An Analysis of Buffer Sharing and Prefetching Techniques for Multimedia Systems*

Raymond T. Ng† and Jinhai Yang
Department of Computer Science
University of British Columbia
Vancouver, B.C., V6T 1Z4
Canada.

Abstract

In this paper, we study the problem of how to maximize the throughput of a continuous-media system, given a fixed amount of buffer space and disk bandwidth both pre-determined at design-time. Our approach is to maximize the utilizations of disk and buffers. We propose doing so in two ways. First, we analyze a scheme that allows multiple streams to share buffers. Our analysis and preliminary simulation results indicate that buffer sharing could lead to as much as 50% reduction in total buffer requirement. Second, we develop three prefetching strategies: SP, IP1 and IP2. As will be demonstrated by SP, straightforward prefetching is not effective at all. In contrast, IP1 and IP2, which prefetch more intelligently than does SP, could be valuable in maximizing the effective use of buffers and disk. Our preliminary simulation results show that IP1 and IP2 could lead to a 40% improvement in throughput.

keywords: buffer allocation, prefetching strategies, analysis of continuous-media or multimedia systems

1 Introduction

With the advances in networking, storage, and I/O interface technologies, providing effective multimedia support in database management systems has become a topic of great interest and value. To support audio and video data, multimedia database management systems need to deal with several tough issues. First, audio and video data are delay-sensitive. As recording and playback of video and audio data are continuous operations, a management

*Research partially sponsored by NSERC Grants OGP0138055 and STR0134419, and the CITR Grant on "Distributed Continuous-Media File Systems."

†Person handling correspondence. Email: rng@cs.ubc.ca.

system, once starts displaying audio or video data, must guarantee that enough resources are 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. Third, a multimedia object may consist of multiple components: audio, video and text. It is the responsibility of the management system to ensure that these multiple streams can be synchronized during retrieval.

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, 10, 11, 13, 14]. With respect to the topic area of this paper, these studies can be grouped into two major categories. The first group is primarily concerned with intelligent disk scheduling. Studies in this group include the sweeping scheme proposed by Chen, Kandlur and Yu [2], the sorting-set algorithm developed by Gemmel [3], the SCAN-EDF strategy designed by Reddy and Wyllie [11], and the hard real-time approach analyzed by Tindell and Burns [13]. The second group deals with constrained block allocation, which limits the distance between successive blocks of a multimedia stream. Studies in this group include the scattering parameter approach developed by Rangan and Vin [10], the cluster strategy introduced by Gemmel and Christodoulakis [3, 4], and the audio data placement work of Yu et. al. [14]. To a large extent, most of these proposals aim to minimize seek latencies so as to satisfy the continuity requirements of multimedia streams. And most of them are developed from a design perspective.

For a large class of applications and systems, however, a good understanding of the dynamic behaviour of the system is at least as important as the design. For instance, consider news on-demand systems (e.g. [6]). Such a system typically has highly non-uniform, and asynchronous arrivals of queries/requests. It is the intent of this paper to investigate how to provide better dynamic support for such systems. More specifically, given a fixed amount of buffer space and a fixed disk bandwidth both pre-determined at design time, we study how to maximize the throughput of a multimedia system, and minimize the response time of queries, with the guarantee that all continuity requirements will be satisfied. For a system with fixed disk bandwidth and buffer space, the response time of queries are primarily governed by the utilization of disk and buffers. Thus, our approach is to utilize buffers and the disk as effectively as possible. In particular, in this paper, we will report:

- a scheme that allows multiple streams to share buffers. We will analyze the importance of buffer sharing, at varying disk utilization levels and consumption/playback rates of the streams. Our analyses show that buffer sharing can lead to a 50% reduction in total buffer requirement. Given the fact that audio and video data requires a huge amount of buffer space, a 50% reduction indeed represents very substantial savings. Our preliminary simulation results also provide further evidence showing the effectiveness of buffer sharing.

- three prefetching strategies: SP, IP1 and IP2. As will be demonstrated by Strategy SP, straightforward prefetching may not be effective at all. In contrast, Strategies IP1 and IP2, which prefetch more intelligently than does SP, could be very valuable in

within each cycle	t_1	t_2	...	t_n	remaining time
disk activities	read S_1	read S_2	...	read S_n	idle

Figure 1: Disk Activities within a Cycle

maximizing disk and buffer utilizations, as well as system throughput. Our preliminary simulation results indicate that IP1 and IP2 can lead to a 40% improvement in throughput.

The organization of the paper is as follows. Section 2 presents a preliminary analysis on periodic retrieval of multiple streams, and gives several basic equations needed in later analyses. Section 3 analyzes the importance of buffer sharing. Section 4 introduces and analyzes the prefetching strategies. Section 5 presents preliminary simulation results. Section 6 discusses how to support the proposed techniques in a multiple disk environment, and how to support data streams that are non-contiguously placed on disks.

2 Preliminary Analysis: Periodic Retrieval of Multiple Streams

As observed in [4, 10], the most natural way to process multiple streams simultaneously is to interleave the reading of the streams in a cyclic fashion. In this paper, we assume that within all the cycles/periods, streams are read in a fixed order, cf. Figure 1. [2, 3] explore the benefit of allowing the reading order to change from one period to another, so as to minimize total seek time. We will discuss this strategy of variable reading order in greater details later.

2.1 Defining Disk Utilization

Let there be n multiple streams denoted by $S_1, \ldots, S_n$. Let the consumption rate [1] of Stream S_i be P_i, and the amount of time reading S_i in each period be t_i. Then if $s_{i,j}$ denotes the seek (or switching) time from S_i to S_j, we have: $t_1 + \ldots + t_n + s_{1,2} + \ldots + s_{n,1} \leq t$, where t denotes the total length of the cycle. To simplify notations, let $s = s_{1,2} + \ldots + s_{n,1}$. Then the disk utilization, ρ, is given by:

$$\rho = \frac{t_1 + \ldots + t_n + s}{t} \tag{1}$$

Figure 2 summarizes the meanings of the symbols to be used in this paper.

[1] The consumption rate refers to the rate the data obtained from disk are consumed. For an uncompressed stream, its consumption rate is the same as its playback rate.

Symbol	Meaning of Symbol
B_{max}	maximum number of available buffers
B	total buffer consumption of n streams
B_{shar}	total buffer consumption of n streams with buffer sharing
B_i	buffer consumption of Stream S_i
Bl	block size in non-contiguous placement
G	seek time between non-adjacent blocks in non-contiguous placement
P	total consumption rate of n streams
P_i	consumption rate of Stream S_i
P_i^{pft}	consumption rate of Stream S_i after prefetching
R	maximum disk reading rate
S_i	the i-th stream
s	total switching time within a cycle
$s_{i,j}$	switching time between Streams S_i and S_j
t	length of a cycle
t_i	reading time for S_i within a cycle
T_i	length of Stream S_i (in seconds)
ρ	disk utilization

Figure 2: Meanings of Symbols Used

2.2 Determining t_i

Now let us take a closer look at each Stream S_i. The analysis below 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 proposed in [3], or by storing data contiguously (e.g. such as in a spiral optical disk). In Section 6.2, we will relax this assumption to handle other situations of data placement.

Within each period, the total amount of data consumed by S_i is $t * P_i$, and the amount read for S_i is $t_i * R$, where R is the maximum disk reading rate. Thus, the continuity requirement of S_i can be expressed as:

$$t_i * R \quad \geq \quad t * P_i \tag{2}$$

However, in order to reduce the number of buffers used for each stream, we have:

$$t_i * R \quad = \quad t * P_i \tag{3}$$

From Equation 3, 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 1 and 3, t_i can be determined by:

$$t_i \quad = \quad (t * \rho - s) * \frac{P_i}{P} \tag{4}$$

2.3 Determining a Lower Bound of t

The above equation gives the amount of reading time for S_i in terms of t, the length of the cycle. In the following, we establish a lower bound on t, by combining Equations 2 and 4:

$$t \geq \frac{s * R}{R * \rho - P} \tag{5}$$

This equation leads to two interesting observations. First, the equation is valid only if $(R * \rho - P) > 0$. Even if the disk utilization ρ is set to the maximum 1, it is necessary that $R > P$. This is the most obvious admission control criterion. That is, without violating their continuity requirements, a system cannot admit so many streams that their total consumption rate P exceeds the disk bandwidth. In Section 4, we will show how this constraint can be relaxed by prefetching.

Second, t is inversely proportional to ρ. In other words, the longer the length of the period, the less utilized the disk becomes (for the n streams). This is because as t increases, the proportion of time wasted in switching (i.e. $\frac{s}{t}$) within every cycle becomes smaller. In other words, a longer period corresponds to a higher percentage of useful work (i.e. data transfer) done by the disk, and the disk becomes more effective. Hence, the proportion of the time when the disk is idle becomes higher. In Section 4, we will show how to make use of this relationship between t and ρ to maximize prefetching.

