Adaptive Cycle Extension in Multimedia Document Retrieval

Youjip Won and Kyungsun Cho

Division of Electrical and Computer Engineering
Hanyang University, Seoul, Korea
{yjwon|goodsun}@ece.hanyang.ac.kr

Abstract. Cycle based disk scheduling approach is widely used to satisfy the timing constraints of the multimedia data retrieval. While cycle based disk scheduling for multimedia data retrieval provides effective way of exploiting the disk bandwidth, it is possible that ongoing streams get exposed to jitter when the cycle is extended due to commencement of new session. In this article, we present the novel idea of avoiding temporal insufficiency of data blocks, jitter, which occurs due to the commencement of new session. We propose that sufficient amount of data blocks be available on memory such that the ongoing session can survive the cycle extension. This technique is called "pre-buffering". We examine two different approaches in pre-buffering: (i) loads all required data blocks prior to starting retrieval and (ii) incrementally accumulates the data blocks in each cycle. We develop an elaborate model to determine the appropriate amount of data blocks necessary to survive the cycle extension and to compute startup latency involved in loading these data blocks. The simulation result shows that limiting the disk bandwidth utilization to 60% can greatly improve the startup latency as well as the buffer requirement for individual streams. The algorithm proposed in this work can be effectively incorporated into modern streaming server design.

Keywords: Disk Scheduling, Round, Jitter, Multimedia, Streaming, Zoned Disk, Buffer Management

1 Introduction

1.1 Motivation

Recent advances in speed of microprocessor, communication medium, storage technology enable the user to enjoy online streaming service at the cost effective manner. Main challenge in providing the remote streaming service is to guarantee the continuous flow of the data blocks from the very source of the data to the client. "Session" is the state where the sequence of data blocks is delivered from the server to the client end. Technologies related to three categories, i.e. server, network transport and the client exploit their respective resources to guarantee continuous flow of data blocks and to eventually achieve the higher Quality of Service playback. Higher level programming constructs, e.g. MHEG, SMIL, XML, etc is used to specify the continuity, synchronization, or QoS requirement of the multimedia document presentation. In this article, we deal with the server side issue involved in retrieving

A.B. Chaudhri et al. (Eds.): EDBT 2002 Workshops, LNCS 2490, pp. 391-405, 2002.

the data blocks from the disk. For the continuous flow of data blocks, individual sessions need to be allocated a certain fraction of disk bandwidth. The streaming server software(or underlying operating system) is responsible for allocating disk bandwidth to individual sessions and achieves the objective via properly scheduling the disk I/O requests from applications.

Legacy disk scheduling algorithms such as FIFO, SCAN, C-SCAN, etc. focus on maximizing the disk throughput or minimizing the average response time. These scheduling algorithms are not able to provide any bandwidth guarantee to individual application. For successful playback of multimedia document, data blocks needs to be supplied to the consumer, which can reside in either same address space or in remote space, in continuous fashion conformant to specified data rate. For the continuous supply of the data blocks, it is critical that each streaming session is allocated required fraction of data transfer capacity of the disk. A number of disk scheduling algorithms have been proposed for this purpose. Most widely used strategy in scheduling the disk operation for multimedia data retrieval is to use the notion of cycle(or round). In a cycle based disk scheduling algorithm, individual session is supplied a certain amount of data block in each cycle. The amount of data blocks retrieved for individual session in a cycle should be sufficient for playback for the length of cycle. The length of the cycle depends on the aggregate playback bandwidth in the disk subsystem and hence forth, the cycle length needs to be adjusted with respect to the aggregate playback bandwidth at the disk as new streaming session starts or the ongoing session terminates. While cycle based disk scheduling delivers effective utilization of disk bandwidth, online extension of a cycle causes jitter to some of the users because the amount of data blocks which have been read from the disk in each cycle is not sufficient to survive the newly extended cycle. This phenomenon can actually be observed in commercially available streaming servers. In some existing implementations of the streaming server, they resolve this problem by setting the length of round sufficiently large in the beginning and do not change it[1-3]. The amount of data blocks loaded in each round does not change either. While this simple approach does not have problem of temporal insufficiency of data blocks, it has important drawbacks: long startup latency and wastage of synchronization buffer. We carefully believe that in a certain situation, it may be better to adaptively change the cycle length as workload intensity changes. However, in cycle based disk scheduling in retrieving the multimedia data, cycle extension entails the temporal insufficiency of data blocks and thus may cause jitter to some of the ongoing streams. In this work, we propose a technique, “pre-buffering”, which enables to absorb this jittery situation by preload a certain amount of data blocks prior to starting the service. Our modeling approach in this article is towards multi-zoned disk system.

