream has just finished reading. However, with the presence of the idle period at the end of a cycle, the first stream must wait a bit longer before it can read again. In fact, this comment applies to all streams in that during the idle period, all streams are being consumed, while no stream is reading. The implication is that more buffer space must be allocated to each stream so that it can survive a longer gap between its successive reading periods.

5.2 Non-uniform Portion Sizes

Another complication caused by $\delta < 1$ is that the portion or slot size may need to be changed. Recall that when $\delta = 1$, a stream reads for $\frac{T}{n}$ seconds, where T is the length of the cycle and n is the number of concurrent streams. If $\delta < 1$, and T is still the cycle length, then to account for the idle period, the reading period of each stream must be less than $\frac{T}{n}$. Therefore, if portions are still chosen to be $\frac{T}{n}$ seconds worth of data, as discussed in earlier sections, Slot 0 will not be empty after Stream A (i.e., the first stream) has just finished reading. This will complicate the consumption of Portion A2 because now part of A2 is contained in Slot 0. This motivates the need to change the portion size to an appropriate level. Since there are n reading periods within a total time of δT, the portions must be changed to the size of $\delta \frac{T}{n}$ seconds worth of data, or $p * \delta \frac{T}{n}$ bytes, where p is the consumption rate of the streams.

The above portion size would work for all n reading periods. In particular, if the same number of slots are used as for the case when $\delta = 1$, and all slots are of the size $p * \delta \frac{T}{n}$, enough data would be stored to last until the last stream has just finished reading. But there would not be any buffer space for the data to be consumed during the idle time. This motivates the need to have non-uniform portion sizes. More specifically, the portions consumed at the end of a cycle must be enlarged. Consider for example the situation for 4 streams. As shown in Table 2, the portions that are consumed between the beginning of the last reading period and the beginning of the next cycle are A4, B3, C2 and D1. Each of these portions must be enlarged by an amount of $(1 - \delta)T$ seconds worth of data. Since these enlarged portions share slots with other "normal-sized" portions (e.g., D1 shares Slot 0 with A1, B1 and C1), it would not be economical to simply increase the sizes of the slots that contain these portions. The best way is to make all slots the same size as the normal-sized portions (i.e., $p * \frac{T\delta}{n}$ bytes), but have n extra slots of size $p * (1 - \delta)T$ to contain the extra data corresponding to the n enlarged portions. Thus, the total space required is given by:

$$\frac{n(n+1)\delta}{2} * \frac{b}{n-1} \; + \; n * p * T * (1 - \delta) \tag{17}$$

The first factor corresponds to the normal-sized portions, and the second factor corresponds to the n enlarged portions. By substituting $pT = (\frac{n}{n-1})b$ (see Section 2.2), the second factor can be rewritten in terms of b as $(\frac{n}{n-1})\; b * n * (1 - \delta)$. Figure 4 plots the buffer space requirements given by the above formula as percentages of nb (i.e., the total amount of buffer space needed without buffer sharing) when $\delta = 0.9$. The curve with circles represents the minimum predicted by Equation(12) for the same disk utilization level. Similarly, Figure 5 shows the comparison when $\delta = 0.75$. In both cases, the proposed adjustment to the CES Buffer Sharing scheme gives savings in buffer space asymptotically close to the ideal minimum.

The slot address of a portion can be computed in exactly the same way as shown in Section 3.4 with one modification. In calculating the value of K, the stream portion parameter, special attention must be made to find out if K corresponds to an enlarged portion. If this is the case, the slot that contains the extra amount of $p * (1 - \delta)T$ must also be accessed, together with the slot whose address is returned by Procedure ComputeSlotAddr. We omit the details here. Furthermore, it is not difficult to see that the current arrangement can be slightly enhanced, particularly when n is small, by the same approach described in Section 4. Again we omit the details here.

