Adaptive Cycle Management in Soft Real-time Disk Retrieval *

Youjip Won[1] Il-Hoon Shin[2] Kern Koh[2]

[1]Dept. of Electronics and Computer Engineering
Hanyang University
Seoul Korea zip:133-791
yjwon@ece.hanyang.ac.kr

[2]Computer Science and Engineering Division
Seoul National University
Seoul Korea zip:151-742
{jeje|kernkoh}@oslab.snu.ac.kr

Abstract

The objective of this study is to determine the *right* cycle management policy to service periodic soft real-time disk retrieval. Cycle based disk scheduling provides an effective way of exploiting the disk bandwidth and meeting the soft real-time requirements of individual I/O requests. It is widely used in real-time retrieval of multimedia data blocks. Interestingly, the issue of cycle management with respect to dynamically changing workloads has not been receiving proper attention despite its significant engineering implications on the system behavior. When cycle length remains constant regardless of varying I/O workload intensity, it may cause under-utilization of disk bandwidth capacity or unnecessarily long service startup latency. In this work, we present a novel cycle management policy which dynamically adapts to the varying workload. We develop pre-buffering policy which makes the adaptive cycle management policy robust against starvation. The proposed approach elaborately determines the cycle length and the respective buffer size for *pre-buffering*. Performance study reveals a number of valuable observations. Adaptive cycle length management with incremental pre-buffering exhibits superior performance to the other cycle management polices in startup latency, jitter and buffer requirement. It is found that servicing low playback rate contents such as video contents for 3G cellular network requires rather different treatment in disk subsystem capacity planning and call admission criteria because relatively significant fraction of I/O latency is taken up by plain disk overhead.

Keywords: Streaming, I/O Scheduling, Pre-buffering, Cycle Management, Jitter, Soft real-time

1 Introduction

1.1 Motivation

Recent advances in the speed of microprocessors, communication media, and storage technologies enable the computer to harbor and to transport huge amount of information. Furthermore, advances

*This work was supported by grant No. R08-2003-000-11104-0 from the Basic Research Program of the Korea Science and Engineering Foundation and from Statistical Research Center for Complex Systems at Seoul National University.

in 3G wireless communication services, home networking, WPAN technology accompanied by rapid decrease in hardware form factor have opened up a new era of ubiquitous dissemination and consumption of multimedia contents. Despite all these technical achievements and the initial success of proto-type services, the realization of streaming service is still challenged by a number of technical issues. These mainly stem from the fact that most commodity operating systems and Internet protocols are based upon best effort service paradigm while streaming service mandates data rate guarantee. This fact practically prohibits the cost effective deployment of streaming services.

In this work, we examine the retrieval of data blocks from the disk for real-time streaming service. The subject of multimedia disk scheduling has been rigorously investigated over the past few years. Most of the efforts have been focused on how to schedule a *given* set of requests. In practice, the number of ongoing sessions dynamically changes at the session start, session termination, pause or due to some other reasons. While the majority of the previous works have dealt with cycle based disk scheduling policy, how to manage the cycle under dynamically changing workloads has not been given proper attention despite its significance on system behavior. The length of the cycle is primarily governed by the aggregate playback bandwidth and the cycle length directly affects several important system behaviors, e.g. service startup latency, per stream buffer size, etc. I/O scheduler either can dynamically change the cycle length with respect to workload or can make it constant. Many of the preceding works including a number of prototype streaming servers[29, 25, 3] use fixed length cycle policy. A fixed cycle length policy initializes the length of a cycle sufficiently long and does not change it afterward. The initial cycle length is determined based upon a certain threshold value of disk utilization. This policy may suffer from unnecessary long latency or disk under-utilization.

In this work, we propose *adaptive cycle management*(ACM) policy as a promising way of real-time multimedia data retrieval. *ACM* policy dynamically adjusts the cycle length with respect to the

changes in aggregate playback rate. This enables us to minimize the session startup latency and per session buffer consumption. It provides greater flexibility in exploiting the bandwidth capacity of the disk subsystem. On-line extension of a cycle may entail the temporal insufficiency of data blocks and may cause jitter to some of the ongoing streams. This is because the amount of data blocks which have been read from the disk in a cycle is not sufficient to survive the newly extended cycle. This phenomenon can actually be observed in some commercially available streaming servers. For the smooth extension of a cycle, *ACM* policy loads the data blocks ahead of schedule, which is called *pre-buffering*. A number of different pre-buffering policies are analyzed and evaluated in this work.

1.2 Related Works

Disk scheduling for real-time multimedia playback can be categorized into two: deadline driven and cycle based scheduling. In deadline driven scheduling, each disk I/O is accompanied with the deadline. However, deadline driven scheduling may entail significant head movement overhead. A number of approaches have been proposed to mitigate this overhead by combining it with SCAN algorithm, e.g. SCAN-EDF, Priority SCAN, Feasible Deadline SCAN[5, 1, 20, 4]. Minimizing seek time EDF[12] also mitigates the head movement overhead by sorting requests considering their seek time costs as well as their deadlines. On the other hand, cycle based scheduling algorithm exploits the soft real-time nature of multimedia I/O and can more effectively utilize the disk subsystem capacity[17, 19, 7]. In general operating environment, file system is required to service various types of workload each of which has different latency constraints, e.g. real-time, soft real-time, best effort, and etc. A number of algorithms and file system implementations addressed the issue of handling mixed workloads while maximizing the disk bandwidth utilization[23, 29, 31]. Ghandeharizadeh et al.[8] and Kamel et al.[13] investigated multimedia object retrieval that reduces the overall jitter in multi-priority requests.

Neogi et al.[15] proposed to prebuffer the data blocks when a fraction of a cycle is unused. There

have been a number of proposals for prefetching the leading portion of the requested video file to minimize the startup latency[10, 16]. These works are based upon the best effort approach and does not provide any scheduling nor cycle length allocation for pre-buffering. On the other hand, we elaborately model the length of cycle and the respective buffer size required to prebuffer the data blocks and reflect this information in I/O scheduling. Triantafillou et al.[26] proposed to prebuffer the data into on-disk cache. Using on-disk cache can increase the number of concurrent sessions for a given buffer size. However, in this approach, the cycle length still needs to be extended with the arrival of new sessions and the ongoing sessions will suffer from jitter.

While the zoning technique provides effective utilization of the storage capacity, its varying transfer rate adds another dimension of complexity to the scheduling of data block retrieval. A number of approaches have been proposed to effectively utilize the variable transfer rate of zoned disks for multimedia data retrieval. The simplest approach is to use the average transfer rate of multi-zoned disks[22]. Since this approach is grounded at the stochastic expectation, the actual transfer rate can go below the required transfer rate in a certain cylinder. To overcome this uncertainty, Ghandaharizadeh[9] proposed to place the data blocks to each zone in a round-robin fashion. Meter et al.[14] proposed an analytical model for a multi-zoned disk and performed a physical experiment of the file system performance in the zoned disk. Tse et al.[27] showed that a multi-zoned disk exhibits significant improvement in throughput and proposed an optimal partitioning scheme to achieve the maximum transfer rate. Servicing real-time playback from the disk requires synchronization buffer. If we fetch the entire file into main memory, we do not need buffer for synchronization. Won et al.[30] analyzed the trade-off between these two and proposed an algorithm which minimizes total buffer requirement. In this work, we develop elaborate mechanism to manage the cycle for satisfying soft real-time requirement for multimedia data retrieval while minimizing service startup latency and buffer size requirement.

The rest of this paper is organized as follows. Section 2 describes the issues in multimedia

data retrieval. Section 3 introduces the concept of cycle management and pre-buffering. Section 4 provides rigorous modeling and analysis framework for adaptive cycle management and pre-buffering. In section 5, we present the results of performance evaluation. Section 6 concludes the paper.

2 Multimedia Data Retrieval

2.1 Synopsis: Disk Mechanism

A magnetic disk drive consists of one or more rotating platters on a common spindle. Data is written and is read by magnetic heads, generally one per surface. A track is a concentric circle on one surface. The collection of tracks at the same distance from the center of the platters constitutes a cylinder. Three attributes, <cylinder, track, sector>, uniquely identify the location of data blocks. Time to read (or write) the data blocks from (or to) the disk consists of seek latency (the time needed to move the disk head to the respective track), rotational latency (the time needed to rotate the platter so that the target block arrives underneath the disk head), and data transfer time (the time needed to read/write the data blocks).