2.4 Buffer Requirements of Multiple Streams

Recall from the above analysis that the basic strategy to support multiple streams simultaneously is that for each Stream S_i, enough data of S_i must be read in time t_i to cover the consumption of S_i for time t. To achieve this, buffers are needed for S_i. 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: $B_i = t_i * R - t_i * P_i$. Combining with Equation 4, we get:

$$B_i = P_i * (R - P_i) * \frac{t * \rho - s}{P} \tag{6}$$

Thus, the total buffer requirement of the n streams is:

$$B = \sum_{i=1}^{n} B_i = \frac{t * \rho - s}{P} * \sum_{i=1}^{n} P_i * (R - P_i) \tag{7}$$

Two observations can be drawn from the above equation. First, it is obvious from the equation that the longer the period length t, the higher the value of B is. Second, if B_{max} is the maximum number of buffers available in the system, it is necessary that $B \leq B_{max}$. By substituting Equation 7 into $B \leq B_{max}$, we get an upper bound of the cycle length t:

$$t \leq \frac{B_{max} * P}{\rho * \sum_{i=1}^{n} P_i * (R - P_i)} + \frac{s}{\rho} \tag{8}$$

This equation can be combined with Equation 5 to provide the following admission control policy.

Admission Control Let $S_1, \ldots, S_{n-1}$ be all the streams in the current cycle, and S_n be the stream to be decided whether admission is possible.

1. Compute the lower bound (of t) based on Equation 5 and the upper bound based on Equation 8.

2. If the lower bound is strictly greater than the upper bound, then it is not possible to add S_n without violating continuity requirements.

3. Otherwise, S_n can be admitted to form a new cycle, and any value between the lower and upper bound can be chosen as the length of the new cycle. $\square$

In Section 4, we will return to this issue of picking a value for t, and analyze in greater details how to do that to maximize prefetching.

3 Buffer Sharing and its Benefits

Thus far, we have analyzed the handling of multiple streams from the viewpoints of disk bandwidth and buffer allocation. As defined in Equation 7, the total buffer requirement of n streams is based on the assumption that each stream S_i occupies B_i buffers within each cycle. However, as will be shown in Figure 3, S_i does not need all B_i buffers at all times. In fact, almost always S_i's buffer requirement is less than B_i. Thus, a simple way to minimize total buffer consumption and thus to maximize buffer utilization is to allow the n streams to share buffers. In this section, we will analyze the benefits of buffer sharing at varying disk utilization levels and consumption rates of the streams.

3.1 Streams with Identical Consumption Rates

3.1.1 A Simple Example

Figure 3 shows a simple situation when there are 3 streams S_1, S_2, S_3 in the cycle, all of which has the same consumption rate. Thus, by Equation 4, 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 3 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, $3b/2$

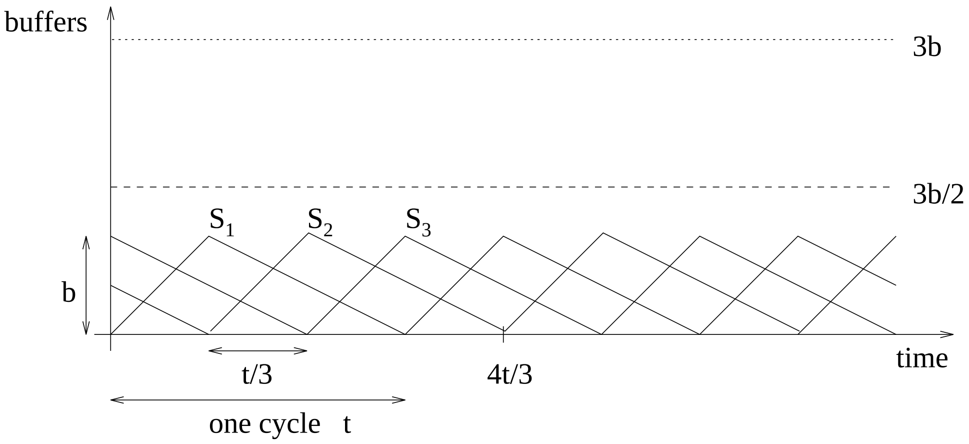


Figure 3: Buffer Sharing for 3 Streams with Identical Consumption Rates

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.

In the following, we will first analyze more formally the situation when all the streams have the same consumption rate. In particular, we will first study the case when the disk is fully utilized. We will then generalize our analysis to the case when the disk utilization ρ is less than 1. In Section 3.2, we will analyze the situation when the streams may have different consumption rates.

3.1.2 Disk Utilization $\rho = 1$

Since the consumption rates are the same, the individual buffer requirement B_i is the same, which is equal to b say. 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, or alternatively, S_2 will run out of data t_0 seconds later. Thus, the current level of buffered data for S_2 is $\frac{t_0}{(n-1)*t_0}b = \frac{1}{n-1}b$. Similarly, it is not difficult to see that the current level of buffered data for S_3 is $\frac{2}{n-1}b$. Hence, the total number of buffers needed is:

$$B_{shar} = \sum_{i=1}^{n} \frac{i-1}{n-1}b = \frac{n}{2}b \tag{9}$$

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 [12].) Given a disk whose maximum reading rate is 1000KB per second, 4 streams can be supported simultaneously, provided that there are enough buffers. Let the total switching time be $s = 0.1$ secs. Furthermore, let us pick the minimum cycle length, which corresponds to $\rho = 1$. Then by Equation 5, $t = 2.5$ secs. By Equation 6, the maximum 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. □

3.1.3 Disk Utilization $\rho < 1$

The above analysis assumes that the disk utilization ρ is equal to 1. In the following, we study the situation when $\rho < 1$. In other words, there is an idle period within each cycle (cf: Figure 1). This complicates the calculation of buffer requirements at a particular point in a cycle in that within the idle period, no stream is reading, and all streams are consuming data. Thus, more buffers are needed for each stream. This is reflected in the following table which generalizes the one shown in Section 3.1.2. The table below shows the buffer requirement of each stream at the point after S_n has finished reading:

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

where $c = \frac{1-\rho}{1-\rho/n}$, which reflects the length of the idle period relative to the length of the cycle. It is easy to see that when $\rho = 1$, the above table reduces to the one shown in Section 3.1.2.

A simple summation of the entries in the table above yields:

$$B_{shar} = \sum_{i=1}^{n}(cb + \frac{(i-1)\rho}{n-\rho}b) = \frac{2n - n\rho - \rho}{2(n-\rho)} * nb \tag{10}$$

Notice that when $\rho = 1$, $\frac{2n-n\rho-\rho}{2(n-\rho)} * nb$ reduces back to $nb/2$, which is the result obtained before. Furthermore, it is not difficult to see that for a fixed n, B_{shar} decreases as ρ increases. In other words, the percentage savings in buffer space brought about by buffer sharing increases as ρ increases, and achieves its maximum of 50% when $\rho = 1$.

Finally, according to Equation 6, b is equal to $(R - \frac{P}{n}) * \frac{t*\rho-s}{n}$. Thus, the full equation of B_{shar} is:

$$B_{shar} = (R - \frac{P}{n}) * (t * \rho - s) * \frac{2n - n\rho - \rho}{2(n-\rho)} \tag{11}$$

This equation can replace Equation 7 (and thus Equation 8) in the admission control test shown in Section 2.4.

3.2 Streams with Different Consumption Rates

So far, we have analyzed buffer sharing among streams that have the same consumption rate. In the rest of this section, we analyze the situation for n streams with heterogeneous consumption rates $P_1, \ldots, P_n$. The major complication here is that with heterogeneous consumption rates, the time point within a period at which the total buffer requirement reaches the maximum may not necessarily be right after Stream S_n has just finished reading. In fact, this point of maximum total buffer requirement may occur right after any S_i has finished reading. To analyze the situation more thoroughly, we introduce the following notations. Let BS_i stand for the buffer requirement of Stream S_i when buffers are shared. Let BA_i stand for the total buffer requirement of all the streams right after S_i has just finished reading.

