Minimizing Memory Requirements in Media Servers

Edward Chang and Yi-Yin Chen *

Abstract

Poor memory management policies lead to lower throughput and excessive memory requirements. This problem is aggravated in multimedia databases by the large volume and real-time data requirements. This study explores the temporal and spatial relationships among concurrent media streams. Specifically, we propose adding proper delays to space out IOs in a media server to give more room for buffer sharing among streams. Memory requirements can be reduced by trading time for space. We present and prove theorems that state the optimal IO schedules for reducing memory requirements for two cases: streams with the same required display rate and different display rates. We also show how the theorems can be put in practice to improve system performance.
Keywords: multimedia, disk scheduling, memory requirement.

1 Introduction

Poor memory management policies lead to lower throughput and excessive memory requirements. This problem is exacerbated in multimedia databases by the large volume of the media data, which must be delivered in time to avoid display disruptions. This real-time requirement is fulfilled by prefetching and staging large amount of data in memory to sustain continuous data consumption between disk IOs. A few seconds of excessive prefetching or inefficient management of the memory buffer pool incurs a large increase in the memory requirement or decrease in the number of concurrent streams the limited memory can support.

Most research of multimedia storage system has focused on reducing disk delays via disk arm scheduling policies and data placement schemes [5, 6, 8, 10, 12]. In this paper, on the other hand, we focus more on the effective use of memory. It turns out that in many circumstances memory is more of a critical resource than disk bandwidth, and using memory judiciously can lead to improved throughput, lower hardware costs, and reduced initial latencies, as compared to conventional strategies.

In particular, the contributions of this paper are as follows.

- We formally show that the amount of memory needed to support a given number of

*Stanford University, Department of Electrical Engineering, Gates Hall 5, Stanford, CA 94305. email: echang@db.stanford.edu, yychen@leland.stanford.edu

concurrent streams can be minimized by spacing out in time the IOs performed in each round of service. We present and prove theorems that state the optimal IO schedules for two cases: streams with the same required display rate and different display rates.

- We propose a disk scheduling algorithm that adds proper delays to space out IOs to give more room for buffer sharing among concurrent streams. This approach runs counter to traditional schemes that squeeze out the last drop of disk bandwidth to improve throughput. But surprisingly, wasting disk bandwidth leads to improvement in throughput. This result suggests that with multimedia storage systems, unlike traditional file systems, reducing seek overhead alone does not yield better performance.

The rest of this paper is organized as follows. Section 2 formulates the quantitative model for memory requirements. Section 3 proves the the theorems which state the conditions that minimize the variance of memory requirement. Section 4 presents a disk scheduling scheme, called Stretch, that stretches out IOs to minimize memory use. Finally, we evaluate Stretch in Section 5.

2 Memory Model

2.1 Definitions

To analyze the performance of a media server, we typically are given the following parameters:

- TR: the disk's data transfer rate.
- $\gamma(d)$: a concave function that computes the rotational and seek overhead given a seek distance d. For convenience, we will refer to the combined seek and rotational overhead as the seek overhead.
- Mem_{Avail}: the storage system's available memory.
- N: the number of stream requests. Each request is denoted as R_1, R_2, ..., R_N. Each stream requires a display rate of DR_i ($DR_i < TR$).

We also have the following tunable parameters. They can be adjusted, within certain bounds, to optimize system throughput.

- T: the period for servicing a round of requests. We assume that T is constant, i.e., it does not vary depending on N, the number of streams being serviced at a given time. As discussed below, T must be made large enough to accommodate the maximum number streams we expect to handle.

Parameter	*Description*
Mem_{Avail}	Total available memory, MBytes
DR	Data display rate, Mbps
TR	Disk transfer rate, MBytes/s
CYL	Number of cylinders on disk
$\gamma(d)$	Concave function for rotational and seek overhead give n distance d
T	Service time for a round of N requests
N	Number of requests being serviced
N_{Limit}	System enforced limit on number of requests
R_i	i_{th} request
cyl_i	Seek distance of the ith request
$K_{Cushion}$	Cushion buffer factor, ranging from 1 to 2.
$K_{MSharing}$	Memory sharing factor, ranging from 1/2 to 1.

Table 1: Parameters

- S: the segment size, i.e., the number of bytes read for a stream with one contiguous disk IO.

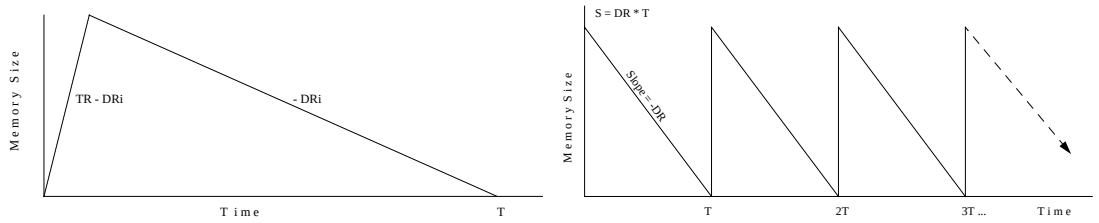
- N_{Limit}: the maximum number of concurrent requests the media server allows. The media server implements an admission control policy that turns away requests when the system is already handling N_{Limit} requests.