1.2 Related Works

A number of works presented fine model for modern disk subsystem[4],[5]. These models serve as the basis in developing the disk scheduling algorithm for multimedia data retrieval[6-9]. To provide online playback of multimedia data, the server is required to retrieve the data blocks from the disk satisfying a certain data transfer rate. The rate variability in multi-zoned disk thus adds another dimension of complexity in developing the scheduling algorithm for multimedia file system. The scheduling

policies proposed in [6-9] do not take into account the fact that disk transfer rate varies depending on the cylindrical position of the disk head. Ghandaharizadeh et al[10] presented the placement model of multimedia data and the disk scheduling technique in multi-zoned disk. They effectively incorporated the variability in data transfer rate of the disk. Meter et al[11] proposed an analytical model for multi-zoned disk and performed physical experiment of the file system performance in zoned disk. Their results show that peak transfer rate drops roughly 25% depending on the position of the disk head. The experiment is performed on the BSD Fast File System. Since the outer track has faster transfer rate, it is more beneficial to place the popular file in outer track. However, placing files in outer track causes occasional long seek movement and thus can adversely affect the disk throughput. Tewari et al[12] proposed the method of placing the data blocks in zoned disk based on its access frequency while delivering reasonable seek overhead. Tse et al[13] showed that multi-zoned disk exhibits significant improvement in throughput and proposed optimal partitioning scheme to achieve maximum transfer rate. Neogi et al.[14] investigate the behavior of low power disk where the disk platter stops rotating when the system is idle. They model the data loading operation for multimedia data retrieval. They proposed to pre-buffer the data blocks when fraction of a round is unused.

2 Disk Scheduling for Multimedia Document Retrieval

2.1 Disk Mechanism

Magnetic disk drives consist 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. The triple <cylinder, head, sector> uniquely identifies the location of data blocks. The time to read(or write) the data blocks from (or to)the disk consists of seek latency(time to move the disk head to the respective track), rotational latency(time to rotate the platter so that the target block arrives underneath the disk head), and data transfer time(time to read/write the data blocks). In random disk I/O, major fraction of the I/O latency consists of seek time[15]. The objective of multimedia disk scheduling is to retrieve the data blocks while providing a certain playback bandwidth to individual streaming session. SCAN-EDF policy[16] can provide a certain level of bandwidth guarantee. However, it may suffer from low disk bandwidth utilization due to the excessive head disk movement overhead. More effective method of retrieving the multimedia blocks from the disk are cycle based disk scheduling[6-9].

Most of the modern disk drive adopts the technology called *Zoning*. Zoning is a technique adopted by hard disk manufacturers to increase the capacity of the disk. In Multi-zoned disk, a number of adjacent cylinders are group 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 number of sectors. The objective of this technique is to exploit the constant linear bit density such that the outer cylinders have larger storage capacity than the inner disk. Multi-zoning certainly provides superior storage efficiency. However, the disk exhibits different data transfer rate depending on the

position of the disk head. While zoning technique provides effective utilization of the storage capacity, its varying transfer rate adds another dimension of complexity in scheduling the data block retrieval operation. Especially in an application such as multimedia data retrieval, the variability in the data transfer rate makes the scheduling problem more complicated. [8, 9] investigated disk scheduling issue for continuous media retrieval in zoned disk. Ghandaharizadeh et al [10] proposed to place the multimedia data blocks to each zone in round-robin fashion and proposed SCAN based disk scheduling algorithm for multimedia data retrieval.