Most of the modern disk drives adopt a technique called *Zoning*. Zoning is a technique adopted by hard disk manufacturers to increase the capacity of their disks. In a multi-zoned disk, a number of adjacent cylinders are grouped into a zone and the cylinders in the same zone are formatted with the same number of sectors. Tracks in the outer zone have more sectors. The objective of this technique is to exploit the constant linear bit density such that the outer cylinders have a larger storage capacity than the inner ones. Multi-zoning provides superior storage efficiency. However, the disk exhibits different transfer rate depending on the position of the disk head. Fig. 1 illustrates the transfer rates with respect to the cylindrical distance from the outer most cylinder. The unit in X-axis is 10 MByte. The disk can transfer data at 29 MByte/sec in the outer most track. For the inner

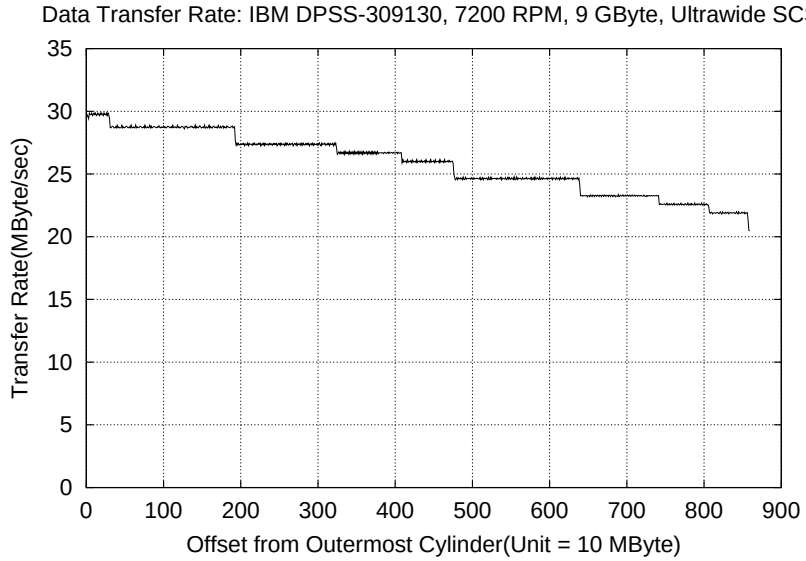


Figure 1: Data Transfer Rate in Zoned Disk: IBM DPSS-309170, 7200RPM, 9 GByte, Interconnection type= Ultrawide SCSI 160

most track, data transfer rate is approximately 24 MByte/sec.

In this work, we assume that the data blocks are placed using the placement strategy proposed in Ghandaharizadeh et al.[9]. Under this placement strategy, the number of data blocks retrieved for a session in a cycle needs to be the integer multiples of the number of zones. Fig. 2 illustrates the placement of the data blocks in a multi-zoned disk. Our modeling framework effectively incorporates the transfer rate variability of zoned disk.

2.2 Call Admission Control and I/O Scheduling

There are two main issues in guaranteeing continuity: admission control and I/O Scheduling. There are a number of different software layers where these functions can be implemented: file system, middleware, and application layers. Fig. 3 illustrates the software architecture of a real-time multimedia application. I/O scheduling and call admission control can be performed in one of the followings: the file system layer, middleware, or application layer(streaming server). When we implement the admission control and I/O scheduling module within file system layer, file system is in charge of scheduling the I/O requests and of regulating the I/O traffic. However, the legacy file

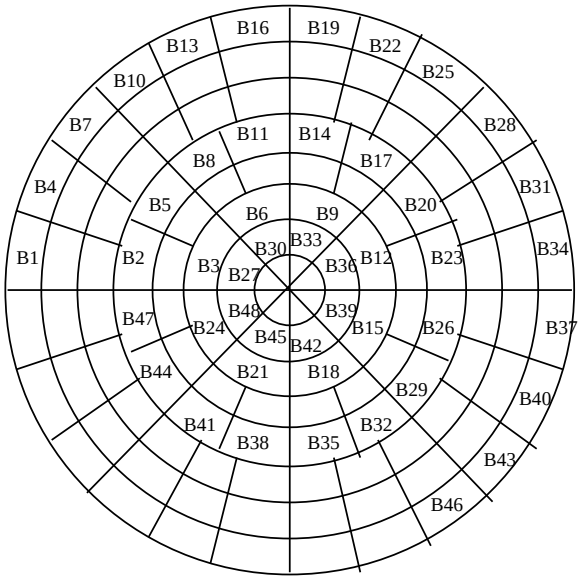


Figure 2: Round Robin Block Placement in Zoned Disk

system, e.g. Unix file system and its clones, are designed to provide only best effort I/O service and these legacy file systems cannot provide I/O rate guarantee to the application. Recently, a number of file systems[25, 29] addressed this issue. These file systems define new programming interfaces so that the application can control the admission of service requests and can specify the timing requirement of the I/O request. The I/O scheduler of these file systems prioritizes the I/O requests to satisfy the timing constraint of a given I/O request. This approach provides greater flexibility in file system usage. The second approach is to use middleware in handling call admission control and I/O scheduling[24]. Middleware takes care of all the I/O requests issued from various applications, e.g. streaming server, ftp server, text editor, or etc. It guarantees continuity by properly regulating the various types of incoming service requests. While this approach facilitates the flexible control over various types of different applications, it does not control the lower level I/O scheduling and thus the system capacity may not be effectively exploited. When we implement the call admission control and I/O scheduling module in application layer, streaming server software is the one to embed these functions in our case. Streaming software is in charge of satisfying the timing requirement of I/O requests. I/O scheduler of a streaming server should have global knowledge on I/O traffic on the

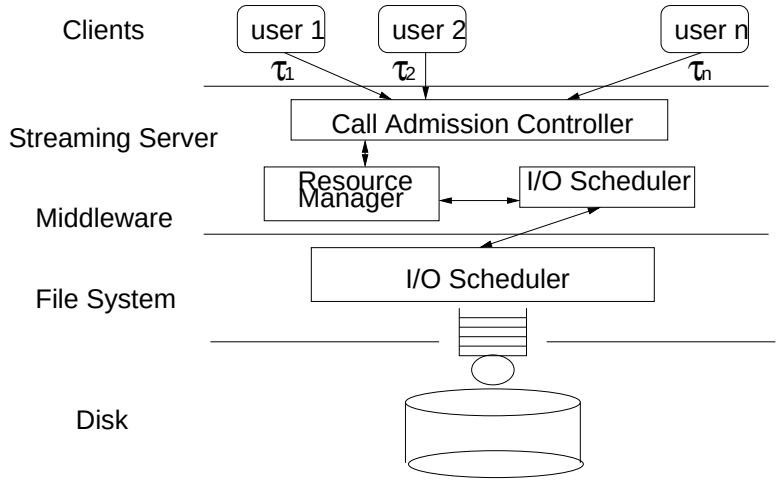


Figure 3: Component Organization of Multimedia Application

given storage subsystem. It practically implies that the given storage subsystem is dedicated for streaming workload. While this approach is one of the most widely used, it leaves much to be desired in effectively exploiting the capacity of a given storage subsystem.

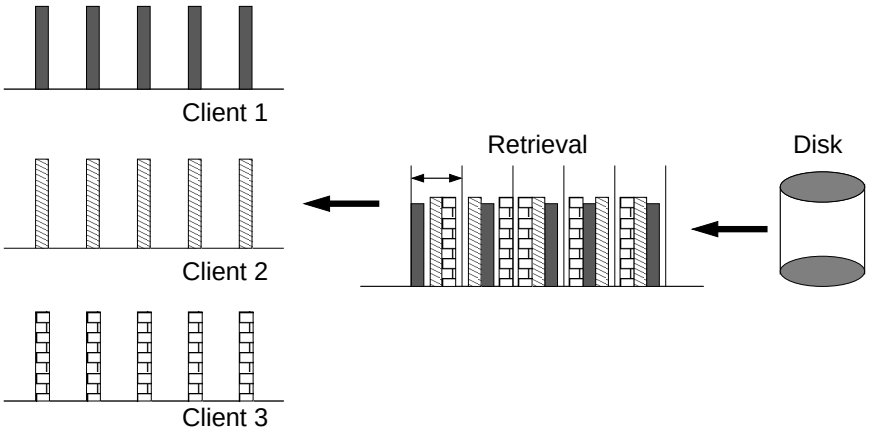


Figure 4: Cycle Based Scheduling and Multimedia Data Retrieval

2.3 Cycle Based Disk Scheduling for Multimedia Data Retrieval

For the efficient support of real-time multimedia playback, one of the most important issues is to schedule a set of soft real-time disk I/O requests to guarantee a certain data rate to the individual sessions. Deadline driven disk scheduling may suffer from excessive disk head movement especially under intense workload[2]. Cycle based disk scheduling effectively addresses this problem

by exploiting the soft real-time nature of multimedia workload. A cycle is the interval between successive bursts of read operations. Fig. 4 illustrates the retrieval of data blocks in cycle based disk scheduling. There are three client sessions. Each of these clients consumes data blocks at a certain data rate. The server retrieves data blocks in each cycle so that it can supply the data blocks to the client, conformant to the respective playback rate. There is an important discrepancy between the playback operation and the disk retrieval operation. Playback of multimedia data is a synchronous operation e.g. 30 frames/sec, but the operation of retrieving the data blocks from the disk is asynchronous. To compromise this discrepancy, a certain amount of the memory buffer are dedicated to individual sessions so that the buffer can absorb the interval variation between successive data block retrievals.