At the beginning of the cycle when S_1 is about to start its reading, the buffer requirements of all the streams are:

$$
\begin{aligned}
BS_1 &= 0; \\
BS_2 &= \frac{t_1}{t - t_2} B_2; \\
BS_3 &= \frac{t_1 + t_2}{t - t_3} B_3; \\
&\cdots \\
BS_n &= \frac{t_1 + t_2 + \ldots + t_{n-1}}{t - t_n} B_n
\end{aligned}
\tag{12}
$$

The above formulas are easy to get. As S_1 is about to start reading, it has no data left, requiring no buffer at this moment. The buffered data of S_2 will last the time $t - t_2$, and its reading is still t_1 time away. Thus, right now the amount of buffered data is $t_1/(t - t_2)$. The buffer requirements of all the other streams at this moment can be calculated in a similar way. Adding up all the above formulas yields:

$$
BA_0 = \sum_{i=1}^{n} BS_i = \sum_{i=2}^{n} \frac{\sum_{j=1}^{i-1} t_j}{t - t_i} B_i \tag{13}
$$

After getting BA_0, it is not difficult to get BA_1, BA_2, etc. This can be done by setting up the difference equations that computes the difference in total buffer requirement between the time point S_i has just finished reading and the time point S_{i+1} has finished reading. For instance, at the time point when S_1 has finished reading, the buffer required by S_1 will be increased from 0 to B_1. However, for all the other streams, since t_1 time has elapsed, their buffer requirements decrease by $B_j * t_1/(t - t_j)$. Thus, it is obvious that

$$
BA_1 = BA_0 + B_1 - \sum_{j=2}^{n} \frac{t_1}{t - t_j} B_j \tag{14}
$$

More generally, we have:

$$BA_{i+1} \;\; = \;\; BA_i + B_{i+1} - \sum_{j=1, j\neq i}^{n} \frac{t_{i+1}}{t - t_j} B_j \qquad (15)$$

for all $1 \leq i \leq n$. Thus, by using Equations 13 and 15, all the values of $BA_1, \ldots, BA_n$ can be obtained. The maximum of these values gives the total buffer requirement when buffers are shared among streams with heterogeneous consumption rates:

$$B_{shar} \;\; = \;\; Max_{1 \leq i \leq n} \{BA_i\} \qquad (16)$$

As an interesting observation, if the streams are served in ascending order of consumption rates, the maximum value will occur at the point when S_n has just finished reading. In this case, BA_n will be B_{shar}, and there is no need to find the maximum according to the above equation.

We can use the general formula above to verify the results presented in previous sections. If all the consumption rates P_i's are the same, it is easy to see BA_{i+1} is always greater than or equal to BA_i. That is to say, the maximum buffer requirement always occurs at the end of the reading of S_n (cf: Sections 3.1.2 and 3.1.3). In particular, for the same consumption rate, Equation 13 becomes $BA_0 = \frac{n\rho(n-1)}{2(n-\rho)}b$, where $B_i = b$ for all i. Similarly, Equation 15 becomes $BA_{i+1} = BA_i + \frac{n-n\rho}{n-\rho}b$. Thus, we have:

$$B_{shar} \; = \; BA_n \;\; = \;\; BA_0 + n * \frac{n - n\rho}{n - \rho} b = \frac{2n - n\rho - \rho}{2(n-\rho)} * nb \qquad (17)$$

This is exactly the same as Equation 10.

3.3 Discussions

Thus far, we have analyzed the benefit of sharing buffers. However, from an implementation point of view, buffer sharing is not easy to support. This is because even given B_{shar} number of total buffers, the buffer manager needs to keep track of the exact buffer locations for each Stream S_i. This bookkeeping is complicated by the fact that the buffer locations for S_i keep changing from one cycle to another. However, since buffer sharing can lead to a 50% reduction in total buffer consumption, in ongoing work we are developing implementation schemes to support buffer sharing.

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

In sum, in this section, we have analyzed the benefits of buffer sharing. First we have considered the situation when all the streams are of identical consumption rate, at varying levels of disk utilization. Then we have extended our analysis to the situation when streams may have different consumption rates. Our analyses show that buffer sharing can reduce total buffer requirement by as much as 50%. In Section 5, we will provide simulation results further demonstrating the effectiveness of buffer sharing.

4 Prefetching Strategies

We have argued that for a system with fixed disk bandwidth and buffer space, the key to maximizing system throughput is to maximize the utilization of the disk and the buffer space. In the previous section, we have studied how buffer sharing can reduce total buffer requirement and improve the utilization of buffers. In this section, we will analyze how prefetching can improve the utilization of the disk. After discussing the general benefits of prefetching, we will present a straightforward prefetching algorithm called SP. But we will argue that SP may not be effective because of its simplicity. We will then introduce two more intelligent prefetching strategies: IP1 and IP2. In Section 5, we will present simulation results comparing the effectiveness of these three prefetching strategies.

4.1 General Benefits of Prefetching

On receiving a new request for a stream (referred to as a new query from now on), the admission controller that we have discussed so far simply checks if there are enough disk bandwidth and buffers to satisfy the new query, using Equations 5 and 8. If there are enough resources, the query is activated. Otherwise, the query sits idle in the waiting queue. Consequently, there are resources – buffers and disk bandwidth – that are not utilized at all [2]. For instance, consider the situation mentioned in Example 1. If the disk bandwidth can support only 4 streams and there are 2MB buffering space, buffer sharing would render 1MB idle. In general, we measure the performance of our system by its throughput and the response time of queries. But given a system with pre-determined (at design time) disk bandwidth and amount of buffer space, the response time of queries are primarily determined by the utilization of disks and buffers. Thus, our goal here is to try to use these resources as much as possible. More specifically, in this section, we explore how data prefetching can maximize resources utilization, and thus lead to an increase in system throughput.

There are at least 3 ways that prefetching can help a query.

- First, if a query has a consumption rate P_i that is larger than R, then even after a query is activated (i.e. becoming one of the queries served in a cycle), the query cannot be consumed immediately without violating the continuity requirements. Thus, to reduce the time between activation and the beginning of consumption, a system can prefetch portion of this query while it is still waiting in the waiting queue.

[2] In this paper, we only consider FIFO as the queueing discipline. It has the advantage of being simple and fair. Adopting other queueing discipline may require additional work to ensure fairness.

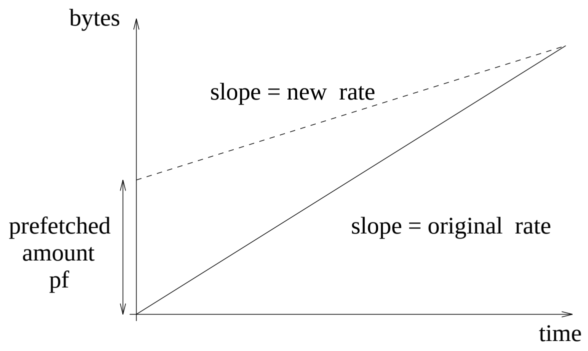


Figure 4: Reducing Consumption Rate by Prefetching

- Second, even if a query S_{n+1} has a consumption rate P_i less than R, prefetching portion of this query before activation may reduce the response time of the query. To see that, let say that $S_1, \ldots, S_n$ are the activated queries. At some point, query S_1 has finished, and S_{n+1} is activated. For reasons apparent later in Section 5.1, the reading order may become $S_2, \ldots, S_{n+1}$. If no data has been prefetched for S_{n+1}, then S_{n+1} cannot be consumed until S_{n+1} starts reading, which is at the end of the cycle. However, if there is sufficient amount of prefetched data of S_{n+1}, consumption of S_{n+1} can start immediately at the beginning of the cycle. Thus, there is a difference in response time which may be as large as one cycle length.

- Third, prefetching portion of this query before activation has the effect of reducing the effective consumption rate of the query after activation. This is illustrated by the simple diagram in Figure 4. The solid line represents the original consumption curve, whose slope is given by the consumption rate P_i. If an amount pf is prefetched, then the new, prefetched consumption rate is given by the slope of the dotted line. A simple analysis reveals that if T_i is the length of the query, the new, prefetched consumption rate is given by:

$$P_i^{pft} = P_i - \frac{pf}{T_i} \qquad (18)$$

  Since the new rate is less than the original rate, there is a possibility that the new rate may pass the admission control test, while the old one may not. Whenever this happens, the response time of the query is substantially reduced (cf: Example 3 later).

4.2 A Simple Prefetching Strategy: SP

Just like normal data retrieval from disk, prefetching requires both disk bandwidth and buffers. One obvious way to allow prefetching to happen is to dedicate a certain level of disk bandwidth and buffers to prefetching. But this would backfire as it reduces the disk bandwidth and buffers available to activated queries. Thus, we make sure that prefetching is not done at the expense of activated queries. To this end, recall that the cycle length t for the activated streams/queries $S_1, \ldots, S_n$ are bounded below and above respectively by Equations 5 and 8. If the system does not support prefetching at all, any value between the upper and lower bounds can be picked as the value of t. However, to support prefetching, an immediate question to answer is how to pick t so as to maximize prefetching, but not at the expense of the activated queries.