2.2 Cycle Based Disk Scheduling for Multimedia Data Retrieval

Fig. 1 illustrates the retrieval of data blocks based on round based disk scheduling. There are three client sessions. Each of these clients consumes data blocks at a certain data rate. The server needs to retrieve the data blocks for these streams satisfying the data rates of the clients. Cycle is the interval between the read bursts in the server. Server retrieves a certain amount of data blocks in each cycle so that it can supply the data blocks to client at some fixed rate. There is important discrepancy between the playback operation and the disk retrieval operation. Playback of multimedia data is synchronous operation e.g. 30 frames/sec, but the operation of retrieving the data blocks from disk is asynchronous. To compromise this discrepancy, a certain amount of memory buffer is dedicated between the disk retrieval operation and the playback operation so that the buffer can absorb the interval variance between the successive data block retrieval. One of the most important and challenging issue is to schedule a set of disk I/O request issued from a number of streaming session while providing a certain level of data rate guarantee to individual sessions.

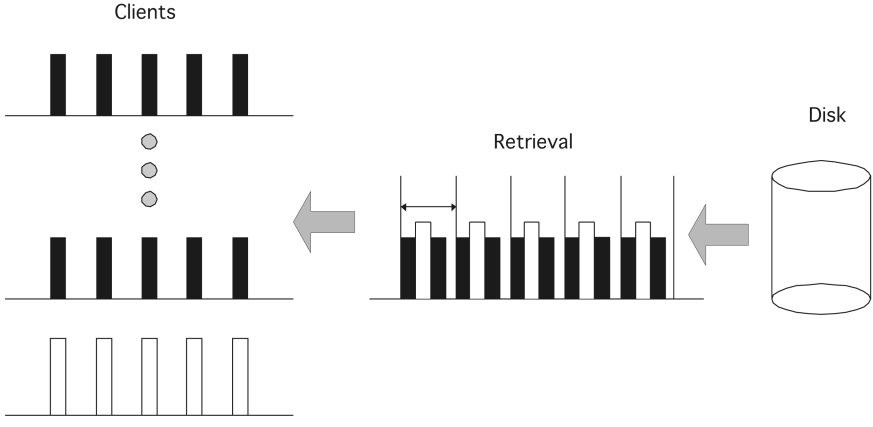

Fig. 1. Round Based Scheduling and Multimedia Document Retrieval

We formulate the general constraints in disk scheduling for continuous media playback. Let $s = \{s_1,...,s_n\}$ be s 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. A cycle is the length of the time interval between successive bursts of read operations. Let $T(s)$ be the length of the cycle for playbacks s. Continuity guarantee problem is to determine period time T and to compute the amount of the data blocks which needs to be supplied from the disk to each session. Two conditions of continuity guarantee can be formally described as in Eq. 1 and Eq. 2. *T*, $r_{display}$, n_i and *b* denotes the length of the cycle, playback rate, the number of blocks to be read in a cycle for stream *i*, and the size of block, respectively. Eq. 1 illustrates the condition that the number of blocks read in a cycle should be sufficient for playback of length *T*.

$$T \times r_{display} \le n_i \times b \tag{1}$$

Eq. 2 denotes the condition that the time to read data blocks for all ongoing sessions for single cycle's playback should be less than *T*. T_{seek}, $T_{latency}$, $T_{fullseek}$, *Z*, r_j and *S* denotes the average seek time, rotational latency, worst case seek time, the number of zones in the disk and data transfer rate of zone *j* and the number of on going streams.