The length of a cycle should be long enough so that the disk can fetch all the required data blocks. The amount of data blocks read in a cycle should be sufficient for cycle length's playback. The continuity guarantee problem is to determine cycle length T and the amount of data blocks to read from the disk for each session. The first condition is that the amount of data blocks read in a cycle should be sufficient for a cycle's playback. This condition can be formulated as in Eq. 1. n, τ_i, r_j and Z denote the number of sessions, max transfer rate of zone i, playback rate of a session j and the number of zones, respectively.

$$T \times \tau_i \leq n_i \times b \tag{1}$$

The second condition is that the disk should be able to read these data blocks, $\sum_{i=1}^{n} n_i$ in less than or equal to T. I/O latency is dependent upon a number of factors: disk head movement overhead, the maximum transfer rate of the disk, etc. The location of data block governs the head movement overhead. In zoned disk, maximum transfer rate varies depending upon the cylindrical position of disk head. All these factors need to be properly incorporated to obtain I/O schedule. Eq. 2 illustrates

this condition.

$$T \geq \sum_{i=1}^{n} \sum_{j=1}^{Z} \frac{n_i \times b}{r_j \times Z} + \sum_{i=1}^{n} T^i_{seek} + \sum_{i=1}^{n} T^i_{lat} + T_{w\_seek} \tag{2}$$

Given all these, we can obtain the length of a cycle as in Eq. 3. $O(n)$ denotes the total head movement overhead to service n streams. Details of this derivation can be found in Appendix A.

$$T \geq \frac{O(n)}{1 - \left(\frac{\sum_{i=1}^{n} \tau_i}{Z} \sum_{j=1}^{Z} \frac{1}{r_j} \right)} \tag{3}$$

3 Adaptive Cycle Management

3.1 Extending a cycle

In practice, the number of concurrent sessions dynamically changes. It may be due to the start of the new session, termination of the ongoing service session, temporal suspension of a playback, or some other reason. As in Eq. 3, the lower bound of the cycle length and the amount of data blocks read in the respective cycle is a function of the aggregate playback rate, $\sum_{i=1}^{n} \tau_i$. When designing streaming server software, it is important to choose the appropriate cycle management policy keeping in mind that workload changes dynamically. The easiest and the most straight forward solution to this problem is to use a constant length cycle. The length of a cycle needs to be sufficiently large to avoid disk under-utilization. Let us call this fixed cycle length policy. In the fixed cycle length policy, the cycle length is set to a certain value, T_{fix}, and the server retrieves $T_{fix} \cdot \tau_i$ amount of data blocks in each cycle for stream i. A request for streaming service is rejected if the cycle length becomes greater than T_{fix} as a result of its admission. As an alternative to the fixed cycle length policy, we can adjust the cycle length with respect to the aggregate playback rate, $\sum_{i=1}^{n} \tau_i$. We call this an adaptive cycle length policy. This policy still has some drawbacks. Though the adaptive cycle length policy can minimize the buffer requirement and service startup latency, it cannot seamlessly adapt to the

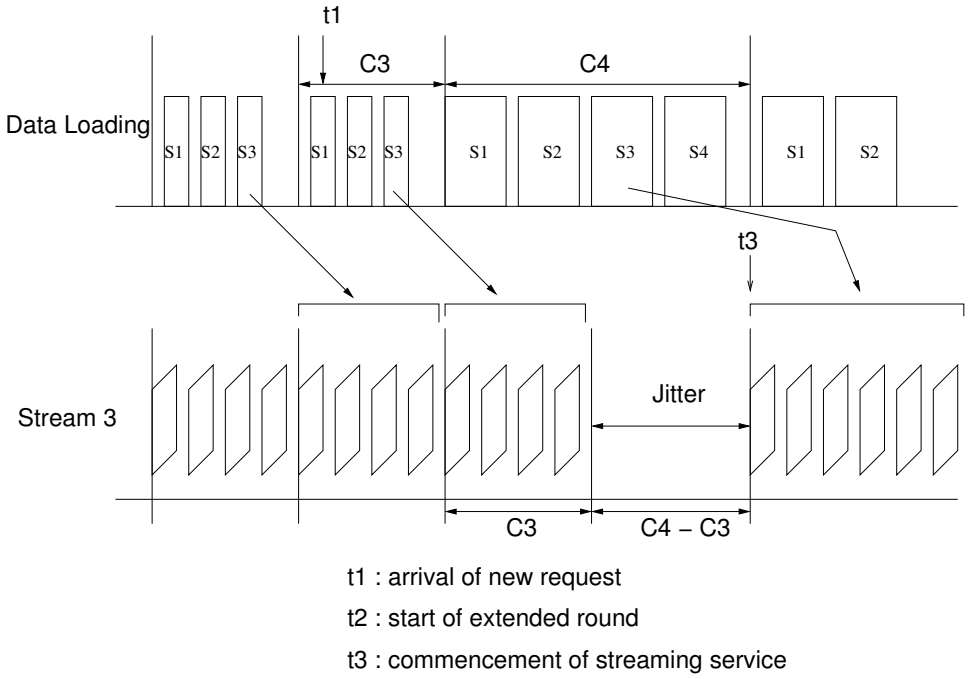


Figure 5: Cycle Extension and jitter

dynamic workload change. Extension of a cycle may entail temporal insufficiency of data blocks and can subsequently cause jitter to ongoing sessions. Fig. 5 illustrates the occurrence of jitters due to the cycle extension. Data blocks retrieved in a cycle become available for playback in the next cycle. The top half of Fig. 5 illustrates that the data block is retrieved from the storage. In the beginning of these are three streams, s_1, s_2 and s_3. The respective cycle length is denoted by C_3. Data blocks for s_1, s_2 and s_3 are being retrieved in round-robin fashion in each cycle. In practice, the order of retrieval is subject to the underlying disk scheduling algorithm. The new service request, s_4, arrives at t_1. When the new request arrives, the resource allocation and call admission module computes the new cycle length and buffer size, and determines whether it is possible to service the newly arriving request. Fig. 5 illustrates that the cycle extension causes jitter to ongoing session. Due to the arrival of new session, cycle length is extended to accommodate a new session. The new cycle length, C_4, becomes effective from the third cycle. The data blocks loaded in the extended cycle, C_4 are available for playback only after t_3. However, the data block retrieved in C_3 in the upper half of the graph is played back in C_3 in the bottom half of Fig. 5 Since the blocks fetched in the third cycle

will be available after t_3, S_3 suffers from to temporal starvation. S_3 experiences jitter for $(C_4 - C_3)$ duration.

3.2 Pre-buffering: Seamless Cycle Extension

A session can survive temporal starvation caused by cycle extension if it has accumulated a sufficient amount of data blocks ahead of schedule. We call the operation of buffering data blocks ahead of schedule pre-buffering. Fig. 6 illustrates the effect of pre-buffering. X and Y axis denote the time and the data blocks in memory. c_i denotes the cycle length for i streams. As data blocks are accumulated, the buffer usage increases. As the player consumes the accumulated data block, the buffer usage decreases. Let n and c_n be the number of sessions and the respective cycle length. In this figure, the session starts at t_0. Prior to start service,the application has accumulated when there are n sessions, the system is in equilibrium state, i.e. consumption and production rates blocks are the same. New request arrives at t_1. As a result, the cycle length is extended to c_{n+1} and it becomes effective from t_2. During the cycle starting from t_2, individual sessions consume the data block loaded in the previous cycle. Data block read in the previous cycle is for c_n's playback and is not sufficient for c_{n+1}'s playback. However, we have accumulated a certain amount of data blocks prior to starting the session. Thus, we can use this data blocks and can survive the cycle extension. The shaded triangular region in Fig. 6 denotes the data blocks supplied from the prebuffered data blocks.