To assist the reader, Table 1 summarizes these parameters, together with other parameters that will be introduced later. The first portion of Table 1 lists the basic fixed and tunable parameters. The second portion describes subscripted parameters that are used for characteristics of individual requests. The third portion gives performance measurement parameters.

2.2 Memory Requirements

Figure 1(a) depicts the amount of memory used by a request in a period T. An IO starts shortly before the data staged into memory in the previous period is used up. The data accumulates in memory at the rate of $TR - DR$ until the IO completes. This periodic behavior repeats itself until the playback ends.

For our analysis we make two simplifying assumptions. First, we assume that memory can be freed in a continuous fashion. In other words, Figure 1(a) shows the actual memory used by a request. In practice, of course, memory is released in pages, so Figure 1(a) would have a sequence of small decreasing steps, each one page in size. This implies that our estimates for memory use may be up to one memory page off for each request. Thus, our continuous release assumption is an optimistic one for buffer sharing schemes. However, if as expected the page size is small compared to the segment size, the difference will be negligible.



(a) Memory Required in A Period (b) Memory Use Function $p_i(t - \tau_i)$

Figure 1: Memory Requirement Function

Our assumption causes us to overestimate memory use: we will assume that each peak in Figure 1(b) is S, while in reality it is $S \times (1 - DR/TR)$. This is a pessimistic assumption, but since typically the data transfer rate TR is much larger than the display rate DR, the difference is very small.

Notice that the small differences caused by our two assumptions tend to cancel out each other. In particular, if the page size is $S \times DR/TR$, the effects will cancel. If the page size is less than this value, as is probably the case, then overall our results will be slightly pessimistic for memory sharing.

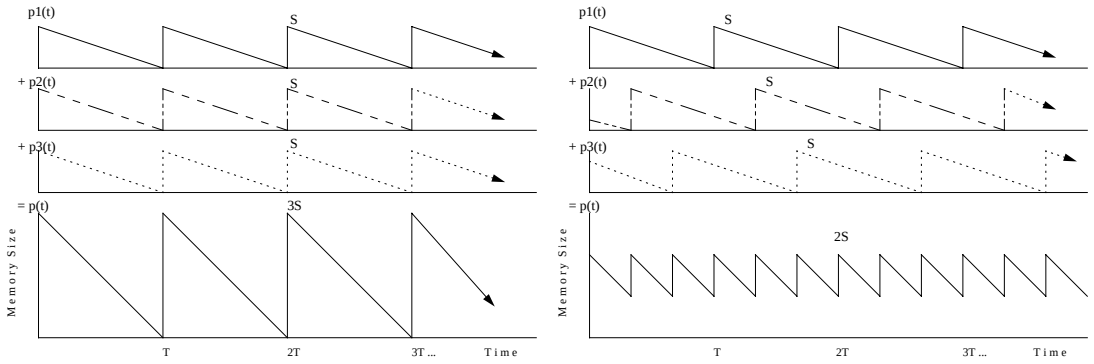
When an IO is initiated, the physical memory pages for the data it reads may not be contiguous due to the way buffers are shared. There are several ways to handle these IOs. One idea is to map the physical pages to a contiguous virtual address, and then initiate the transfer to the virtual space (if the disk supports this). Another idea is to break up the segment IO into multiple IOs, each the size of a physical page. The transfers are then chained together and handed to an IO processor or intelligent DMA unit that executes the entire sequence of transfers with the same performance as a larger IO. Other ideas are discussed in [7].

Let us denote the periodic function in Figure 1(b) as $p_i(t - \tau_i)$, where t represents time and τ_i is the displacement from the beginning of the period (e.g., the example shown in Figure 1(b) has a displacement of 0). The memory use function $p(t)$ for N_{Limit} concurrent requests is a superposition of N_{Limit} such periodic functions, or

$$p(t) = \sum_{i=1}^{N_{Limit}} p_i(t - \tau_i).$$

Notice that each function $p_i(t - \tau_i)$ has a different displacement. To minimize the memory requirement of a system, one has to minimize the largest value of $p(t)$. Since $p(t)$ is periodic (superposition preserves periodicity), it is adequate to just inspect a period of it.