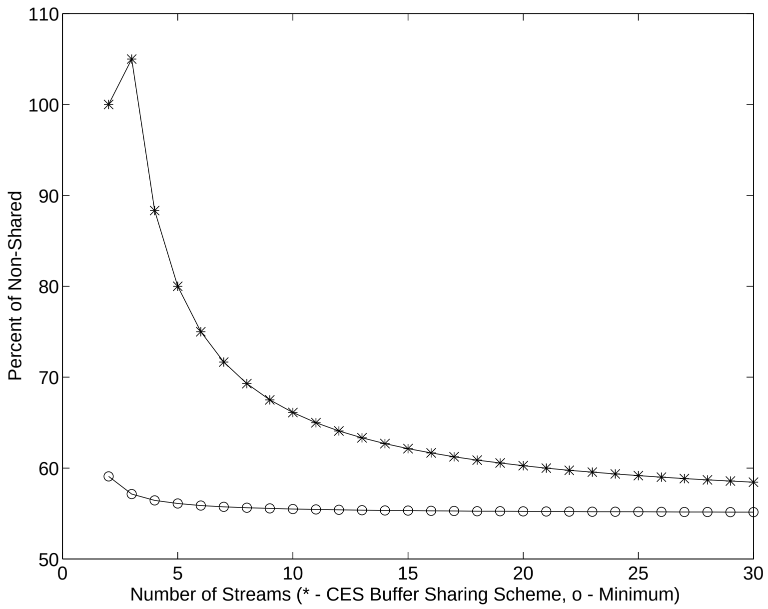

Figure 4: Percentage Savings in Buffer Space by the CES Scheme: 90% Utilization

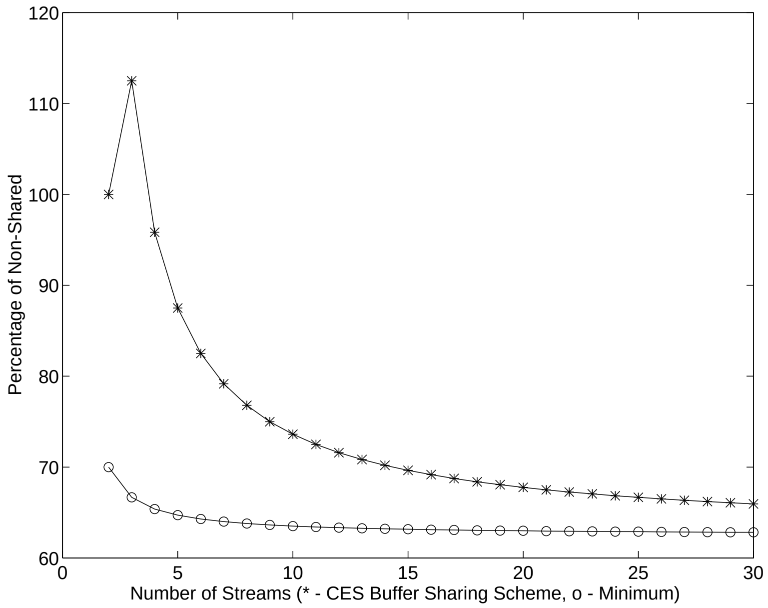

Figure 5: Percentage Savings in Buffer Space by the CES Scheme: 75% Utilization

In sum, in this section, we have studied how the presence of an idle period within a cycle, that is when $\delta < 1$, can affect the ways buffer sharing can be implemented. We have shown that by using slots of different sizes, the CES scheme introduced in Sections 3 can now handle situations when $\delta < 1$. Again the adjusted scheme permits savings in buffer space very close to the minimum.

6 The CES Scheme During Transition Periods

6.1 Transition Periods

Whenever the number of concurrent streams changes (typically due to a completion or termination of a stream, or the admission of a new stream), the system enters into a *transition* period. More specifically, let $S_1, \ldots, S_n$ be the concurrent streams, and their cycle be denoted by C_n. Now suppose that a new stream S_{n+1} has been admitted, and that the new cycle will be C_{n+1}. The transition period is the period in between the old cycle C_n and the new cycle C_{n+1}. The transition period may manifest itself in at least two ways.

- The cycles C_n and C_{n+1} may have different lengths. If T_n and T_{n+1} are the lengths of C_n and C_{n+1} respectively, then T_{n+1} is greater than T_n. In this case, the transition period involves extending the cycle length from T_n to T_{n+1}. As noted in [8, 9], a natural way is to increase the cycle length in small increments, until the cycle length reaches T_{n+1}. This process may take seconds to complete. In general, the time taken depends mainly on the exact values of T_n and T_{n+1} and the disk utilization δ.