There are two policies in loading the data ahead of schedule. The first approach is to start service only after a sufficient amount of data blocks are loaded into the memory. We call this *Full Prebuffering*. This approach is illustrated in Fig. 6. Disadvantage of this approach is long startup latency. The user waits a number of cycles for the service until a sufficient amount of data blocks becomes available. The second approach is to make the cycle sufficiently large so that the server can retrieve more data blocks than what is to be consumed in a cycle. As playback proceeds, the server incrementally accumulates the data blocks which are used to survive cycle extension,

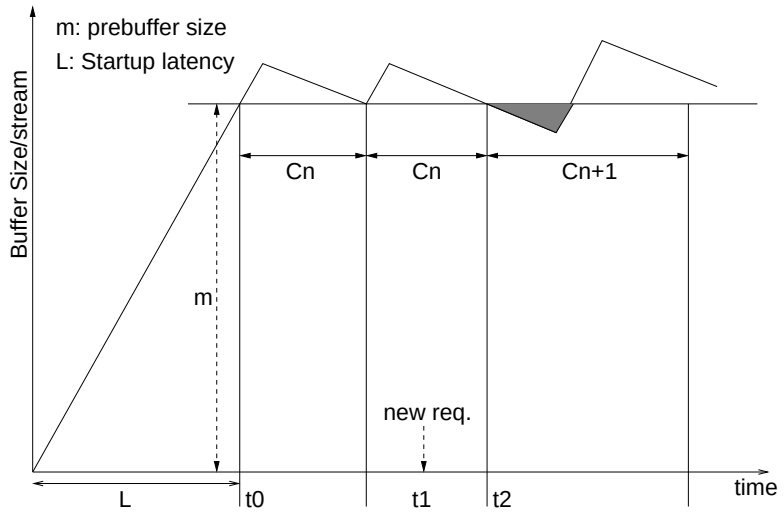


Figure 6: Prebuffering

until a sufficient amount of data blocks are preloaded. We call this approach *Incremental Prebuffering*. Service can start immediately after request arrival and the user experiences relatively shorter startup latency. However, incremental pre-buffering can be more vulnerable to jitter since the cycle may get extended before the server accumulates sufficient amount of data blocks. Section 5 presents an in depth comparison of these two policies under various operating environments.

4 Modeling and Analysis of Prebuffering Policies

4.1 Full pre-buffering

In Full pre-buffering policy, we have to decide how much to prebuffer. For this purpose, we use the notion of *service limit*, ζ, on disk bandwidth utilization. Disk bandwidth utilization is $\frac{aggregate\ bandwidth\ usage}{maximum\ disk\ bandwidth}$. In zoned disk, this can be formally written as $\sum_{i=1}^{n} \tau_i \sum_{j=1}^{Z} \frac{1}{Z \cdot r_j}$. We set up the system so that the total data rate does not exceed the service limit, ζ.[1]. The lower bound on cycle length and the respective buffer size are obtained using Eq. 3. Let T_{max} be the cycle length when the

[1]In practice, streaming service providers are very careful not to overestimate the service limit of disk bandwidth utilization. This conservative approach usually results in significant underutilization of disk capacity. Typically, the service limit is set to less than 10%.

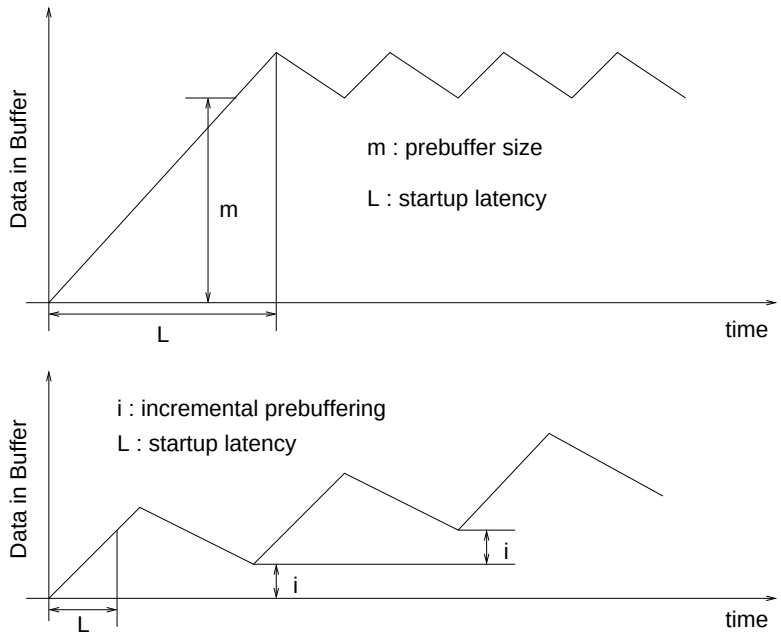


Figure 7: Full pre-buffering vs. Incremental Prebuffering

variable	Meaning
T	cycle length
n_i	number of data blocks read in a cycle for stream i
τ_i	playback rate of stream i
b	block size in byte
r_j	maximum transfer rate of zone j
Z	number of zones
T^i_{seek}	seek time incurred during accessing stream i
T^i_{lat}	rotational latency for stream i
T_{w_seek}	worst case seek
α	fraction of cycle used for reading the data blocks for the next cycle
B^j_{max}	buffer size requirement for session j when the disk utilization is at maximum
B^j_n	buffer size requirement for session j when there are n sessions.
T_{max}	cycle length when the disk utilization is at maximum
T_i	cycle length when there are i streams

Table 1: Summary of Math Notations

disk reaches its service limit. The main idea of full pre-buffering is that the server prebuffers the data blocks for T_{max}'s playback before the start of service. The buffer size for session j, B^j_{max} can be represented as in Eq. 4.

$$B^j_{max} = \left\lceil \frac{T_{max}\tau_i}{b} \right\rceil b \tag{4}$$

We examine the startup latency in full pre-buffering(L in Fig. 6). Full pre-buffering requires that the B^j_{max} amount of data blocks are fetched into memory prior to the start of service. Let T_n and B^j_n be the minimum cycle length and the amount of data blocks retrieved in a cycle for stream j when there are n concurrent streams. Total buffer size, $\hat{B}^j_n$ is the total amount of buffer allocated to session j. It is the sum of the size of prebuffered data blocks and the amount of data retrieved in each cycle, i.e. $\hat{B}^j_n = B^j_{max} + B^j_n$. Let us assume that with the arrival of a new request, the cycle length is extended to T_n. The new stream, say s_j, can start only after B^j_{max} amount of data are fetched into memory. s_j reads B^j_n amount of data blocks in a cycle. L_j, the startup latency for session j, can be formulated as in Eq. 5.

$$\begin{aligned} L_j &= \left\lceil \frac{B^j_{max}}{B^j_n} \right\rceil T_n \\ &= \left\lceil \lceil \frac{T_{max}\tau_n}{b} \rceil b \frac{1}{B^j_n} \right\rceil T_n \end{aligned} \tag{5}$$

4.2 Incremental pre-buffering

Full buffering may entail non trivial service startup latency. To reduce startup latency, we propose a policy called *Incremental Prebuffering*. When the cycle is longer than its minimum requirement, we can retrieve more data blocks than what is to be consumed during a single cycle. In incremental pre-buffering, we determine the cycle length so that the application retrieves more data than what is to be consumed(either displayed or transmitted) in a cycle. The idea of incremental pre-buffering is to use these surplus data blocks to survive the cycle extension. In incremental pre-buffering, the

server does not wait until the B_{max} amount of data blocks becomes available in memory. Instead, it immediately starts playback. It accumulates the data blocks until the B_{max} amount of data blocks becomes available in memory.

The *consumption ratio*, α ($0 < \alpha \leq 1$) is the ratio between data retrieved and consumed in a cycle. αn_i denotes the amount of data blocks played in a cycle while n_i denotes the amount of data blocks read in a cycle. In incremental pre-buffering, the length of a cycle and the respective data blocks to be read in a cycle should be large so that α fraction of the data blocks suffices for a cycle's playback. The continuity requirement in incremental pre-buffering can be written as in Eq. 6.

$$T\tau_i \leq \alpha n_i b \tag{6}$$

Solving Eq. 6 and Eq. 2, we obtain the number of data blocks read in a cycle, n_i, as in Eq. 7. n_i is upper bounded by B^i_{max}. Details of the derivation can be found in Appendix B.

$$n_i \geq \frac{O(n)\tau_i}{\alpha - \left(\frac{\sum_{1=1}^{n}\tau_i}{Z}\sum_{j=1}^{Z}\frac{1}{r_j}\right)}\frac{1}{b} \quad and \quad n_i \leq B^i_{max} \tag{7}$$

$(1-\alpha)n_i$ data blocks are accumulated for each cycle until the B^i_{max} amount of data blocks becomes available in memory. In incremental pre-buffering, a session starts after the single cycle's data blocks are fetched into memory. Eq. 8 illustrates the cycle length in incremental pre-buffering.

$$L = T \geq \frac{O(n)}{\alpha - \left(\frac{\sum_{1=1}^{n}\tau_i}{Z}\sum_{j=1}^{Z}\frac{1}{r_j}\right)} \tag{8}$$

4.3 Array of Disks and Adaptive Cycle Extension

Till now, our modeling effort is based upon the single disk assumption. In practice, it is not uncommon that the streaming server uses disk array as its storage subsystem. Our modeling on

buffer size requirement and cycle extension can easily be extended to the realm of RAID storage via slight modification. There are a number of ways to place a set of files over these disks. The simplest approach is localized placement where a file is placed on a single disk. This placement policy can cause hot-spot problems. By striping the file over all disks, we can evenly distribute the load among the disks. The striping unit can be a block(coarse grain striping) or a byte(fine grain striping)[6, 18]. Replicating a file is also a resort to increase the access bandwidth for a file[28].