In fact, setting t to any value between the upper and lower bounds does not have any influence whatsoever on the completion times of the activated queries, as the completion time of a query is determined by its consumption rate and length [3]. Thus, as long as a value is picked between the lower and upper bounds, the activated queries will not be affected. Let us consider setting t to its lower bound. Then as discussed in Section 2.3, this corresponds to a disk utilization ρ of 1. In other words, all the disk bandwidth is used up for the activated queries, and nothing is left for prefetching. On the other hand, consider setting t to its upper bound. From the point of view of disk bandwidth allocation, this time there is ample room for prefetching because as discussed in Section 2.3, a longer cycle length corresponds to a lower disk utilization ρ. However, as discussed in Section 2.4, the trouble is that all the buffers are used up for the allocated queries. Thus, at the end, no prefetching can be done. Hence, the question to address is which value of t in between the upper and lower bounds maximizes prefetching.

There is actually another factor that affects the amount of prefetching that can be done. All the above analysis is based on the assumption that the cycle for the current collection of activated queries keep on going. Let T_{finish} denote the time the next activated query will have finished. The range bounding t is only valid before T_{finish}, after which the current cycle has to be changed anyway, and new calculations are required. Thus, the consideration of T_{finish} suggests a simple strategy (referred to as SP) to pick t so as to maximize prefetching. It equates the amount of data that can be retrieved in time T_{finish} with the amount of buffers that are available. This is formalized below. First, it is obvious that the amount of data that can be prefetched in time T_{finish} is: $D_{pf} = T_{finish} * R * (1 - \rho)$. By substituting Equations 3 and 4, we get:

$$D_{pf} \quad = \quad T_{finish} * R * (1 - \frac{s}{t} - \frac{P}{R}) \tag{19}$$

On the other hand, according to Equation 7, the buffers available for prefetching is given by:

$$B_{pf} \; = \; B_{max} - B \quad = \quad B_{max} - \frac{t}{R} \sum_{i=1}^{n} P_i * (R - P_i) \tag{20}$$

To maximize prefetching, SP sets $D_{pf} = B_{pf}$, which is equivalent to:

$$T_{finish} * R * (1 - \frac{s}{t} - \frac{P}{R}) \quad = \quad B_{max} - \frac{t}{R} \sum_{i=1}^{n} P_i * (R - P_i) \tag{21}$$

[3] This is assuming normal termination, not preempted by such events as user quitting prematurely or system failures.

within each cycle	t'_1	t'_2	...	t'_n	remaining time
disk activities	read S_1	read S_2	...	read S_n	prefetch S_{n+1}

Figure 5: Cyclic Activities with Prefetching

This is a quadratic equation in t in the form of $at^2 + bt + c = 0$. Solving this quadratic equation in the standard way gives a positive solution v_0 (and a negative solution). If v_0 falls within the lower and upper bounds of t, which occurs more often than not, v_0 is the value of t. Otherwise, if v_0 is strictly less than the lower bound, t is set to the lower bound. And if v_0 is strictly greater than the upper bound, the upper bound becomes the value of t.

The equations presented above do not assume buffer sharing, and are based on Equation 7. Since prefetching is orthogonal to buffer sharing, a similar set of equations can be derived for the buffer sharing case based on Equation 11. Strategy SP is summarized in the following.

Strategy SP Let $S_1, \ldots, S_n$ be all the activated queries, as allowed by the admission controller. Let S_{n+1} be the query at the head of the waiting queue.

1. Use Equation 21 to determine the length t of the cycle for $S_1, \ldots, S_n$.
2. Use the remaining disk bandwidth and buffers to prefetch for S_{n+1} at the end of each cycle.
3. Prefetching stops when an activated query has finished, or the system has run out of buffers. □

Figure 5 shows the disk activities within each cycle with prefetching. It differs from Figure 1 in two respects. First, its cycle length may be different from that without prefetching. Thus, the reading times for the activated queries are $t'_1, \ldots, t'_n$, instead of $t_1, \ldots, t_n$. More importantly, after the activated queries have finished reading, the disk may no longer be idle, and may be engaged in prefetching S_{n+1}.

4.3 Motivation for a More Intelligent Prefetching Strategy

Prefetching Strategy SP maximizes prefetching for the query S_{n+1} at the head of the waiting queue. Doing so, it may minimize the response time of S_{n+1}. However, as a result, S_{n+1} may use up too much system resources, particularly free buffers – for its own good, but not necessarily for the overall benefit of the system. More specifically, SP just lets S_{n+1} prefetch as much as possible, but does not consider whether S_{n+1} really needs that much data to get started once an activated query has finished. As shown in the example below, too much prefetched data only occupy buffer space, without doing any good to system performance.

Example 2 Consider a situation similar to the one described in Example 1. There are 4 activated queries, each with a consumption rate 240KB/s. And there is 1MB of buffer space left. Now consider a scenario where the query S_5 is the only query in the waiting queue with the same consumption rate. As discussed before, SP would allow S_5 to prefetch as much as possible, using up all 1MB of buffer space. Eventually, when one of the activated queries has finished, S_5 would be activated. Let us consider the potential benefits of prefetching 1MB of S_5. Among the three general benefits of prefetching discussed in Section 4.1, the only benefit applicable in this case is to reduce the response time of S_5 by at most a cycle length. However, as calculated in Example 1, 456KB is all that is needed for S_5 within a cycle. In other words, 456KB is sufficient to minimize the response time of S_5. Thus, the question is whether prefetching an extra 544KB can lead to any gain. The answer is no, because once the consumption of S_5 begins, its completion time depends entirely on its length and its consumption rate. Giving extra buffers does not help in any way. And in fact, it can be harmful to the entire system as there is now 544KB less of buffer space available. □

The above example suggests that while maximizing prefetching, the SP's approach of prefetching just for the query at the head of the waiting queue may not be sufficient. Thus, for a more effective prefetching strategy, the questions to be answered are: a) how to maximize prefetching, and b) how to determine how much to prefetch for a query in the waiting queue. The following example shows how looking ahead beyond the query S_{n+1} at the head of the waiting queue can help to determine the amount to prefetch for S_{n+1}.

Example 3 Consider the situation discussed in the previous example again. There are 4 activated queries with consumption rate 240KB/s each. Suppose there are now two queries in the waiting queue: S_5 and S_6 both with consumption rate 240KB/s. Further assume that the disk has a maximum reading rate of $R = 1150$KB/s, and there is now 1.5MB of buffer space. Now let us consider the time when one of the activated queries has finished, and consider two different amounts of prefetched data of S_5.

First, assume that 456KB of S_5 has been prefetched, which would minimize the response time of S_5. By Equation 18, the new, prefetched consumption rate of S_5 is $240-456/30 = 225$, assuming that the total length of S_5 is 30 seconds. The question is whether S_5 and S_6 can be activated simultaneously. The answer is no because the total consumption rate $P = 3*240+225+240 = 1185 > 1150$.

Alternatively, assume that 1500KB of S_5 has been prefetched. Then, by Equation 18, the prefetched consumption rate of S_5 is $240-1500/30 = 190$. In this case, the total consumption rate $P = 3*240+190+240 = 1150$ which is $\leq R = 1150$. [4] Thus, as long as there are enough buffers to accommodate S_6, both S_5 and S_6 can be activated, reducing drastically the response time of S_6. Thus, the consumption rate of S_6 can be used to determine an appropriate amount to prefetch for S_5. □

[4] In practice, it is not so simple just to ensure that the total consumption rate is not greater than the maximum reading rate. As shown later in Strategy IP1, what needs to be done is a full admission control test. But here we simplify the situation to illustrate the point that prefetching can lead to the activation of extra queries.

The above example shows that prefetching S_5 for the appropriate amount can lead to a gain for S_5 and other queries in the waiting queue. It also leads to an interesting question: how to distribute prefetching among queries in the waiting queue. In other words, given the same amount of buffer space available for prefetching, how much of each query in the waiting queue should be prefetched so as to maximize the reduction in total consumption rate, thereby maximizing the number of queries that can be activated.

To answer this question, let us consider a "marginal gain" analysis on the buffers, quite similar to the one used in [8]. More specifically, for a query S_i, with an original consumption rate P_i, we calculate the reduction in consumption rate we would obtain if we prefetch one extra KB of S_i. By Equation 18, this value is equal to $P_i - P_i^{pft}$ which is equal to $\frac{1}{T_i}$. Thus, given queries S_1, S_2 whose lengths are T_1, T_2 respectively, prefetching more for the stream whose length is the shorter between T_1 and T_2 would result in a sharper drop in the combined consumption rate of the two streams. In other words, if the combined consumption rate has to drop below a certain value in order to pass admission control, prefetching more for the shorter query would require fewer buffers than prefetching for the longer one. Consider the following example.