- The cycles C_n and C_{n+1} may also be different in their buffer allocations and addressing. For example, if the number of concurrent streams increases from 3 to 4, then with the CES scheme, a transition needs to take place from the slot allocations described in Table 1 to the allocations shown in Table 2. As will be argued later in Section 6.2, this transition can be very complicated. Actually, this transition would be much simpler if we *assume* that the system always has enough space to accommodate two different sets of slots simultaneously. For example, if the 10 slots for 4 streams can be created in a buffer region different from the region that contains the 6 slots for 3 streams, the transition could be instantaneous. However, this assumption is obviously too strong. In the rest of this section, we consider the situation when this assumption may not be satisfied, and slot changes must be done in place.

Before we begin to analyze how the CES scheme can be adjusted to deal with transition periods, we point out that as noted earlier, a transition period may be caused by a decrease in the number of concurrent streams. Since typically making a transition from C_{n+1} to C_n is easier to deal with than the reverse transition, our analysis below focuses exclusively on the case when the number of concurrent streams increases.

6.2 Complications Caused by Changing Cycle Length

As observed above, a transition from the cycle C_n to C_{n+1} may require a longer cycle length (i.e., $T_{n+1} > T_n$). Thus, each stream must read more data in its reading period to guarantee continuous consumption during a longer cycle. From the point of view of buffer allocation by slots, there are two possible approaches to achieve this.

The first approach is to change the slot sizes. However, this entails changing the slot boundaries based on the current cycle length. Furthermore, recall from the above discussion that in most cases extending from T_n to T_{n+1} must be done in small increments. This implies that if changing slot size is our approach, a series of changes to the slot boundaries would be required. However, it is easy to see that changing slot boundaries can easily corrupt the portions that are contained in the original slots. And a series of slot boundary changes would be extremely messy and complicated to maintain, and would likely require a larger amount of buffer space.

The aforementioned complications caused by changing slots sizes motivate the second approach. While keeping the slot sizes the same as before, the second approach to allow streams to read more data is to increase the number of slots allocated to the streams. By keeping the slot sizes unchanged, this approach avoids the tough problem of changing slot boundaries encountered by the first approach. However, it has its own problems to deal with. The major one is that as the cycle length increases, the size of a portion needs to be changed, so much so that the portion size is no longer identical to the slot size. Thus, a portion may be contained in more than one slot, and a slot may contain more than one portion (unless a larger amount of buffer space can be tolerated). In either case, slot allocation and addressing become complicated, and the CES scheme no longer works.

6.3 Constant Cycle Length

The above analysis suggests that a change in cycle length would seriously complicate buffer allocation and addressing during a transition period. The obvious alternative is then to keep the cycle length constant. In this way, there is no need to change slot boundaries, portion and slot sizes when the transition from C_n to C_{n+1} is made. The crucial question, however, is whether it is always possible to keep the cycle length unchanged when the number of concurrent streams changes.

To answer this question, we recall from Section 2 that for a given collection of concurrent streams, $S_1, \ldots, S_n$, the cycle length T is bounded below by Equation (6) and above by Equation (9). If the range $[\frac{s * R}{R * \delta - P}, \frac{B_{max} * P}{\delta * \sum_{i=1}^{n} P_i * (R - P_i)} + \frac{s}{\delta}]$ defined by the two equations is empty, then it is not possible to support the collection of streams without violating the continuity requirements. Otherwise, any value within the range can be picked as the value of the cycle length.

Let us take a closer look at the above range, and denote the range for n streams by μ_n. Naturally, when n increases to $(n+1)$, the range becomes narrower. (Eventually n reaches the value when μ_n becomes empty.) However, it is important to observe that when all the streams have an identical consumption rate, which is the case considered in this paper, the upper bound of μ_n does not change. In particular, the expression $\frac{B_{max} * P}{\delta * \sum_{i=1}^{n} p_i * (R - p_i)} + \frac{s}{\delta}$ is equal to $\frac{B_{max} * n * p_i}{\delta * p_i * \sum_{i=1}^{n} (R - p_i)} + \frac{s}{\delta}$, because P can be replaced by $n * p_i$. As $\sum_{i=1}^{n} (R - p_i) = n * (R - p_i)$, the latter expression can be simplified to become $\frac{B_{max}}{\delta * (R - p_i)} + \frac{s}{\delta}$, which is independent of n. This implies that the only reason why μ_{n+1} is narrower than μ_n is that the lower bound $\frac{s * R}{R * \delta - P}$ grows as n increases. [2] We can therefore conclude that $\mu_{n+1} \subset \mu_n$.