Eq. 9 and Eq. 10 state the minimum cycle length requirement in fine grain striping and coarse grain striping, respectively. By replacing Eq. 2 with one of these formulas, we can easily derive the buffer size, cycle length for various pre-buffering policies for disk arrays with coarse grain striping and fine grain striping, respectively. Let us re-define n_i as the number of the data blocks read from the disk array in a cycle. Let D be the number of disks in the disk array. In fine grain striping[21], individual disks reads $\frac{n_i}{D}$ amount of data and thus Eq. 2 is modified into Eq. 9.

$$T_{fgs} \geq \sum_{i=1}^{n} \sum_{j=1}^{Z} \frac{(n_i/D) \times b}{r_j \times Z} + \sum_{i=1}^{n} T_{seek}^{i} + \sum_{i=1}^{n} T_{lat}^{i} + T_{w\_seek} \tag{9}$$

The amount of data blocks to be read from the disk in coarse grain striping is the same as in fine grain striping. However, since the data blocks are striped in a block-wise fashion, data blocks retrieved from a single disk are for $\frac{n}{D}$ sessions, where n is the number of sessions. Disk head movement overhead is governed by the number of sessions to serve. Thus, coarse grain striping entails less disk overhead, i.e. seek and rotational latency. The continuity requirement of coarse grain striping can be formulated as in Eq. 10.

$$T_{cgs} \geq \sum_{i=1}^{\frac{n}{D}} \sum_{j=1}^{Z} \frac{(n_i) \times b}{r_j \times Z} + \sum_{i=1}^{\frac{n}{D}} T_{seek}^{i} + \sum_{i=1}^{\frac{n}{D}} T_{lat}^{i} + T_{w\_seek} \tag{10}$$

Comparing Eq. 9 and Eq. 10, we can observe that disk overhead, i.e. seek time and rotational

latency, is smaller in coarse grain striping, which implies shorter cycle length. On the other hand, startup latency of coarse grain striping can be as long as $(D + 1)T_{cgs}$, but the startup latency of fine grain striping is T_{fgs}. Further detailed analysis of multimedia data retrieval from array of disks should be dealt with in separate context. The interested readers are referred to[2, 21].

5 Performance Analysis

In this section, we analyze the performance and the effectiveness of the pre-buffering strategy. We examine jitter, startup latency, and buffer requirement under four different cycle management policies: (i) *fixed cycle length* policy where the length of the cycle is set sufficiently large and does not change with workload, (ii) *adaptive cycle length without pre-buffering* where cycle length is adjusted with respect to the aggregate playback bandwidth, (iii) *adaptive cycle length with full pre-buffering* and (iv) *adaptive cycle length with incremental pre-buffering*. We consider three playback rates in our simulation study: 64 Kbits/sec, 1.5 Mbits/sec and 19.2 Mbits/sec. A playback rate of 64 Kbits/sec is used to examine the streaming server behavior for a mobile wireless streaming service. A playback rate of 1.5 Mbits/sec and 19.2 Mbits/sec are used to simulate the server for MPEG-1 and HDTV (ATSC compressed) streaming services, respectively. Playback length is 100 minutes.

The disk is modeled after the IBM Deskstar 180 GXP with a storage capacity of 180 GByte. The disk platter is partitioned into 27 zones. The sector size is 512 bytes. The transfer rate of each zone ranges from 29 MB/s to 56 MB/s. The disk rotates at 7200 RPM and the average and full seek times are 8.5 msec and 15 msec, respectively. We assume that there are 100 video titles in the server for 64 Kbits/sec and 1.5 Mbits/sec/sec, and 10 video titles for 19.2 Mbits/sec[2]. 19.2 Mbits/sec is playback rate for ATSC digital TV[11]. Data blocks in a file are placed in consecutive locations on the disk platter. We assume CBR playback. In practice, user accesses are concentrated on a small number of

[2]180 GByte can store about 10 files of 100 minutes in length encoded with 19.2 Mbits/sec

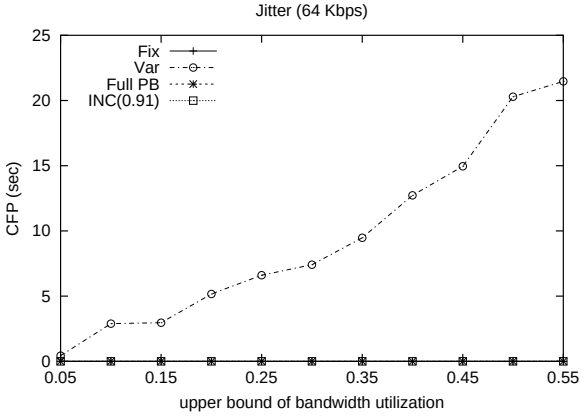
"hot" files while the rest of the video titles are rarely accessed. To model this situation, we use Zipf distribution with a parameter of 0.271. The request arrival process is modeled using the Poisson process with an average inter-arrival time of 3 sec. The block size is 4 KB and the simulation runs for 6 simulated hours.

5.1 Jitter

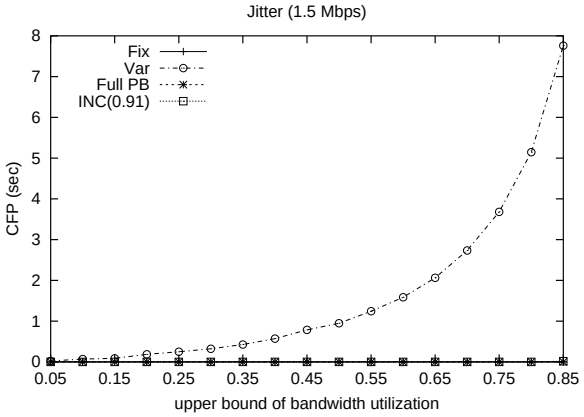
Cycle extension delays timely retrieval of data blocks. From an application's point of view, this delay can potentially prohibit the advancement of a scene. As illustrated in Fig. 5, the amount of delay corresponds to the difference between the lengths of successive cycles before and after extension. We call this delay *frozen period*. The effectiveness of a cycle management policy relies on how to insulate the application from screen freeze phenomenon. The length of total frozen period for a playback is called the *Cumulative Frozen Period (CFP)*.

Fig. 8(a), 8(b) and Fig. 8(c) illustrate the average length of the Cumulative Frozen Period (*CFP*). The *X* and *Y* axis denote the service limit and *CFP*, respectively. *Fix*, *Var*, *Full PB* and *Inc* correspond to fixed cycle length, adaptive cycle length, adaptive cycle length with full pre-buffering and adaptive cycle length with incremental pre-buffering, respectively. In incremental pre-buffering, we assume that 9% of the cycle is used for prebuffering, where $\alpha = 0.91$. With 64 Kbits/sec playback rate streams (Fig. 8(a)), disk utilization rarely goes beyond 55% with a 3 sec average inter-arrival time. We vary the service limit from 0.05 to 0.55. In all bandwidth utilization settings, *Fix*, *Full PB* and *INC* do not cause any jitter. However, dynamic extension of a cycle *without* pre-buffering causes jitter to individual playback sessions. The total jitter increases up to 20 sec depending on the service limit.