To show the intuition behind the buffer space minimization, let us first illustrate through an example. Figure 2(a) presents an example function $p(t)$ composed of three requests, $p_1(t - \tau_1)$, $p_2(t - \tau_2)$, and $p_3(t - \tau_3)$. In the figure, $\tau_1 = \tau_2 = \tau_3 = 0$, so the maximum value of $p(t)$



(a) Worst Memory Requirement (b) Minimum Memory Requirement

Figure 2: Memory Requirement Examples

is $3 \times S$. In general, the memory requirement is the highest when the peaks of the requests overlap, e.g., when $\tau_1 = \tau_2 = \tau_3$. In practice this is impossible since the requests cannot use the disk arm at the same time to perform their IO. However, this indicates that when IOs are closer to one another, $p(t)$ has a larger peak and hence requires that the system have a larger memory to handle the worst case. On the other hand, when the IOs are separated by time, there is more space to share, and hence the total memory requirement is smaller. We next show how to separate the requests in time to minimize memory usage.

3 Minimizing Memory Requirement

We have demonstrated that the memory requirement is reduced when the delays are added to separate I/O's. This section states and proves the optimal delays that minimize memory space requirement. First we prove for a simplified case where all requests share the same display rate (or data consumption rate). Applying the same procedure we subsequently provides an approximation for the general case where requests have different display rates.

3.1 Constant Display Rate

[Theorem 1] We are given a multimedia storage system that supports N_{Limit} continuous streams with equal display rate DR. Minimizing memory usage requires the IO start times to be spaced equally in T.

[proof]: Consider the periodical function $p_i(t - \tau_i)$ shown in Figure 1(b). Let τ_i be 0 for now for simplicity (add it back shortly). After normalize the period of the function T to 2π, the

function is expressed analytically in the region of $(0, 2\pi)$ as:

$$p_i(t) = S - \frac{S \times t}{2\pi}$$

where $S = DR \times T$. The period is 2π and hence the fundamental frequency ω_0 is given by

$$\omega_0 = 2\pi f = \frac{2\pi}{T} = 1.$$

It therefore follows that Fourier series [13] will consist of angular frequency components $\omega = 1, 2, 3, etc.$, and the series is

$$\begin{aligned} p_i(t) = \frac{a_0}{2} + a_1 \cos t + a_2 \cos 2t + ... + a_n \cos nt + ... \\ + b_1 \sin t + b_2 \sin 2t + ... + b_n \sin nt + ... \\ = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos nt + b_n \sin nt. \end{aligned} \tag{1}$$

The various coefficients in this series can be evaluated by using the following equations:

$$\begin{aligned} a_0 = \frac{2}{T} \int_{-T/2}^{T/2} p_i(t) dt \\ a_n = \frac{2}{T} \int_{-T/2}^{T/2} p_i(t) \cos n\omega_0 t dt \\ b_n = \frac{2}{T} \int_{-T/2}^{T/2} p_i(t) \sin n\omega_0 t dt. \end{aligned} \tag{2}$$

Substituting the expression for $p_i(t)$ in Equation 2, we get

$$\begin{aligned} a_0 = S \\ a_n = 0 \\ b_n = \frac{S}{\pi n}(-1)^n. \end{aligned}$$

It is evident that all cosine terms are zero. The required series using the coefficients in Equation 1 is now given by

$$p_i(t) = \frac{S}{2} + \frac{S}{\pi} \sum_{n=1}^{\infty} \frac{(-1)^n}{n} \sin nt. \tag{3}$$

The function $p_i(t)$ has components of frequency $\omega = 1, 2, 3, ..., etc$. Notice that the amplitude of the components is inversely proportional to the frequency, so the magnitude of lower-frequency components is larger than that at the higher frequencies.

Now adding the delay factor τ_i back into Equation 3 yields

$$p_i(t - \tau_i) = \frac{S}{2} + \frac{S}{\pi} \sum_{n=1}^{\infty} \frac{(-1)^n}{n} \sin n(t - \tau_i). \tag{4}$$

Recall that $p(t)$ is the superposition of N $p_i(t-\tau_i)'s$, subsequently $p(t)$ is written as

$$p(t) = \frac{SN}{2} + \frac{S}{\pi}\sum_{i=1}^{N}\sum_{n=1}^{\infty}\frac{(-1)^n}{n}\sin n(t-\tau_i) \tag{5}$$

The objective is to minimize the maximum value of $p(t)$. But as we have mentioned, since $p(t)$ is discontinuous, it is cumbersome to solve. Further, there is no close form solution to the minimax problems. Instead, we change the objective to minimizing $VAR_{p(t)}$. For the constant display rate case, these two objectives are equivalent.

1. The variance of $p(t)$ according to the definition is

$$VAR_{p(t)} = \int_{t=-\infty}^{\infty}(p(t)-\mu_{p(t)})^2 dt.$$

   For convenience, we use $p'(t)$ to represent $p(t)-\mu_{p(t)}$, which gives us

$$VAR_{p(t)} = \int_{t=-\infty}^{\infty} p'(t)^2 dt,$$

$$where\ p'(t) = \frac{S}{\pi}\sum_{i=1}^{N}\sum_{n=1}^{\infty}\frac{(-1)^n}{n}\sin n(t-\tau_i) \tag{6}$$

2. $\int_{t=-\infty}^{\infty} p'(t)^2 dt$ can be written as $\int_{t=-\infty}^{\infty} |p'(t)|^2 dt$, the autocorrelation of $p'(t)$, since $p'(t)$ is a real function.