From the point of view of dealing with the transition periods, the relationship $\mu_{n+1} \subset \mu_n$ is very desirable. This is because for all the different numbers of concurrent streams that can be supported by the system, we can pick a cycle length T that need not be changed as n changes. In particular, for any given system, we can find the maximum number n_{max} of streams that can be supported. n_{max} is equal to $n_{empty} - 1$, where n_{empty} is the least integer that would cause the lower bound $\frac{s * R}{R * \delta - P}$ to be strictly greater than the upper bound $\frac{B_{max}}{\delta * (R - P_i)} + \frac{s}{\delta}$. Then we can find the range $\mu_{n_{max}}$, and pick any value within the range to be the value of T. And we can be assured that T is contained in the range μ_n for all $1 \leq n \leq n_{max}$.

If the number of concurrent streams increases from n to $(n + 1)$, and yet the cycle length remains unchanged, a natural question to ask is which aspect of the system needs to be changed to accommodate an extra stream. The answer is the disk utilization δ. In particular, for n streams, let the disk utilization be δ_n, and the idle time be I_n. As the number of streams increases by one, the disk accommodates the extra stream S_{n+1} by shortening its idle time to I_{n+1}, and using the amount $(I_n - I_{n+1})$ of time to read for S_{n+1}. Thus, the disk utilization increases from δ_n to δ_{n+1}. It is not difficult to verify that as long as $(n+1) \leq n_{max}$, δ_{n+1} is always less than or equal to 1, indicating that the disk is in a feasible state.

We are now in a position to complete our description on how the CES scheme works during a transition period. When the number of streams increases from n to $(n+1)$, $(n+1)$ extra slots are allocated. As described in Section 5, n of these $(n+1)$ slots are of the normal size, and one of these is enlarged with the amount to be consumed in the idle period. The first time when S_{n+1} reads from disk, all its portions are stored in these newly allocated slots. But as other streams begin to read, these slots will be released to contain the right portions from the other reading streams.

Table 5 gives an example of how the slot allocations change during a transition period from 3 to 4 streams. The first reading period in the table is the 3$^{\text{rd}}$ reading period shown in Table 1. As discussed before, the first time Stream D reads data is in the idle period of

[2] This in turn is caused by the increases in s and P as n grows.

Slot	Reading Periods 3	idle	4	5	6	7	8
0	C1	C1	A1	B1	C1	D1	A1
1	C2	C2		B2		D2	
2			A2		C2		A2
3			A3	A3		D3	D3
4	B3	B3		B3	B3		A3
5	C3	C3	C3		C3	C3	
6		(D1)	A4	A4	A4		A4
7		D2		B4	B4	B4	
8		D3	D3		C4	C4	C4
9		D4	D4	D4		D4	D4

Table 5: Transition Period: Slot Allocations from 3 to 4 Streams

the original cycle. Thus, to better illustrate the situation during this idle period, we add the column labelled "idle" in Table 5. The first six slots in this column are exactly the same as those in the previous column. This is to indicate that as far as Streams A, B and C are concerned, nothing needs to be changed. But the major event taking place during this idle period is the reading of the portions of D. D1, D2, D3 and D4 are read into the newly allocated slots: Slot 6 to 9. The entry "(D1)" in this column denotes that D1 was read into Slot 6 at the beginning of the idle period, but has been consumed by the end of the idle period.

Next is the reading period for Stream A which reads its portions into Slot 0, 2, 3 and 6, which are completely empty at the beginning of this reading period. Meantime, the portions B3, C2 and D2 have been consumed. This gives rise to the next column, labelled "4". The reading of the portions and the filling up of the slots continue in exactly the same way as described in Section 3.2. The end of the second reading period of D, denoted by the column labelled "7", marks the end of the transition period. Thus, it is no surprise to find that the last column in Table 5, labelled "8", is basically equivalent to the column in Table 2 that represents the 5^{th} reading period.

Note that during the transition period, special care must be taken to accommodate the changes in the enlarged portions. More specifically, when there are only 3 streams, the enlarged portions are A3, B2 and C1. But with 4 streams, the enlarged portions become A4, B3, C2 and D1. Thus, in filling up the slots, the system must act appropriately for these new enlarged portions. We omit the details here.

In sum, in this section, we have analyzed how the presence of transition periods can affect the implementation of buffer sharing. We have argued that if the cycle length is allowed to change as the number of concurrent streams changes, the CES scheme would break down.