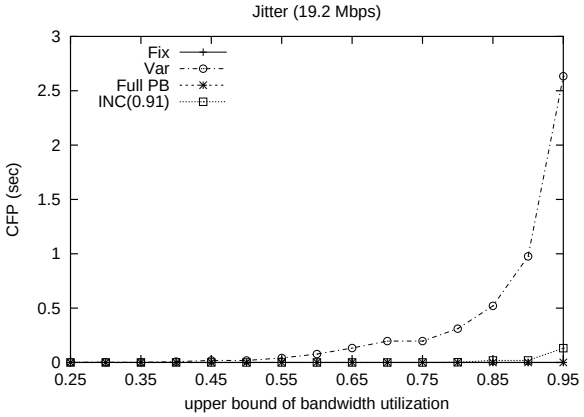
For 1.5 Mbits/sec playback, jitters are found only in the *VAR* policy(Fig. 8(b)). We observe that the total length of jitter increases very fast as disk utilization increases. This is because when the disk bandwidth utilization is low, the respective cycle length is relatively short. Thus, even though cycle extension occurs, the difference in cycle lengths before and after the cycle extension is relatively



(a) Playback rate = 64 Kbits/sec



(b) Playback rate = 1.5 Mbits/sec



(c) Playback rate = 19.2 Mbits/sec

Figure 8: Cumulative Frozen Period under different playback rates. Request arrival is modeled after Poisson arrival with inter-arrival time, 3 sec. *Fix*: fixed cycle length, *Var*: adaptive cycle length, *Full PB*: adaptive cycle length with full pre-buffering, *Inc*: adaptive cycle length with incremental pre-buffering

insignificant. However, as disk bandwidth utilization increases, the cycle length increases rapidly with respect to the disk bandwidth utilization. Subsequently, the differences in cycle lengths before and after the cycle extension becomes more significant.

We find an interesting system behavior in this test. Let us examine the total length of frozen period between the 1.5 Mbits/sec(Fig. 8(b)) and 64Kbits/sec(Fig. 8(a)) playback. Consider 55% disk bandwidth utilization. When the playback rate is 1.5 Mbits/sec, the user experiences total frozen period of 1.5 sec during 120 minutes of playback. Meanwhile, the user suffers from total frozen period of 20 sec when the playback rate is 64 KBits/sec. We observe that the lower rate playbacks are much more vulnerable to cycle extension. When the playback rates of individual sessions are low, the disk subsystem can support relatively larger number of streams. This causes larger disk head movement overhead, i.e. larger $O(n)$ in Eq. 3, and subsequently makes the cycle length significantly longer. This phenomenon bears an important implication on capacity planning of the streaming server. When the playback rate is relatively low, as in the mobile wireless streaming service, the target disk utilization of the storage subsystem should be much lower than the target utilization of streaming server for the high speed networked environments or for high definition TV service.

The advantage of using pre-buffering becomes more obvious when the playback rate is relatively low. Let us consider the 19.2 Mbits/sec playback rate. Even when the server is heavily loaded, e.g. 90% disk utilization, the cumulative frozen period for each session is only 1 sec on average with *VAR* policy (adaptive change without pre-buffering). In fact, cycle length does not increase beyond 3 sec when the playback rate is 19.2 Mbits/sec. Since the cycle length is much shorter with higher playback rate, the jitter caused by cycle extension tends to be less severe. This is because with a 19.2 Mbits/sec playback rate, disk bandwidth is saturated with a small number of sessions and thus disk head movement overhead is relatively low.

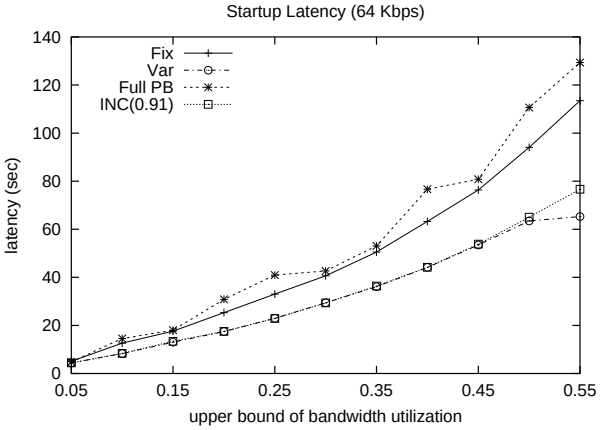
We can summarize the findings as follows. When the playback rate is either 1.5 Mbits/sec or 64 Kbits/sec, we significantly reduce the length of cumulative frozen period by pre-buffering the data

blocks. Prebuffering the data blocks effectively removes the jitter. The improvement becomes more significant as the server becomes more heavily loaded. Instead of adaptively extending the cycle, using sufficiently long cycle is also a way to achieve jitter free playback. Incremental pre-buffering exhibits almost as good performance as full pre-buffering or the fixed cycle length policy from the aspect of cumulative frozen period.

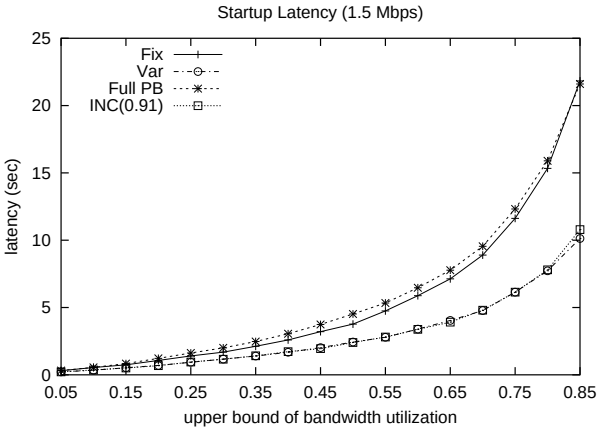
5.2 Start-Up Latency

Jitter and startup latency are two important metrics for evaluating the cycle management policy. In section 5.1, we found that the cycle management policies with pre-buffering produce far superior jitter behavior than policies without pre-buffering. However, the startup latency overhead of pre-buffering may offset the advantage of pre-buffering especially in on-demand service environment. In this section, we examine the startup latency of individual policies. Fig. 9(a), 9(b) and Fig. 9(c) illustrate the average service startup latency for each playback rate. The X and Y axis denote the service limit and the elapsed time between the arrival of a request and the start of the actual service, respectively. The adaptive cycle length policy without pre-buffering(*VAR*) maintains the cycle length as short as possible. It results in the shortest startup latency. However, this policy suffers from the worst jitter behavior as shown in section 5.1. On the other hand, the startup latency seen with an incremental pre-buffering policy with a consumption factor of 0.91 is almost as good as the startup latency of the *VAR* policy at all playback rates.

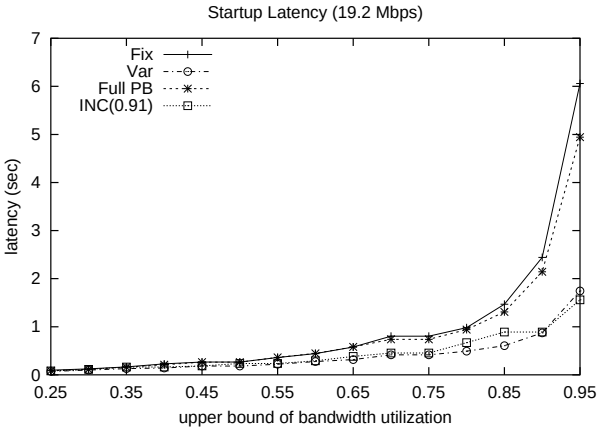
For 19.2 Mbits/sec playback rate (Fig. 9(c)), startup latency of full pre-buffering ranges from 0 to 5 sec depending on service limit. In incremental pre-buffering policy, startup latency is less than 2 sec. Since the server starts dispatching the data blocks right after the first cycle, the startup latency in incremental pre-buffering is smaller than the startup latency of the full pre-buffering and fixed cycle length policies. Full pre-buffering policy and fixed cycle length policy have relatively long startup latency. In practice, multimedia client application has several seconds' startup latency which



(a) Playback rate = 64 Kbits/sec



(b) Playback rate = 1.5 Mbits/sec



(c) Playback rate = 19.2 Mbits/sec

Figure 9: Startup latency under different playback rates. Request arrival is modeled after Poisson arrival with inter-arrival time, 3 sec. *Fix*: fixed cycle length, *Var*: adaptive cycle length, *Full PB*: adaptive cycle length with full pre-buffering, *Inc*: adaptive cycle length with incremental pre-buffering

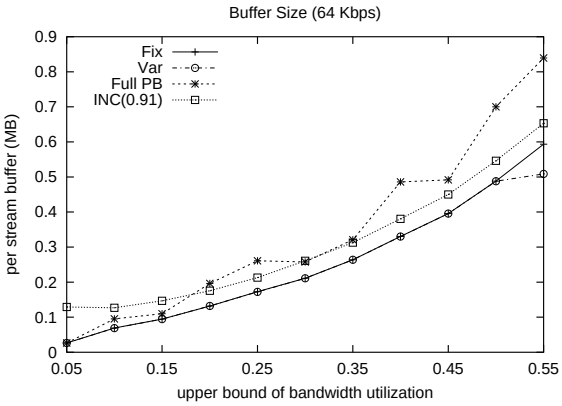
is primarily for pre-buffering the streaming data. Thus, startup latency of 19.2 Mbits/sec stream can be considered to be insignificant. As service limit increases, startup latency rapidly increases in all cycle management policies. We can significantly improve startup latency by properly adjusting the service limit of the disk. For example, in the incremental pre-buffering policy with 1.5 Mbits/sec playback, average startup latency becomes less than 3 sec by making the service limit 60%.