Example 4 The previous example shows that in order to activate both S_5 and S_6 after one other query has finished, prefetching S_5 for 1500KB will do. Suppose the length of S_6 is 15 seconds. Then solving the equation $P = 3 * 240 + 240 + (240 - \frac{pf}{15}) = 1150$ indicates that if we prefetch S_6 entirely, only an amount pf of 750KB would be sufficient to activate both S_5 and S_6. Note that this amount is the bare minimum that allows both queries to be activated. If there are extra buffers, we can do more by prefetching one cycle of S_5 as well, so that not only are they activated, but both S_5 and S_6 can also be consumed immediately at the beginning of their first cycle. $\Box$

4.4 Prefetching Strategy IP1

The prefetching strategy below, called IP1 which stands for "Intelligent Prefetching," finds the shortest query to prefetch, so as to maximize prefetching and the number of queries that can be activated once an active query has completed.

Strategy IP1 Let $S_1, \ldots, S_n$ be all the activated queries, as allowed by the admission controller. Among them, let S_j $(1 \leq j \leq n)$ be the query that will finish the earliest. Also let $S_{n+1}, S_{n+2}, \ldots$ be the queries in the waiting queue, and B_{free} be the total number of buffers available to prefetching.

1. Use Equation 21 to determine the length t of the cycle for $S_1, \ldots, S_n$.

2. Initialize $target$ to S_{n+1}, and $candidateSet$ to S_{n+1} as well. Also set $finalAmt$ to 0.

3. (** first chance **)
   If the combined consumption rate of all the streams in $candidateSet$ is not greater than the consumption rate of S_j (i.e. $P_j \geq \sum_{S_k \in candidateSet} P_k$), go to Step 6.

4. (** second chance **)
   Otherwise,

   (a) Calculate the necessary prefetched consumption rate P_{target}^{pft} of $target$ so that all the streams in $candidateSet$ can possibly be activated when S_j has finished, i.e. $P_{target}^{pft} + \sum_{S_k \neq target; S_k \in candidateSet} P_k \leq P_j + (1-\rho) * R$.

   (b) Use Equation 18 to calculate the amount that needs to be prefetched in order to reduce the consumption rate of $target$ to P_{target}^{pft},
   i.e. $targetAmt = (P_{target} - P_{target}^{pft}) * T_{target}$.

   (c) If $targetAmt > B_{free}$, then go to Step 5 to try the next condition.

   (d) Otherwise, use the admission control test given in Section 2.4 to determine if all streams in $candidateSet$, including the prefetched one, can get in a cycle with all the current activated queries except S_j. If the admission control test fails, go to Step 5.

   (e) Otherwise, set $finalTarget$ to $target$ and $finalAmt$ to $targetAmt$. Go to Step 6.

5. (** third and final chance: both Steps 3 and 4 fail **)

   (a) Set $targetAmt$ to B_{free}.

   (b) Use Equation 18 to calculate the prefetched consumption rate P_{target}^{pft} of $target$, i.e. $P_{target}^{pft} = P_{target} - \frac{targetAmt}{T_{target}}$.

   (c) Use the admission control test given in Section 2.4 to determine if all streams in $candidateSet$, including the prefetched one, can get in a cycle with all the current activated queries except S_j. If the admission control test fails, go to Step 7.

   (d) Otherwise, set $finalTarget$ to $target$ and $finalAmt$ to $targetAmt$. Go to Step 6.

6. (** try to see if more queries can be activated **)
   Consider the next query S_{next} in the waiting queue that is not in $candidateSet$. Add S_{next} to $candidateSet$. Compare the length of S_{next} with the length of $target$. Set $target$ to be the stream with the shorter length. Go back to Step 3.

7. (** no more queries can be activated **)
   If $finalAmt > 0$, prefetch $finalTarget$ for the amount $finalAmt$. □

In the above strategy, the purpose of $candidateSet$ is to ensure FIFO in the activation of queries, even though as argued in the "marginal gain" analysis above, it is possible to prefetch S_{k+1} without prefetching S_k. In each iteration of IP1, the stream with the shortest length in $candidateSet$ is chosen to be the $target$ stream for possible eventual prefetching. Then there are three possibilities for all the queries in the $candidateSet$ to be activated,

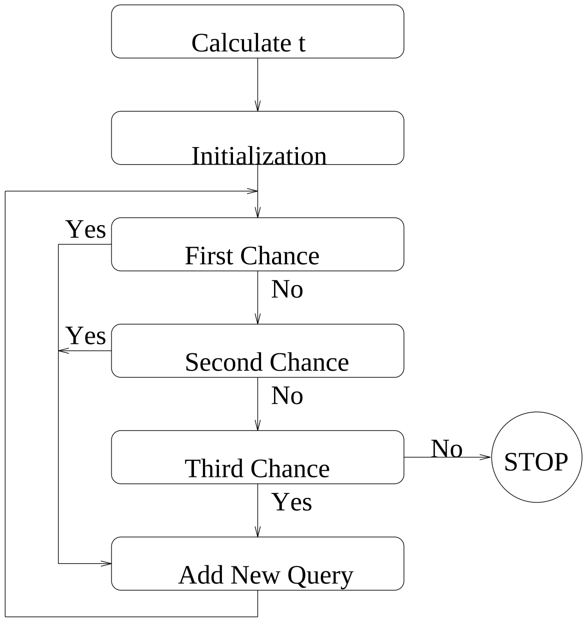


Figure 6: Control Flow of IP1

once S_j has completed (i.e. the next activated query to finish). The first case is when the combined consumption rate of all those in $candidateSet$ does not exceed the consumption rate of S_j. In this case, all queries in $candidateSet$ are guaranteed to be activated once S_j has completed. In addition, nothing needs to be prefetched in this case [5]. Execution then goes to Step 6 to try to see if more queries in the waiting queue can be activated. A new $target$ is found, and a new iteration begins.

If the first condition fails in Step 3, execution goes to Step 4 to see if the second possibility would work out. In this case, IP1 tests if a sufficient amount of $target$ can be prefetched so that all queries in $candidateSet$ can be activated, provided that this amount does not exceed the number of buffers currently available to prefetching (cf. Step 4c). If admission control in Step 4d verifies that all queries can be activated with the help of prefetching, both $target$ and the prefetching amount $targetAmt$ are recorded in the variables $finalTarget$ and $finalAmt$. Execution then goes to Step 6 to try to add another query from the waiting queue to $candidateSet$, and a new iteration begins.

If both the conditions in Steps 3 and 4 fail, IP1 tries the "last resort." It simply tests to see if using all free buffers to prefetch for $target$ will be sufficient to activate all queries in $candidateSet$. If admission control returns a positive answer, all the necessary operations will be taken in Step 5d and 6, and a new iteration begins.

If all three conditions in Steps 3, 4 and 5 fail, it is an indication that not all queries in

[5] This is the case as far as query activation is concerned. But if there are enough buffers available at the end, queries in $candidateSet$ may be prefetched so that they can be consumed immediately at the beginning of their first cycle.

$candidateSet$ can be activated. More precisely, all but the last added query in $candidateSet$ can be activated once S_j has completed. Step 7 prepares for this event by prefetching $finalTarget$ for the amount $finalAmt$. Figure 6 illustrates the main control flow of IP1.

Notice that as presented above, IP1 is only concerned with maximizing the number of queries that can be activated. As discussed in the previous example, IP1 can easily include a Step 8 that would prefetch one cycle worth of data for each query that would be activated, so that every one can be consumed immediately at the beginning of the first cycle. Furthermore, in the case when no query in the waiting queue can be activated even after S_j has completed (i.e. S_{n+1} is the only query in $candidateSet$), another thing Step 8 could do is to use SP to prefetch as much as possible for S_{n+1}. This would take care of the situation when the consumption rate of S_{n+1} needs to be substantially reduced before S_{n+1} can be activated. Last but not least, the admission control used in IP1 above does not consider buffer sharing. Equation 11 can be used in the place of Equation 7 (and thus Equation 8) in admission control, if buffer sharing is used.

Example 5 Let us apply Strategy IP1 to the situation discussed in the previous example. Let us assume that S_1 is the activated query that will finish the earliest. In the first iteration of IP1, S_5 alone is considered in Step 3. Since S_5 has the same consumption rate as S_1, S_5 can certainly take the place of S_1 and be activated once S_1 has completed. Thus, execution goes to Step 6, in which S_6 is added to $candidateSet$. Since S_6's length is shorter than S_5's, S_6 becomes the new $target$.

In the next iteration of IP1, obviously Step 3 fails. Now based on the calculations given in the previous example, the prefetched consumption rate of S_6 is $P^{pft}_{target} = 190$KB/s, and the prefetched amount is $targetAmt = 750$KB. Assuming that the admission control test in Step 4d is passed, $finalTarget$ is set to S_6 and $finalAmt$ to 750KB. Then in Step 6, another query S_7 is added from the waiting queue to $candidateSet$, and a new iteration begins.