3. The autocorrelation of a function in time domain is the power spectrum in frequency domain according to the Autocorrelation Theorem [2] [9]. We can write

$$\int_{t=-\infty}^{\infty} |p'(t)|^2 dt = \int_{f=-\infty}^{\infty} |P'(f)|^2 df,$$

   where $P'(f)$ is the Fourier Transform of $p'(t)$.

Summarize the steps above, the objective of the optimization is now transformed into minimizing the power spectrum of the function in frequency domain. Performing Fourier Transform on $p'(t)$, we obtain $P'(f)$ as

$$P'(f) = \frac{S}{2\pi}(\sum_{n=1}^{\infty}\frac{(-1)^n}{n}(\delta(f+\frac{n}{2\pi}) - \delta(f-\frac{n}{2\pi}))(\sum_{i=1}^{N} e^{-j2\pi\tau_i f}). \tag{7}$$

The variance of $p(t)$ is equal to the power spectrum of $p'(t)$ in frequency domain, or

$$VAR_{p(t)} = \int_{f=-\infty}^{\infty} |P'(f)|^2 df = \frac{S^2}{2\pi^2}\sum_{n=1}^{\infty}(\frac{1}{n^2} \times |\sum_{i=1}^{N} e^{-j\tau_i n}|^2) \tag{8}$$

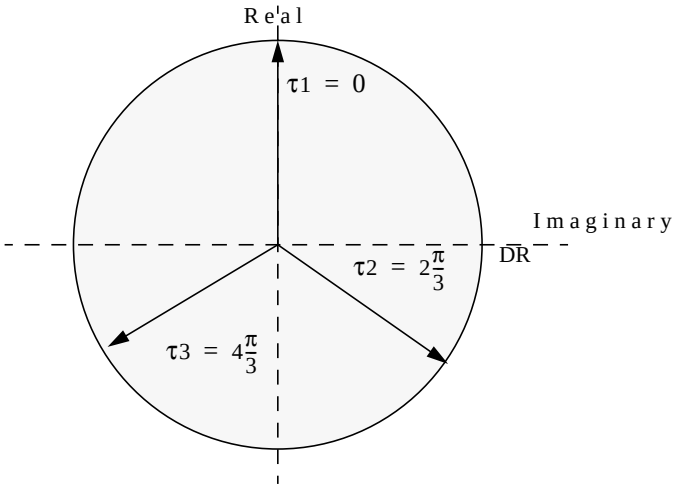


Figure 3: Complex Exponentials

Because S and n are given, the only parameters that can be changed to influence $VAR_{p(t)}$ are those $\tau_i's$. Isolating the terms that we can influence $VAR_{p(t)}$, we get

$$\sum_{n=1}^{\infty} \frac{1}{n^2} \times |\sum_{i=1}^{N} e^{-j\tau_i n}|^2. \tag{9}$$

Minimize $VAR_{p(t)}$ is now reduced to minimizing above Expression 9. The expression reveals that its first harmonic components dominates due to the $\frac{1}{n^2}$ multiplier. Indeed, the coefficient of first harmonic component is 1 and the sum of the coefficients of the rest harmonic components is only $\frac{\pi^2}{8}$. The first harmonic component carries a weight of about 50%. Replacing n with 1 in Expression 9 we get the first harmonic component:

$$|\sum_{i=1}^{N} e^{-\tau_i}|^2. \tag{10}$$

To minimize $VAR_{p(t)}$ it is sufficient to accomplish

$$Minimize\ |\sum_{i=1}^{N} e^{-\tau_i}|^2. \tag{11}$$

Expression 11 is the magnitude of the sum of N complex exponentials. To minimize the sum of N complex exponentials, we want pick $\tau_i's$ that cancel out the amplitude in the complex exponential plane. For equal amplitudes, this is to choose $\tau_i's$ that equally divide 2π. Since we have previously normalized T to 2π, the $\tau_i's$ that divides T evenly minimizes $VAR_{p(t)}$.

For the constant display rate case, this strategy works not only for $n = 1$ (the first harmonic component) but also for the $n's$ that are not a multiple of N. Use Figure 3 to illustrate ($N = 3$), the $\tau_i's$ that makes the first harmonic component zero is $\{0, \frac{2\pi}{3}, \frac{4\pi}{3}\}$. Plugging this set of $\tau_i's$

into the second harmonic component, we get a new delay set $\{0, \frac{4\pi}{3}, \frac{8\pi}{3}\}$ that is equivalent to $\{0, \frac{4\pi}{3}, \frac{2\pi}{3}\}$. Notice that although the delays of R_2 and R_3 change their position, the sum nevertheless remains zero for the second harmonic component. This is not true though when n is a multiple of N. When n is dividable by N, all vectors are shifted to the zero phase position to re-enforce one another. However, since the components we eliminate clearly dominates, eliminating the first harmonic component minimizes the variance.