5.3 Buffer Size Requirement

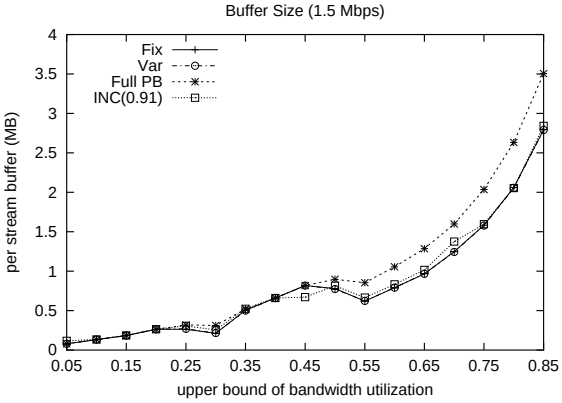
Buffer size is subject to cycle length and pre-buffering policy. Fig. 10(a), 10(b) and 10(c) illustrate the results of our experiment. The X and Y axis denote the service limit and per stream buffer requirement, respectively. An adaptive cycle length without pre-buffering exhibits the smallest buffer size requirement because it keeps the cycle length as short as possible and does not prebuffer data blocks. When the disk subsystem is utilized to its service limit, the buffer size are approximately the same in four cycle management policies.

For 1.5 Mbits/sec and 19.2 Mbits/sec playback rates, the simulation results show that the buffer size requirement is approximately the same for all cycle management policies(Fig. 10(b) and Fig. 10(c)). With the 19.2 Mbits/sec playback rate, a small number of concurrent sessions can saturate the server. Since the disk is fully utilized to its service limit in most of the time in this case, the advantage of adaptive cycle management can be marginal.

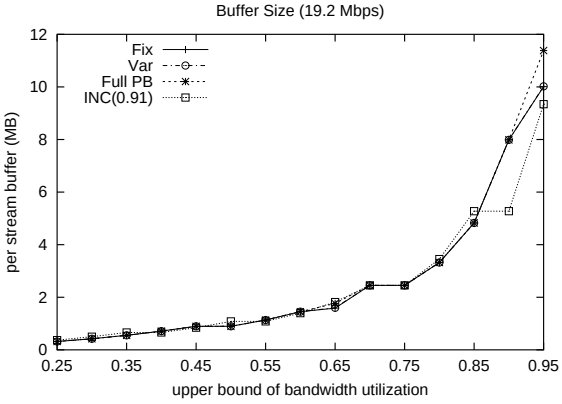
It is worth noting that as service limit gets closer to 100%, the per stream buffer size increases rapidly in all the scheduling policies. For example, a single 1.5 Mbits/sec stream with incremental pre-buffering requires 2.8 MByte of buffer space when the service limit is 0.85. However, when the service limit is 0.60, the per session buffer size decreases to 0.8 MByte. We can significantly reduce the buffer size requirement by carefully controlling the service limit.



(a) Playback Rate = 64 Kbits/sec



(b) Playback Rate = 1.5 Mbits/sec



(c) Playback Rate = 19.2 Mbits/sec

Figure 10: Buffer Size Requirements under different playback rates. *Fix*: fixed cycle length, *Var*: adaptive cycle length, *Full PB*: adaptive cycle length with full pre-buffering, *Inc*: adaptive cycle length with incremental pre-buffering

6 Conclusion

Cycle-based disk scheduling policy has been widely used to retrieve multimedia data blocks for real-time playback. Determining the length of the cycle for a given workload and setting up the respective data retrieval schedule have been subjects of rigorous studies in recent years. Interestingly however, the issue of how to manage the cycle length with respect to dynamically changing workloads has not been receiving proper attention despite its significant engineering implications. The objective of this study is to determine the *right* cycle management policy for periodic soft real-time disk request and eventually to help software engineers or system developers in designing more efficient streaming systems.

Cycle management policies can be categorized into two types: *fixed cycle length* and *adaptive cycle length*. Adaptive cycle length policy effectively utilizes buffer memory and minimizes startup latency. However ongoing streams may get exposed to jitter when the cycle is extended due to the commencement of new streaming sessions. On the other hand, fixed cycle length policy may cause long startup latency and buffer overhead. To resolve the temporal insufficiency of data blocks in adaptive cycle length management policies, we propose a technique called *pre-buffering* which is to make a sufficient amount of data blocks available in memory prior to starting service. We examine two ways of pre-buffering: *full pre-buffering* and *incremental pre-buffering*. We develop an elaborate model to compute the cycle length and the respective buffer size in both *full pre-buffering* and *incremental pre-buffering*.

Through simulation-based experiments, we examine three aspects of the system behavior: the *total jitter*, *startup latency*, and *buffer size* under different policies: *fixed cycle length*, *adaptive cycle length with full pre-buffering*, *adaptive cycle length with incremental pre-buffering* and *adaptive cycle length without pre-buffering* policies. Prebuffering enabled policies deliver jitter-free playback. Adjusting the cycle length without pre-buffering causes non-negligible amount of jitter. *Full pre-buffering* and

fixed cycle length policies entail prohibitively long startup latency, e.g. approximately 120 sec in 64 Kbits/sec playback rate streams. Startup latency in the *incremental pre-buffering* policy is almost as good as the the cycle management policy which does not prebuffer data blocks. We find that pre-buffering overhead in incremental pre-buffering is almost negligible, especially under higher playback rates(1.5 Mbits/sec or higher). We also find that we can significantly reduce startup latency and the buffer size requirement by properly adjusting the service limit. It is found that servicing low playback rate contents such as video contents for 3G cellular network requires rather different treatment in disk subsystem capacity planning and call admission criteria because relatively significant fraction of I/O latency is taken up by plain disk overhead. Analyzing the results of our study, we conclude that the *adaptive cycle length management with incremental pre-buffering* is the promising way of managing the cycle in soft real-time disk I/O. In this work, we develop elaborate mechanism to manage the cycle for satisfying soft real-time requirement for multimedia data retrieval while minimizing service startup latency and buffer size requirement. The result of our work provides practical guideline for server capacity planning and setting up resource allocation strategy in providing multimedia streaming service.

References

[1] R K. Abbott and H. Garcia-Molina. Scheduling i/o requests with deadlines: A performance evaluation. In *Proceedings of RTSS*, pages 113–124, December 1990.

[2] Antine Mourad. Issues in the design of a storage server for video-on-demand. *Multimedia Systems*, 1996(4):70–86, 1996.

[3] W. J. Bolosky, R. P. Fitzgerald, and J. R. Douceur. Distributed schedule management in the tiger video fileserver. In *ACM SIGOPS Operating Systems Review(ACM)*, volume 31, pages 212–223, 1997.

[4] M J. Carey, R. Jauhari, and M. Linvy. Priority in dbms resource scheduling. In *Proc. of VLDB*, pages 397–410, 1989.

[5] S. Chen, J. A. Stankovic, J. F. Kurose, and D. Towsley. Performance evluation of two new disk scheduling algorithms for real-time systems. *Journal of Real-Time Systems*, 3:307–336, 1991.

[6] D. Gemmell, H. Vin, D. Kandlur, P. Rangan, and L. Rowe. Multimedia Storage Servers: A Tutorial. *COMPUTER*, 28(5):40–49, May 1995.

[7] J. Gemmell. Multimedia Network File Servers:Multi-Channel Delay Sensitive Data Retrieval. In *Proceedings of ACM Multimedia Conference*, pages 243–250. ACM, Oct. 1993.

[8] S. Ghandeharizadeh, L. Huang, and I. Kamel. A cost driven disk scheduling algorithm for multimedia object retrieval. *IEEE Transactions on Multimedia*, 5(2):186–196, 2003.

[9] Ghandeharizadeh, Shahram and Kim, S. and Shahabi, C. Continuous Display of Video Objects Using Multi-Zoned Disks. Technical report, University of Southern California, 1995.

[10] Sreenivas Gollapudi and Aidong Zhang. Buffer management in multimedia database systems. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 186–190, Hiroshima, Japan, June 1996.

[11] http://www.atsc.org. Atsc standard.

[12] Y. Huang and J. Huang. Disk scheduling on multimedia storage servers. *IEEE Transactions on Computers*, 53(1):77–82, 2004.

[13] I. Kamel, T. Niranjan, and S. Ghandeharizadeh. A novel deadline driven disk scheduling algorithm for multi-priority multimedia objects. In *Proceedings of IEEE Data Engineering Conference*, 2000.

[14] R. V. Meter. Observing the effects of multi-zoned disks. In *Proc. of Usenix Technical Conference*, San Jose, CA, USA, 1997.