Suppose S_7 has the same rate and length as S_6, and is the new $target$. It is not difficult to verify that Steps 3, 4 and 5 fail in this iteration. Thus, execution goes to Step 7, and the final decision is that S_6, which is $finalTarget$, will be prefetched for 750KB. As discussed before, if there is a Step 8 in IP1 to minimize response time, S_5 will also be prefetched so that the consumption of S_5 can start immediately at the beginning of its first cycle. □

4.5 Prefetching Strategy IP2

As introduced above, Strategy IP1 involves testing three conditions (i.e., Steps 3, 4 and 5) to find out if prefetching can take place. However, a closer examination of these conditions reveal that the conditions are "heuristic" in that satisfying the conditions does not necessarily guarantee eventual admissions of the queries in $candidateSet$. For instance, in Step 4 of IP1, the values computed in Steps 4a and 4b may not be good enough to pass the test in Step 4d. A similar comment applies to Step 5. The bottom line is that the computation of IP1 may include many failed attempts in admission control, all of which increase the overhead of running IP1. In developing another intelligent prefetching strategy, which we call IP2, we aim to *couple* the calculation of the prefetched amount $targetAmt$ together with admission control. In particular, as will be shown below in IP2, we develop a quadratic equation in $targetAmt$ so that if there is a positive solution of $targetAmt$, admission control is guaranteed to be successful, and thus no additional admission control test is needed.

To develop the quadratic equation and IP2, we need to consider in greater depth some of the issues related to the prefetched buffers. The main issue is the time to release the prefetched buffers, if some of the activated queries/streams have prefetched buffers. There are two natural possibilities. The first one is to have the amount of prefetched buffers gradually reduced as the stream is being consumed. This option, however, requires quite complicated modeling and implementation. The other possibility is to assume that the amount of prefetched buffers does not change until the stream is completely consumed. Given its simplicity for implementation, we adopt this later option in our analysis.

Let B^{pft} denote the total amount of prefetched buffers belonging to the activated queries. Then the maximum amount of buffers available to prefetching is: $B_{free} = B_{max} - B - B^{pft}$, where as in previous sections, B denotes the total amount of buffers needed for the cyclic reading of the activated streams, and B_{max} denotes the total amount of buffers in the system. To simplify our presentation, we use the variable D to denote $targetAmt$, the target amount for prefetching. Then a necessary condition for prefetching to occur is:

$$D \leq B_{free} = B_{max} - B - B^{pft} \tag{22}$$

Next, to couple admission control directly in the calculation of D, we consider again the admission conditions given in Equations 5 and 8. [6] But now two considerations must be taken into account. First, this is an admission control test to be considered T_{finish} time later, i.e., when one of the activated queries S_j has been completed. Second, the amount of prefetched buffers must be included. These two considerations lead to two modifications to Equations 5 and 8. First, the amount of buffers available to cyclic reading is no longer B_{max} in Equation 8. It should instead be $B_{max} - B^{pft} + B_j^{pft} - D$, where B_j^{pft} denotes the amount of prefetched buffers of the completed stream S_j. Second, the total consumption rates of all the queries is no longer P in both equations. It should instead be:

$$\begin{aligned} P &= \Big(\sum_{i=1, i\neq j}^{n} P_i\Big) + \Big(\sum_{S_k \neq target; S_k \in candidateSet} P_k\Big) + (P_{target} - D/l) \\ &= \Big(\sum_{i=1, i\neq j}^{n} P_i\Big) + \Big(\sum_{S_k \in candidateSet} P_k\Big) - D/l \end{aligned} \tag{23}$$

where l denotes the length of $target$, the stream to be prefetched. It is not difficult to verify that these two modifications give rise to a quadratic inequality in D in the form of $aD^2 + bD + c \geq 0$, where the coefficients a, b and c are defined by:

$$a = \frac{s}{l^2} - \frac{1}{l}$$

[6] Again, if buffer sharing is used by the system, Equation 11 should be used in the place of Equation 8.

$$
\begin{aligned}
b &= \Big(\sum_{i=1, i \neq j}^{n} P_i\Big) + \Big(\sum_{S_k \in candidateSet} P_k\Big) - R + \frac{B_{max} - B^{pft} + B_j^{pft} + sR - 2sP_{target}}{l} \\
c &= (B_{max} - B^{pft} + B_j^{pft})(R - \sum_{i=1, i \neq j}^{n} P_i - \sum_{S_k \in candidateSet} P_k) - \\
&\quad s[\sum_{i=1, i \neq j}^{n} P_i(R - P_i) + \sum_{S_k \in candidateSet} P_k(R - P_k)] \qquad (24)
\end{aligned}
$$

We are now in a position to present Strategy IP2.

Strategy IP2 Replace Steps 4 and 5 of IP1 by:

4. Find out whether there exists a positive solution of D subject to Equation 22 and $aD^2 + bD + c \geq 0$ where a, b and c are defined in Equation 24.
   (a) If such a solution cannot be found, go to Step 7.
   (b) Otherwise, set $finalTarget$ to $target$ and $finalAmt$ to the maximum positive solution of D. Go to Step 6. □

On first sight, the above computation of IP2 involving Equations 22 and 24 seems to be terribly complicated. However, the computing overhead of IP2 is actually much smaller than that of IP1. This is because by computing D directly, the overhead involved in all the failed attempts carried out in Steps 4 and 5 of IP1 can now be avoided. What remains to be seen, however, is whether IP2 is more effective than IP1, or vice versa. In the next section, we will present simulation results that address this issue.

5 Preliminary Simulation Results

5.1 Details of Simulation Package

We have implemented a discrete-event simulation package to evaluate the techniques proposed in this paper. The package runs under Unix on Sparc-stations, and consists of about 5,000 lines of C code. For ease of coding, all the queries to be executed in a simulation are submitted to the waiting queue at the beginning of the simulation. Thus, the main outputs of the simulation package do not include response times of queries, but include such statistics as peak and average disk and buffer utilizations, and the total time to complete all queries. Furthermore, to make our simulations as close to reality as possible, we have implemented the following features in our package.

- As observed in [10], a transient period is required before a new Stream S_{n+1} can be added to a (new) cycle. This is because the new cycle length t' (for one more stream) is strictly larger than the current cycle length t. Thus, if we directly serve S_{n+1} at the end of the current cycle, starvation will occur for all the queries $S_1, \ldots, S_n$ in the current cycle, because they only have data buffered for a cycle of length $t < t'$.

Our simulation package handles the transient period in two steps. First, the cycle for $S_1, \ldots, S_n$ is gradually increased from a length of t to t'. This is only possible if the disk utilization ρ is strictly less than 1 (i.e. there is some free disk bandwidth for increased data reading). Thus, when a cycle length is picked in admission control, ρ cannot be chosen as high as 1. Our simulations indicate that it may take several seconds before the length changes from t to t'. Once the length of the cycle for $S_1, \ldots, S_n$ reaches t', S_{n+1} can be added to the end of the (transient) cycle, in effect starting the new cycle.

- When an activated query has completed, there are two ways to invoke the admission controller. One way is to wait till that particular cycle ends; the other is to wake up the controller immediately after the query has finished (even amidst a cycle). The former policy, while much easier to implement, does not optimize system performance, especially when the cycle length is long and the disk utility is low. We have implemented the latter policy, and found out that system performance is improved.

- As discussed in Section 4.5, there are two ways to release prefetched buffers. One way is to release the buffers gradually at the end of each cycle. The other way is to release the buffers when the query has completed. The latter policy has been implemented.

Apart from making our simulation package as close to reality as possible, we have designed and run our simulations based on real figures (e.g. minimum and maximum seek times equal to 5 and 25ms respectively). We will give further details on all the simulations presented below.

5.2 Effectiveness of Buffer Sharing

In Section 3, we have analyzed that buffer sharing can lead to a 50% reduction in total buffer requirement, when the disk utilization ρ is equal to 1. Here we simulated a situation when ρ keeps changing and has an average value less than 1. In this series of simulation, we used 50 queries, each with consumption rate 240KB/s. The lengths of the queries were from 20 to 120 seconds, with the average being 60 seconds. In order to support a sufficiently high number of concurrent queries, the maximum disk reading rate was set to $R = 2000$KB/s. The graph in Figure 7 shows the minimum buffer space needed, when the number of concurrent queries varies from 3 to 7 – with and without buffer sharing. As expected, in all cases, buffer sharing requires less buffer space than without buffer sharing. The savings in buffer space was between 20% to 40%, depending on the average disk utilization.