For the constant display rate case, minimizing the variance happens also to minimax $p(t)$. Since $p(t)$ under this special case is a superposition of N identical functions that separated by an equal distance $\frac{T}{N}$, it is easy to see that the optimal $p(t)$ has N identical peaks. Changing any $\tau_i's$ would cause the minimax value of $p(t)$ to be larger. Subsequently, we claim that minimizing $VAR_{p(t)}$ minimizes the memory requirement for the constant display rate case. We have therefore proven that spacing out I/O's equally in T minimizes the memory requirement. □

[Corollary 1] The minimum memory space required to support N_{Limit} streams with the same display rate DR in given period T is $\frac{S(N+1)}{2}$.

[Proof]: With N_{Limit} requests, $p(t)$ is

$$p(t) = \sum_{i=1}^{N_{Limit}} p_i(t - \tau_i),$$

$$where \quad p_i(t - \tau_i) = DR \times T - DR \times (t - \tau_i).$$

The minimum memory requirement is the maximum value of $p(t)$. Since Theorem 2 shows that spreading out IOs evenly in T minimizes the total memory requirement, we substitute the proper $\tau_i's$ into function $p(t)$ to get

$$p(t) = \sum_{i=1}^{N_{Limit}} p_i(t - \frac{i \times T}{N_{Limit}}),$$

$$\text{where} \quad p_i(t - \frac{i \times T}{N_{Limit}}) = DR \times T - DR \times (t - \frac{i \times T}{N_{Limit}}). \tag{12}$$

Because $p_i(t - \tau_i)'s$ are monotonically decreasing, the start time of IOs gives the maximum value of $p(t)$. In addition, since all $p_i(t - \tau_i)'s$ have the same shape due to the same display rate, all IO start times give the same maximum value. Without loss of generality, let us pick $t = T$ and substitute $t = T$ back into Equation 12, obtaining

$$Max \quad p(t) = \frac{DR \times T}{N_{Limit}} \times \sum_{i=1}^{N_{Limit}} i$$

$$\text{or} \quad Max \quad p(t) = DR \times T \times (N_{Limit} + 1)/2.$$

Since $DR \times T = S$, the above expression is equivalent to $S \times (N_{Limit} + 1)/2$. This gives us our minimum memory requirement. □

Let us now revisit the example of Figure 2(a). In that example we had three requests, $p_1(t - \tau_1)$, $p_2(t - \tau_2)$, and $p_3(t - \tau_3)$. To minimize the memory requirement, we can choose $\tau_1 = 0$, $\tau_2 = \frac{T}{3}$, and $\tau_3 = \frac{2T}{3}$ to equally divide up T. Figure 2(b) plots the memory requirement for $p(t)$ by summing up the $p_i(t)'s$ with these displacements. We can see that as predicted by Corollary 1, the peak memory requirement is $2 \times S$. For larger number of requests, we can cut memory use by one half.

3.2 Different Display Rates

Supporting different display rates becomes increasingly important since a multimedia database or file system contains different types of media (e.g., audio and video). Even if a media server stores compressed videos of the same type (e.g. MPEG-2), the increasing demands of adaptive rate control requires the video data to be delivered in layers of coding depending on the *quality of service* (QoS) requirements [3]. The following theorem states the condition that minimizes the memory requirement variance when rates are different.

[Theorem 2] We are given N_{Limit} continuous streams, each with a possibly different display rate $DR_i's$. Minimizing the memory usage is equivalent to minimizing $|\sum_{i=1}^{N_{Limit}} DR_i \times e^{-j\tau_i}|^2$.

Please refer to reference A for a complete proof.

4 Putting Theorems into Practice

In this section we show how the theorems can be used to enhance media server performance. We first present a disk policy for media server that have the same display rate. We show how the memory can be saved by trading time for space. In the second part of this section, we show a simple augment to a typical admission policy. We demonstrate through an example that delaying IOs admits more requests.

4.1 Scheme Stretch

The goal of scheme Stretch is to minimize memory usage by spacing out IOs in a period as suggested by Theorem 1. Since the data on disk for the requests are not necessarily separated by equal distance, we must add time delays between IOs to space them equally in time.