Fortunately, we have shown that it is possible to keep the cycle length constant, while allowing the number of concurrent streams to vary. In this case, the CES scheme works just fine.

7 Conclusions

In this paper, we have studied how to implement buffer sharing in continuous-media systems. We have proposed the CES Buffer Sharing Scheme which is based on slots and portions. The scheme only allows interleaving of streams to the level of slots, and as such, avoids rampant interleaving which is extremely costly to maintain. In particular, under the CES scheme, the slot address of any portion of a stream at any given instant can be computed in constant time, requiring no bookkeeping at all. Furthermore, regardless of the level of disk utilization, the scheme gives buffer savings asymptotically very close to the ideal minimum, which may be as high as 50%. Last but not least, we have also demonstrated that CES can work effectively during transition periods.

We have also proposed and studied the NCES Buffer Sharing Scheme which tries to further optimize the CES scheme by a more careful reuse of slots. When n, the number of concurrent streams, is small, the NCES scheme gives quite significantly higher savings than the CES scheme. However, we only recommend using the NCES scheme when n is small, partly because this difference in savings gradually becomes negligible as n grows, and partly because the NCES scheme is harder to implement than CES.

In ongoing work, we will integrate the schemes into a distributed continuous-media file system which is under development at the University of British Columbia [6]. We will also study how to extend the CES and NCES schemes to handle concurrent streams with different consumption rates: $p_1, \ldots, p_n$. One of the key issues is the impact of variations in consumption rates on slot and portion sizes. While the most general situation may be very difficult for CES to deal with, we believe that if for all $1 \leq i \leq n$, p_i has to be a multiple of some base rate, then it is quite likely that by some form of splitting and merging slots, CES scheme may work just as well.

References

[1] D. Anderson, Y. Osawa and R. Govindan. (1992) *A File System for Continuous Media*, ACM Trans. on Computer Systems, 10, 4.

[2] M. Chen, D. Kandlur and P. Yu. (1993) *Optimization of the Grouped Sweeping Scheduling with Heterogeneous Multimedia Streams*, Proc. ACM-Multimedia, pp 235–242.

[3] J. Gemmell. (1993) *Multimedia Network File Servers: Multi-channel Delay Sensitive Data Retrieval*, Proc. ACM-Multimedia, pp 243–250.

[4] J. Gemmell and S. Christodoulakis. (1992) *Principles of Delay-Sensitive Multimedia Data Storage and Retrieval*, ACM Trans. on Information Systems, 10, 1, pp 51–90.

[5] G. Miller, G. Baber and M. Gilliland. (1993) *News On-Demand for Multimedia Networks*, Proc. ACM-Multimedia, pp 383–392.

[6] G. Neufeld, N. Hutchinson, R. Ng and M. Ito. (1993) *A Distributed Continuous-Media File System*, CITR grant.

[7] R. Ng, C. Faloutsos and T. Sellis. (1991) *Flexible Buffer Allocation Based on Marginal Gains*, Proc. ACM-SIGMOD, pp 387–396.

[8] R. Ng and J. Yang. (1994) *Maximizing Buffer and Disk Utilizations for News On-Demand*, Proc. VLDB 94.

[9] P. Venkat Rangan and H. Vin. (1991) *Designing File Systems for Digital Video and Audio*, Proc. ACM Symposium on Operating Systems Principles, pp 69–79.

[10] A. Reddy and J. Wyllie. (1993) *Disk Scheduling in a Multimedia I/O System*, Proc. ACM-Multimedia, pp 225–233.

[11] L. Rowe and B. Smith. (1992) *A Continuous Media Player*, Proc. 3rd Intl. Workshop on Network and OS Support for Digital Audio and Video.

[12] K. Tindell and A. Burns. (1993) *Scheduling Hard Real-Time Multimedia Disk Traffic*, Technical Report, University of York, England.

[13] F. A. Tobagi, J. Pang, R. Baird, and M. Gang. (1993) *Streaming Raid - A Disk Array Management System for Video Files*, Proc. ACM-Multimedia, pp 393–400.

[14] J. Yang. (1994) *Maximizing Buffer and Disk Utilizations for News On-Demand*, MSc. Thesis, UBC.

[15] C. Yu, W. Sun, D. Bitton, Q. Yang and R. Bruno. (1989) *Efficient Placement of Audio Data on Optical Disks for Real-Time Applications*, Communications of ACM, 32, 7, pp 862–871.