5.3 Effectiveness of Prefetching Strategies

In this series of simulation, we evaluated the effectiveness of our prefetching strategies. We again used 50 queries, each with consumption rate 240KB/s, and length 90 seconds. The maximum disk reading rate was set to 1000KB/s. The two graphs in Figure 8 show the time taken to complete the 50 queries and the average disk utilization with varying amounts of buffer space. In both graphs, the x-axis is the amount of buffer space, varying from 5MB to

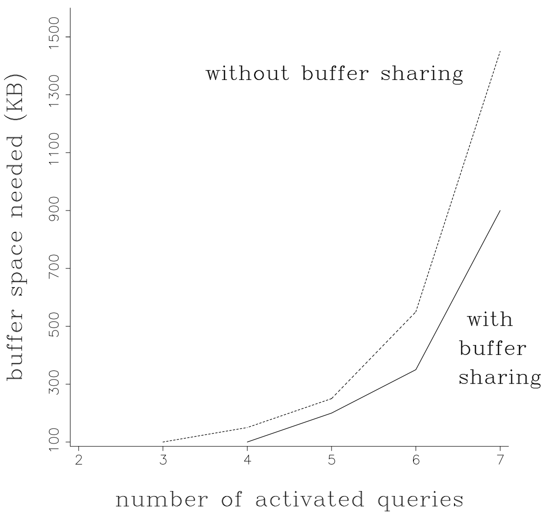


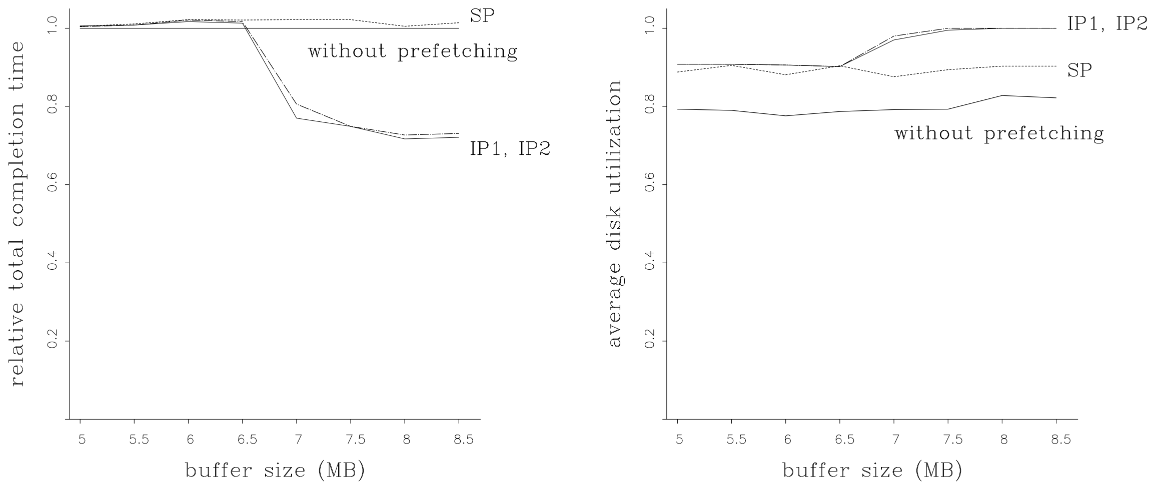
Figure 7: Benefit of Buffer Sharing

8.5MB. In Figure 8(a), the y-axis is the total time taken to complete 50 queries using SP, IP1 and IP2, normalized by the time taken without prefetching. Thus, the horizontal line at 1.0 in Figure 8(a) represents the situation without prefetching. With small amounts of space available to prefetching, IP1 and IP2 do not lead to any gain in performance. However, as more and more space becomes available, IP1 and IP2 are able to activate more and more queries faster than if no prefetching is allowed. Consequently, the total time taken becomes smaller. As shown in Figure 8(a), IP1 and IP2 could lead to a 30% savings in total time taken. Alternatively, the throughput of a system using IP1 or IP2 could be $3/7 \approx 40\%$ higher.

The performance gain caused by IP1 and IP2 can be best explained by the graph in Figure 8(b). If no prefetching takes place, the average disk utilization is around 0.8. But as more buffer space becomes available to prefetching, IP1 and IP2 are able to better utilize the disk by prefetching, and the average disk utilization gradually climbs up to 1.0. Moreover, not shown here, the utilization of buffers follows a similar trend.

Unlike IP1 and IP2, the simple prefetching strategy SP does not perform well at all. As shown in Figure 8(a), not only does it not lead to any reduction in total time taken, but it may also take longer than if no prefetching is allowed. As analyzed in Section 4.3, this is due largely to the fact that SP uses up buffers in an unwise manner. As shown in Figure 8(b), the average disk utilization for SP is still higher than if no prefetching is allowed. This is another indication that while the disk is kept busy by prefetching, the way that SP conducts prefetching is problematic and totally ineffective.

Based on our simulations, IP1 and IP2 seem to be equally effective. This shows that the



(a) relative total time taken

(b) average disk utilization

Figure 8: Effectiveness of SP, IP1 and IP2

heuristic conditions used in IP1 are good enough in most cases. However, we should point out that the actual times used in executing IP1 and IP2 are not included in the simulation. As discussed in Section 4.5, IP2 has less computing overhead than IP1. In a real system, this difference in overhead may have some impact on the overall performance of the system.

The series of simulation discussed above did not allow buffers to be shared. In another series of simulation, we allowed buffers to be shared, and used the version of admission control that is based on Equation 11, but not on Equation 7. The results of this series of simulation were very similar to those presented above. The only difference was that buffer sharing saved a few hundred KBs of buffer space, and made it available to prefetching. Thus, the point when IP1 and IP2 started to show improvement now began a few hundred KBs earlier than was shown in Figure 8(a).

In yet another series of simulation, we used 50 queries with identical consumption rate but different lengths. The results led us to the same conclusion: given sufficient amount of buffer space, IP1 and IP2 eventually outperform SP and the policy without prefetching, by utilizing the disk and the buffers more effectively.

6 Further Extensions and Discussions

6.1 Extension to a Multiple Disks Configuration

In previous sections, we have analyzed buffer sharing and prefetching strategies in a single disk environment. In the following, we will discuss how these proposed ideas can be supported in a multiple disk environment, such as one using a RAID architecture [9].

There are several basic scenarios, depending largely on how the data are placed on disks. If the data are completely replicated on all the disks (such as for a high degree of fault tolerance), then we can simply treat all the disks as one logical disk unit whose combined disk bandwidth is the sum of all individual disk bandwidths. In other words, we can replace R in all our formulas with $R_{all} = \sum R_i$. Similarly, if the data are perfectly stripped on all the disks, the entire collection of disks can be treated logically as a single disk unit. And again, replacing R with R_{all} will be sufficient.

In the following, we will focus our attention on a more complicated situation in which: a) there is no duplication of data and each stream is stored on only one disk; and b) all disks share a common pool of buffers. On one hand, for the sake of optimizing buffer utilization, the number of buffers assigned to each disk should not be fixed a priori. But on the other hand, the buffer manager should enforce some kind of fairness condition to as to prevent a particular disk from snapping up all the buffers. In the simple multiple disks algorithm (denoted by SMD) given below, we use the heuristic that no disk can get more than 50% of the buffers in the entire system.

Except for the sharing of a common pool of buffers, each disk functions independently. Thus, the kind of buffer sharing analyzed in Section 3 can be supported in the current environment in the same way as in a single disk environment. This is, however, not the case for prefetching. As shown in Section 4, one of the most difficult problems in supporting prefetching is to estimate the number of buffers that are available now and that will be available when the new streams are actually admitted. The estimation becomes even more complicated when there are multiple disks constantly changing the number of buffers in use.

The above problem can be dealt with by the introduction of a reservation scheme. At any point in time when a disk D wants any number of buffers, it sends a request to the buffer manager reserving that amount of buffer space, even if the space will not be needed until later. The buffers are released back to the buffer manager for public use, when the query requesting data stored on D is completed. In this way, the buffer manager has an accurate record on the number of occupied/reserved buffers, until the next event occurs to any of the disks.

Apart from the estimation issue, there is also a fairness issue surrounding how to support prefetching in a multiple disks environment. As discussed in Section 4, whenever IP1 or IP2 is invoked, there is a tendency that a lot of buffers will be dedicated to the task. In an environment where multiple disks share a common pool of buffers, how to coordinate prefetching from more than one disk becomes an important question. While we consider a

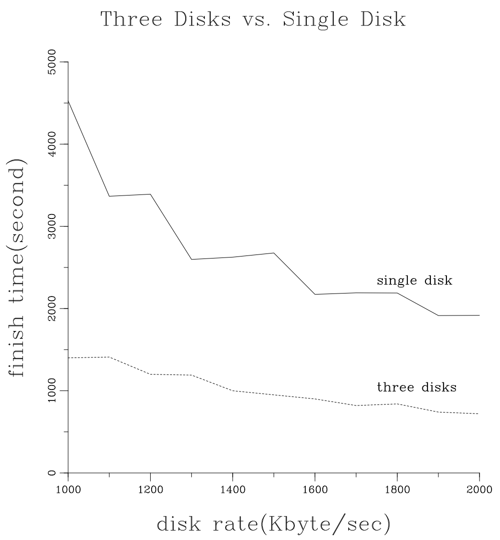


Figure 9: Effectiveness of SMD

more thorough study on this question beyond the scope of this paper, Algorithm SMD below uses the simple policy that at any point in time, only one disk is allowed to prefetch. The following outlines a simple algorithm for multiple disks.