For instance, if the seek distances in a disk sweep are cyl_1, cyl_2,..., and $cyl_{N_{Limit}}$ cylinders, and cyl_i is the maximum of these, then we must separate each IO by at least the time it

takes to seek to and transfer this maximum i^{th} request. One can choose a different separator for each period, depending on the maximum seek distance for the requests of that period. However, as we have argued earlier, there is no benefit allowing T to vary from cycle to cycle. To have a constant T and simplify the algorithms, scheme Stretch uses the worst possible seek distance (CYL) and rotational delay, together with a segment transfer time, as the universal IO separator, Δ, between any two IOs. The length of a period, T, will be N_{Limit} times Δ. The value of Δ must be at least as large the worst seek overhead plus the worst rotational delay to cover the worst case.

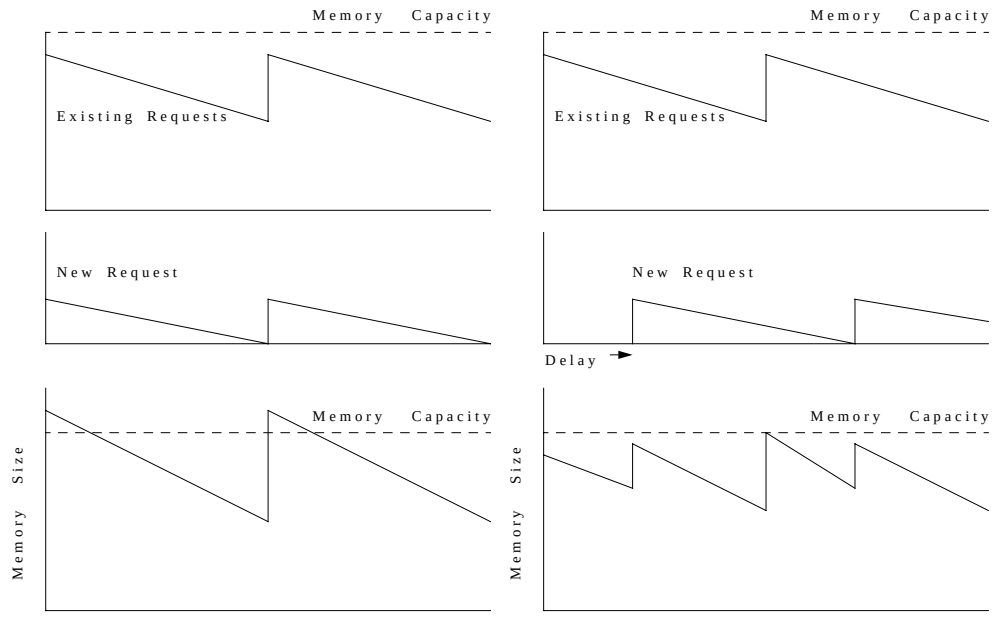
At the first glance, compared with an elevator-like disk scheduling policy (hereafter we call *Sweep*) that amortizes disk latency among requests, scheme Stretch seems like a terrible idea. Here, we first explain what constitutes the memory requirements in a media server to show why the worst disk latency could possibly lead to better system performance. In next section we evaluate Stretch against Sweep with real disk parameters.

The memory requirements of a media server depends on three factors: disk latency, IO time variability, and memory sharing. In the study of [4], we show that the segment size S is directly proportionally to the average disk latency $\gamma(d)$. It is clear that Stretch suffers from the worst $\gamma(d)$ since its d is large, and subsequently requires a larger S. However, Stretch benefits from the other two factors:

1. No IO time variability: For Sweep, since the IO service order depends on the media data's on-disk order, a request can be serviced in different order from period to period. For example, if the IOs for a request is serviced in the beginning of one period, and at the end of next period, the system needs $2 \times S$ amount of buffer for each request [15]. Stretch, on the other hand, since services requests in a fixed order, it eliminates the "$2\times$" factor in front of the segment size.

2. Maximum memory sharing: Theorem 1 shows that when IOs are spaced out equally like in Stretch, the memory saved is about 50% through buffer sharing. On the other hand, Sweep that maximizes disk bandwidth utilization suffers from the worst memory sharing because IOs are bundled.

In short, scheme Stretch saves memory in two ways. First, because IOs in a period are spaced out, memory sharing is at its best. Second, because there is almost no time variability between the IOs of a given request, we eliminate cushion buffers. Stretch does require larger segments (T is artificially enlarged, and $S = DR \times T$), but in our analysis we will see that overall Stretch does save substantial amounts of memory and actually leads to improved throughput over elevator-like disk policy! This result is surprising since Stretch underutilizes bandwidth by slowing down the disk, doing just the opposite of what previous scheduling schemes do.



(a) New Request Rejected (b) New Request Accepted

Figure 4: Delayed Admissions

4.2 Delayed Admissions

For a media server that has to entertain variable bit rates, the analysis is more complex. However, through an example we illustrate that by delaying IOs, a media server can admit more concurrent requests, and hence improve throughput.

A typical media server's admission policy determines if a request can be serviced by checking if the disk bandwidth and memory space are sufficient for servicing the additional request. Figure 4(a) shows that a new request arrives when the memory space left is inadequate to service the request immediately. (The superposition of the peak memory requirements exceeds the capacity.) Most admission control policy would turn the request away. However, Figure 4(b) shows that if the request is delayed by a fraction of T, then the memory is sufficient to admit the request.