[15] Anindya Neogi, Ashish Raniwala, and Tzi cker Chiueh. Phoenix: A lower-power fault-tolerant real-time network-attached storage device. In *Proceedings of ACM Multimedia Conference*, pages 447–456, Orlando, FL, USA, October 1999.

[16] Raymond T. Ng and Jinhai Yang. An analysis of buffer sharing and prefetching techniques for multimedia systems. Technical Report 20, University of British Columbia, Vancouver, B.C., V6T1Z4, Canada, 1994.

[17] B Ozden, A. Biliris, R. Rastogi, and Avi Silberschatz. A Low-Cost Storage Server for Movie on Demand Databases. In *Proc. of VLDB*, 1994.

[18] Banu Ozden, Rajeev Rastogi, and Avi Silberschatz. Disk Striping in Video Server Environments. In *Proceedings of the International conference on Multimedia Computing and Systems*, Hiroshima, Japan, May 1996.

[19] P. Rangan, H. Vin, and S. Ramanathan. Designing an on-demand multimedia service. *IEEE Communication Magazine*, 30(7):56–65, July 1992.

[20] A. Reddy and J. Wyllie. Disk scheduling in a multimedia i/o system, 1993.

[21] Martin Reisslein, Keit W. Ross, and Subin Shrestha. Striping for Interactive Video:Is it Worth it? In *Proceedings of International Conference on Multimedia Computing and Systems*, Florence, Italy, 1999.

[22] Renu Tewari and Richard King and Dilip Kandlur and Daniel M. Dias. Placement of Multimedia Blocks on Zoned Disks. In *Proceedings of SPIE West '96*, 1996.

[23] Y. Rompogiannakis, G. Nerjes, P. Muth, M. Paterakis, P. Triantafillou, and G. Weikum. Disk scheduling for mixed-media workloads in a multimedia server. In *Proceedings of ACM Multimedia Conference*, pages 297–302, Bristol, UK, 1998.

[24] Prashant Shenoy, Saif Hasan, Purushottam Kulkarni, and Krithi Ramamritham. Middleware versus native os support: Architectural considerations for supporting multimedia applications. In *Proceedings of IEEE Real-time Technology and Applications Symposium*, San Jose, CA, USA, Sep 2002.

[25] Prashant J. Shenoy and Harrick M. Vin. Cello: Disk scheduling framework for next generation operating system. In *Proceedings of ACM SIGMETRICS*, pages 44–55, Madison, WI, USA, 1998.

[26] P. Triantafillou and S. Harizopoulos. Prefetching into smart-disk caches for high performance media servers. In *Proceedings of IEEE International COnference on Multimedia Computing and Systems*, TrianICMCS99.

[27] P.K.C. Tse and C.H.C. Leung. Improving multimedia systems performance using constant-density recording disks. *Multimedia Systems*, 8(1):47–56, January 2000.

[28] Y. Wan and D. Du. Video File Allocation over Disk Arrays for Video-On-Demand. In *Proceedings of International Conference for Multimedia Computing and Systems*, 1996.

[29] Ravi Wijayaratne and A.L. Narasimha Reddy. System support for providing integrated services from networked multimedia storage servers. In *Proceedings of ACM Multimedia Conference*, pages 270 – 279, Ottawa, Canada, November 2001.

[30] Youjip Won and Jaideep Srivastava. "smdp: Minimizing buffer requirements for continuous media servers". *ACM Multimedia Systems Journal*, 8(2):105–117, March 2000.

[31] Youjip.Won and Y.S.Ryu. Handling sporadic tasks in multimedia file system. In *Proceedings of ACM Multimedia Conference*, pages 462–464, Los Angelses, CA, USA, 2000.

A Scheduling Constraints

We formulate the general constraints in disk scheduling for continuous media playback. Let s = $\{s_1,...,s_n\}$ be a set of n streams, and let τ_i be the playback rate for stream s_i. n_i and b are the number of disk blocks to be fetched for s_i in a cycle and the size of a block, respectively. Two conditions of continuity guarantee can be formally described as in Eq. 11 and Eq. 12. Eq. 11 illustrates the condition that the number of blocks read in a cycle should be sufficient for playback of length T.

T^i_{seek}, T^i_{lat}, T_{w_seek}, Z, and r_j denotes the average seek time for stream s_i, rotational latency, worst case seek time, the number of zones in the disk, and the data transfer rate of zone j, respectively.

The first condition is that the amount of data blocks read in a cycle should be sufficient for a cycle's playback. This condition can be formulated as in Eq. 11.

$$T \times \tau_i \leq n_i \times b \tag{11}$$

We need to formulate the condition that the time to read the data blocks for all ongoing sessions for a cycle's playback should be less than the cycle length itself.

$$T \geq \sum_{i=1}^{n} \sum_{j=1}^{Z} \frac{n_i \times b}{r_j \times Z} + \sum_{i=1}^{n} T_{seek}^{i} + \sum_{i=1}^{n} T_{lat}^{i} + T_{w\_seek} \tag{12}$$

With Eq. 11 and Eq. 12, we can obtain the length of cycle as in Eq. 13. $O(n)$ in Eq. 13 corresponds to the total disk head movement overhead, i.e., $O(n) = \sum_{i=1}^{n} T_{seek}^{i} + \sum_{i=1}^{n} T_{lat}^{i} + T_{w\_seek}$ in retrieving the data blocks for n streams. From the condition $T \cdot \tau_i \leq n_i b$ and Eq. 13, we can obtain the number of data blocks to read in a cycle.

$$T \geq \frac{O(n)}{1 - \left(\frac{\sum_{i=1}^{n} \tau_i}{Z} \sum_{j=1}^{Z} \frac{1}{r_j} \right)} \tag{13}$$

B Scheduling Constraints with Slack

We formulate the scheduling constraints given that only α fraction of the data blocks read in a cycle will be used for playback in the next cycle. Then, these constraints can be formulated as in Eq. 14 and Eq. 15.

$$T \times \tau_i \leq \alpha n_i \times b \tag{14}$$

$$T \geq \sum_{i=1}^{n} \sum_{j=1}^{Z} \frac{n_i \times b}{r_j \times Z} + \sum_{i=1}^{n} T_{seek}^{i} + \sum_{i=1}^{n} T_{lat}^{i} + T_{w\_seek} \tag{15}$$

Two conditions in Eq. 14 and Eq. 15 can be solved as follows.

$$
\begin{aligned}
\frac{\alpha n_i \times b}{\tau_i} &\geq \sum_{i=1}^{n}\sum_{j=1}^{Z}\frac{n_i \times b}{r_j \times Z} + \sum_{i=1}^{n} T^i_{seek} + \sum_{i=1}^{n} T^i_{lat} + T_{w\_seek} \\
\frac{\alpha n_i \times b}{\tau_i} &\geq \sum_{i=1}^{n}\sum_{j=1}^{Z}\frac{n_i \times b}{r_j \times Z} + O(n) \\
\mathbf{n}\tau^{-1} \times b &\geq b\mathbf{n}\mathbf{I}^{-1}\sum_{j=1}^{Z}\frac{1}{r_j \times Z} + O(n) \\
\mathbf{n}\tau^{-1} \times b - b\mathbf{n}\mathbf{I}^{-1}\sum_{j=1}^{Z}\frac{1}{r_j \times Z} &\geq O(n) \\
b\mathbf{n}\left(\tau^{-1} - \mathbf{I}^{-1}\sum_{j=1}^{Z}\frac{1}{r_j \times Z}\right) &\geq O(n) \\
b\mathbf{n} &\geq \frac{O(n)}{\left(\tau^{-1} - \mathbf{I}^{-1}\sum_{j=1}^{Z}\frac{1}{r_j \times Z}\right)} \\
\mathbf{n} &\geq \frac{O(n)\tau}{b\left(\alpha - \tau\mathbf{I}^{-1}\sum_{j=1}^{Z}\frac{1}{r_j \times Z}\right)} \\
\mathbf{n} &\geq \frac{O(n)\tau}{b\left(\alpha - \tau\mathbf{I}^{-1}\sum_{j=1}^{Z}\frac{1}{r_j \times Z}\right)}
\end{aligned}
$$

The amount of data blocks read for session i can be computed as in Eq. 16.

$$
n_i \geq \frac{O(n)\tau_i}{b\left(\alpha - \tau\mathbf{I}^{-1}\sum_{j=1}^{Z}\frac{1}{r_j \times Z}\right)} \tag{16}
$$

Finally, the cycle length with slack can be computed as in Eq. 17.

$$
T \geq \frac{O(n)}{1 - \left(\frac{\sum_{i=1}^{n}\tau_i}{Z}\sum_{j=1}^{Z}\frac{1}{r_j}\right)} \tag{17}
$$