Outline of Algorithm SMD

1. All the queries are submitted to the waiting queue of the system which provides directory services. Then the queries are re-submitted to the waiting queues of the corresponding disks.

2. The system buffer manager uses a reservation scheme to keep track of the buffers (to be) used by each disk. It also enforces some fairness policy, such as not allowing any disk to get more than 50% of all the buffers.

3. The system ensures that the disks take turn to prefetch data. □

Figure 9 gives some preliminary simulation results showing the effectiveness of SMD. In this simulation, we applied SMD to a 3-disk array. The graph in Figure 9 compares the total times needed by the 3-disk array and a single disk to complete 150 queries, each with consumption rate 240KB/s and length 90 seconds. Under SMD, the disk array achieves the expected speedup.

6.2 Approximation for Non-contiguous Data Placement

Recall from previous sections that for the most part, our analyses are based on contiguous data placement. The following shows how to use our equations established so far to handle

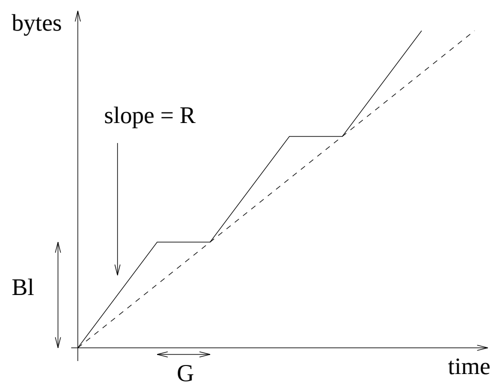


Figure 10: Approximating a Reading Curve with Seeks Between Blocks

the case when data are stored in blocks or clusters of size Bl, and the seek time between adjacent blocks is G. This is a non-contiguous data placement scenario consistent with the ones proposed in [10, 3].

The solid line in Figure 10 represents the reading curve of a stream of the kind described above. To use the equations that we have developed so far, we can approximate the given reading curve by one that assumes contiguous placement. This approximate curve is a straight line that is always below the original reading curve, but with a slope as large as possible. The dotted line in Figure 10 represents the approximate curve. By a simple coordinate-geometry analysis, the dotted line has a slope R_{approx} given by:

$$R_{approx} = \frac{Bl}{G + Bl/R} \tag{25}$$

This value can be used to replace R in all the equations that we have encountered so far. Note that since the approximate reading curve is always below the actual reading curve, at various points in time, the disk may read faster than approximated. Thus, the continuity requirement is not violated by the approximation, and there is no need to worry about starvation. A similar approximation can be applied to other non-contiguous data placement situations. We omit those analyses here.

In this series of simulation, we aimed to show that non-contiguously placed streams can be benefited by the above approximation, which makes them amenable to buffer sharing (and the kinds of prefetching proposed here). In particular, we repeated the simulation described in Section 5.2 with two different groups of queries. The first group consists of streams (queries) that were non-contiguously placed in blocks of size $Bl = 20$KB (i.e. roughly one track), with each block separated by a gap $G = 5$ms. The second group consists of the contiguous streams that used Equation 25 to approximate the non-contiguous streams in the first group. Queries of the second group were allowed to share buffers. Analogous to Figure 7, the following table shows the minimum buffer space (in KB) needed for both kinds of queries, when the number of concurrent queries varies between 3 and 5.

number of concurrent queries	3	4	5
non-contiguous streams	130	360	2520
approx. streams with buffer sharing	110	250	1420

In all cases, the entries in the table show that it is beneficial to approximate non-contiguous streams with contiguous ones, and if allowed to share buffers, the approximating streams can lead to reduction in total buffer requirement.

6.3 Discussions: Applicability of Prefetching to General Multimedia Systems

Our preliminary simulation results indicate that appropriate prefetching can lead to increase in throughput, disk utilization and buffer utilization. However, in order to have higher throughputs and thus lower response times of queries, the price to pay is certainly availability of buffer space. As shown in our examples and simulation results, we believe that the price is not high – provided that the streams are short, say below 5 minutes in length. As far as news on-demand systems are concerned, a large class of news clips falls within this range. However, a natural question to ask is whether prefetching has a role to play in other multimedia systems.

Consider multimedia database management systems. We believe that prefetching indeed has a major role to play in tuning the performance of such systems. This is because for a large class of applications, the audio and video components tend to be short. For example, for applications such as the one described in [5], audio and video may not be the only media, and may work hand-in-hand with other media such as text and images. Audio and video components may also play the role of annotations or illustrations. Moreover, many applications may require frequent user interaction.

What about the other extreme: movies on-demand systems? Unlike those cases discussed above, movies on-demand is concerned with supplying video and audio data to users for long durations and with relatively little user interaction. By Equation 18, reducing the consumption rate of a movie by just 1KB/s requires T KB buffer space, where T is the length of the query in seconds. For example, if a movie is 90 minutes long, this amount of buffer space is already 5.4MB. And to reduce the consumption rate by 50KB/s (as in Example 5), 270MB of buffer space is needed! As shown in Equation 18, the amount of buffer space needed for prefetching (and IP1, IP2) to work is linearly proportional to the length of the movie. However, on the positive side, consider the benefit of prefetching. Recall that prefetching has the effect of activating as many queries (movies) as possible. If prefetching is not used, and a movie M_0 cannot be activated immediately, it has to wait for an activated movie M_1 to finish. Thus, the waiting time of M_0 depends linearly on the length of M_1. In other words, if prefetching is the difference between whether a movie can or cannot be activated immediately, the difference in response time, like the amount of buffer space needed, is linearly proportional to the length of the movies. As an example, this difference in response time may be 30 minutes. Thus, while long queries magnify the buffer space needed for prefetching to work, they also magnify the benefits of prefetching. It is certainly up to an enterprise to decide which is more important and costly: 270MB or 30 minutes.

Two future developments, we believe, may increase the applicability of prefetching. The first one is the obvious advancement in hardware technology. In a few years, 270MB, for example, will cost orders of magnitudes lower than it costs now. The second future development may be one that is concerned with the time when prefetched buffers are released. The prefetching strategies presented here are based on the assumption that the prefetched buffers of a query are released when the query has completed. Alternatively, prefetched buffers can be released gradually, while the retrieval and consumption of the query are still going on. This policy would reduce the total number of buffers needed by a wide margin. (A preliminary analysis suggests a reduction of more than 50%.) However, to support this policy, we would need to introduce a new dimension of time into our analysis and many of our formulas for prefetching. This is a topic of our ongoing research.

7 Conclusions

Providing effective multimedia support in database management systems is a topic of great interest and value. In this paper, we consider one of the key problems encountered in such systems. Given a fixed amount of buffer space and disk bandwidth both pre-determined at design time, we study how to maximize the throughput of the system. Our approach is to maximize the utilizations of buffers and disk. To achieve this goal, we have first proposed a buffer sharing scheme. Analysis and simulation results indicate that buffer sharing could reduce total buffer consumption by as much as 50%. Second, we have developed the prefetching strategies IP1 and IP2 which aim to maximize prefetching and the number of queries that can be activated. Preliminary simulation results show that IP1 and IP2 could be quite effective in maximizing the effective use of buffers and disk, and could lead to a 40% increase in system throughput. Finally, we have also outlined how to support the proposed techniques in a multiple disk environment and with non-contiguous data placement.

In ongoing work, we are studying how to implement the proposed techniques in a distributed continuous-media file system [7]. Key issues to be addressed include how to extend the proposed techniques to support network buffering, and how to effectively implement prefetching and buffer sharing, when the reading orders from one cycle to the next can or cannot be changed. It is also important to study how to support the proposed techniques effectively in a multiple disk environment.

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] R. Goldman-Segall. (1990) *Learning Constellations: a Multimedia Research Environment for Exploring Children's Theory-Making*, Constructionist Learning, ed. I. Harel, Cambridge, MA, MIT Media Laboratory.

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

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

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

[9] D. Patterson, G. Gibson and R. Katz. (1988) *A Case for Redundant Arrays of Inexpensive Disks (RAID)*, Proc. ACM-SIGMOD, pp 109–116.

[10] 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.

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

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

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

[14] 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.