5 Evaluation

In this section, through a case study we evaluate the performance of scheme Stretch. For our evaluation we use the Seagate Barracuda 4LP disk [1]; its parameters are listed in Table 2.

For the seek overhead we use the following concave function [11]:

$$\gamma(d) = \alpha 1 + (\beta 1 \times \sqrt{d}) + 8.33 \text{ if } d < 400$$

Parameter Name	*Value*
Disk Capacity	*2.25 GBytes*
Number of cylinders, CYL	*5,288*
Min. Transfer Rate TR	*75 Mbps*
Max. Rotational Latency Time	*8.33 milliseconds*
Min. Seek Time	*0.9 milliseconds*
Max. Seek Time	*17.0 milliseconds*
$\alpha 1$	*0.6 milliseconds*
$\beta 1$	*0.3 milliseconds*
$\alpha 2$	*5.75 milliseconds*
$\beta 2$	*0.0021 milliseconds*

Table 2: Seagate Barracuda 4LP Family Disk Parameters

$$\gamma(d) = \alpha 2 + (\beta 2 \times d) + 8.33 \text{ if } d \geq 400$$

Note that the seek time is proportional to the square root of the seek distance when the distance is small, and is linear to the seek distance when the distance is large. We derive the parameters for this function as follows. We first allocate 2/3 of the minimum seek time provided by the vendor (0.9 ms) as the disk arm's fixed overhead $\alpha 1$ (which includes the speedup, slowdown, and settle phases). Parameter $\beta 1$ then is the remaining portion of the minimum seek time. We then select $\alpha 2$ and $\beta 2$ so that the maximum seek time matches the manufacture's time (17 ms), and so that function γ is continuous at $d = 400$. The values obtained are given in Table 2.[1]

In each seek overhead we have included a full disk rotational delay T_{Rot} of 8.33 ms. The rotational delay depends on a number of factors, but we believe that one rotation is a representative value. One could argue that rotational delay could be eliminated entirely if a segment is an exact multiple of the track size. (In that case we could start reading at any position of the disk.) However, the optimal segment size depends on the scenario under consideration, so it is unlikely it will divide exactly into tracks. If we assume that the first track containing part of a segment is not full, then in the worst case we need a full rotation to read that first portion, even with an on-disk cache. If we assume that the last track could also be partially empty, then we could need a second rotational delay, and our 8.33ms value may be conservative! Note incidentally that we use a *full* rotational delay [2] (not average) since we are estimating a worst case scenario.

[1] Incidentally, [11] suggests using between 200 *to* 600 cylinders to separate short and long seeks. Although we do not show it here, our results are not very sensitive to the exact value used in this range.

[2] It is important to note that, as opposite to the conventional file systems, we cannot take expected rotational delay here. In the conventional system, since the sample size is infinite, the average rotational delay converges to the expected value in probability, according to laws of large number. However, for media servers, the average does not converge since $N_{Limit} << \infty$. Using expected rotational delay could cause violation of the real-time data requirement.

	$\gamma(d)$	$K_{Cushion}$	$K_{MSharing}$	$\gamma(d) \times K_{Cushion} \times K_{MSharing}$
Elevator Policy	*12.5 (ms)*	*2*	*1*	*25*
Stretch	*25 (ms)*	*1*	*1/2*	*12.5*

Table 3: Memory Requirement Factor

We estimate the total memory requirements by inspecting three factors: $\gamma(d)$, $K_{Cushion}$ and $K_{MSharing}$. Since S is directly proportional to seek overhead, $\gamma(d)$ is a good representative for the segment size. Parameter $K_{Cushion}$ represents the cushion buffer requirement of a scheme. As we have discussed, scheme Sweep requires an additional S amount of cushion for each stream, while Stretch requires none. Thus, we assign $K_{Cushion} = 2$ to Sweep, and $K_{Cushion} = 1$ to Stretch. Next, according to Theorem 1, we know Stretch benefits from memory sharing by a factor of $\frac{N_{Limit}+1}{2 \times N_{Limit}}$, we therefore can assign $K_{MSharing} = 1/2$ to Stretch.

For scheme Sweep, the worst total seek overhead in a disk sweep happens when the seek distances are equally spaced out. This is due to the concavity of the seek function [11, 14]. Subsequently, for Sweep, the worst individual seek overhead is $\gamma(CYL/N_{Limit})$. On the other hand, since Stretch suffers from the worst possible seek, its seek overhead is $\gamma(CYL)$. For $N_{Limit} = 40$, a reasonable workload for Barracuda 4LP disks [4], we list the parameter values of $\gamma(d)$, $K_{Cushion}$ and $K_{MSharing}$ for two disk policies in Table 3. It is surprising to see that the cumulative memory requirement factor of Stretch's is only a half of Sweep's.

The evaluation shows that scheme Sweep, althought maximizes disk bandwidth utilization, helps only reducing the first factor of the memory requirement: the size of a segment. To minimize the memory requirements, a media server must also minimize IO time variability and maximize memory sharing, as demonstrated in Stretch.

6 Conclusion

In this paper we have proven that spacing out I/O's properly minimizes the variance of memory requirement for the storage systems. For the systems where all requests have the same display rate, this is equivalent to minimize the memory requirements. This result is notable since the storage system can schedule I/O's to conserve memory. We propose a disk scheme and admission policy both improve system performance by putting the theorems into practice.

References

[1] Seagate barracuda 4lp family product specification. *URL: http://www.seagate.com*, 1996.

[2] R. Bracewell. *The Fourier Transform and Its Applications, Second Edition, Revised.* McGraw-Hill, 186.

[3] A. Campbell and G. Coulson. A qos adaptive transport system. *Proceedings of ACM Multimedia*, pages 117–127, November 1996.

[4] E. Chang and H. Garcia-Molina. Reducing initial latency in multimedia storage systems. *To appear in IEEE Multimedia*, 1997.

[5] T. Chua, J. Li, B. Ooi, and K.-L. Tan. Disk striping strategies for large video-on-demand servers. *Proceedings of ACM Multimedia*, pages 297–306, November 1996.

[6] S. Ghandeharizadeh, S. Kim, and C. Shahabi. On configuring a single disk continuous media server. *SIGMETRICS PERFORMANCE EVALUATION*, 23(1):37–46, May 1995.

[7] D. Makaroff and R. Ng. Schemes for implememting buffer sharing in continuous-media systems. *Information Systems*, 20(6):445–464, 1995.

[8] B. Ozden, A. Biliris, R. Rastogi, and A. Silberschatz. A low-cost storage server for movie on demand databases. *Proc. VLDB*, September 1994.

[9] A. Papoulis. *Probability, Random Variables, and Stochastic Processes, Second Edition*. McGraw-Hill, 1984.

[10] A. Reddy and J. Wyllie. I/o issues in a multimedia system. *Computer*, 2:69–74, March 1994.

[11] C. Ruemmler and J. Wilkes. An intro to disk drive modeling. *Computer*, 2:17–28, March 1994.

[12] R. Steinmetz. Multimedia file systems survey: approaches for continuous media disk scheduling. *Computer Communications*, pages 133–44, March 1995.

[13] R. Strum and D. Kirk. *Signals, Systems and Communication*. John Wiley and Sons, Inc., 1965.

[14] F. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming raid-a disk array management system for video files. *First ACM Conference on Multimedia*, August 1993.

[15] P. Yu, M.-S. Chen, and D. Kandlur. Grouped sweeping scheduling for DASD-based multimedia storage managemen. *Multimedia Systems*, 1(1):99–109, January 1993.

Appendix A: Proof of Theorems

A.1: Proof of Theorem 2

[Proof] The criteria that minimizes $VAR_{p(t)}$ for different display rates can be understood by re-examining Equation 8. However, instead of using the constant segment size S, we first substitute S with S_i to reflect the different display rates, where

$$S_i = DR_i \times T.$$

Re-writing Equation 8 we obtain

$$VAR_{p(t)} = \frac{1}{2\pi^2} \sum_{n=1}^{\infty} (\frac{1}{n^2} \times |\sum_{i=1}^{N} S_i \times e^{-j\tau_i n}|^2) \tag{13}$$

Isolating the term that contains the adjustable parameters $\tau_i's$, we have the optimization objective simplified to be:

$$Minimize \sum_{n=1}^{\infty} \frac{1}{n^2} \times |\sum_{i=1}^{N} S_i \times e^{-j\tau_i n}|^2.$$

Notice that since $S_i = DR_i \times T$ and T is merely a constant, we can reduce the objective function to:

$$Minimize \sum_{n=1}^{\infty} \frac{1}{n^2} \times |\sum_{i=1}^{N} DR_i \times e^{-j\tau_i n}|^2. \tag{14}$$

The different display rate case is different from the constant display rate case in two ways:

1. The first harmonic component may not be eliminated entirely due to different magnitude of the vectors on the complex plane. For example, a vector -1 cannot cancel out a vector 2.

2. Minimizing the variance of $p(t)$ does not necessarily minimize the maximum value of the $p(t)$.

However, minimizing the first harmonic component still minimizes the function's variance due to the dominance of the first harmonic component. In addition, minimizing the function's variance provides an approximation to the optimum in a convenient close form. Therefore we can state Theorem 2 as following.

Since the first harmonic component dominates Expression 14, we can minimize Expression 14 by just minimizing the first harmonic component. Substituting n with 1 in the expression we therefore obtain the theorem. □