$$T \ge \sum_{i=1}^{S} \sum_{j=1}^{Z} \frac{n_i \times b_i}{Z \times r_j} + \sum T_{seek} + \sum T_{latency} + T_{fullseek} \tag{2}$$

Solving Eq. 1 and Eq. 2, the length of cycle *T* can be obtained as in Eq. 3. Details can be found in [8]. ε in Eq. 3 corresponds to the total disk head movement overhead, i.e. $\varepsilon = \sum T_{seek} + \sum T_{latency} + T_{fullseek}$.

$$T \ge \frac{\varepsilon}{1 - \left(\frac{r_{display} \times S}{Z} \times \sum_{j=1}^{Z} \frac{1}{r_j} \right)} \tag{3}$$

From the condition $T \times r_{display} \le n_i \times b_i$ and Eq. 3, we can obtain the number of data blocks to read in a cycle.

3 Starting New Session

3.1 Cycle Extension

In practice, the number of streaming sessions dynamically changes and the amount of data blocks retrieved from the disk dynamically changes accordingly as well. It may be due to the arrival of the new service request, termination of the ongoing service session, temporal suspension of the playback, or etc. It is also possible that playback rate of a stream can dynamically changes as available bandwidth in the underlying network changes[17]. While cycle based disk scheduling policy efficiently utilizes the disk bandwidth, it cannot seamlessly adapt to the dynamic change in the playback bandwidth. As can be seen in Eq. 3, the cycle length and the amount of data blocks read in a cycle needs to be dynamically adjusted in accordance with the change of aggregate playback rate from the disk. However, the extension of cycle entails the temporal insufficiency of data blocks and subsequently causes jitters to on going streams.

Fig. 2 illustrates the occurrence of jitter in ongoing stream when cycle length is extended due to the start of new session. Top half of Fig. 2 illustrates that the data block is retrieved from the disk. In the

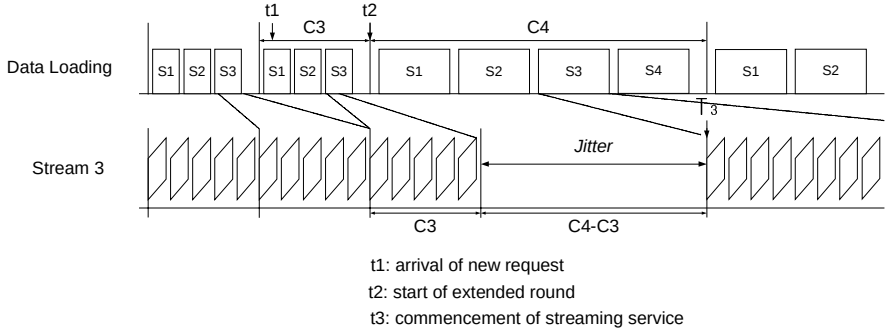

Fig. 2. Extension of Cycle of Jitter

beginning, the disk subsystem is servicing three streams, s_1, s_2 and s_3 and the respective cycle length is denoted by C_3. Data blocks for s_1, s_2 and s_3 are retrieved in round-robin fashion in each cycle. In practice, the order in which the data blocks for each stream are retrieved is subject to the underlying disk scheduling algorithm[7, 8, 18]. The new service request, s_4, arrives at t_1. When the new request arrives, the resource allocation and call admission module checks whether it is possible to support the newly arriving request and the server computes the new cycle length. Based upon the updated cycle length, the amount of data blocks to be read in a newly extended cycle is determined. In Fig. 2, cycle is extended to accommodate new session from the third cycle. It is important of note that data blocks loaded in the extended cycle are available for playback only after t_3. The lower part of Fig. 2 illustrates the situation where s_3 consumes the data blocks supplied from the disk. Data blocks fetched in the second cycle is for C_3's playback duration. Since the blocks fetched in the third cycle will be available only after t_3, it is inevitable that s_3 is exposed temporal lack of data blocks due to the delays in data block retrieval.

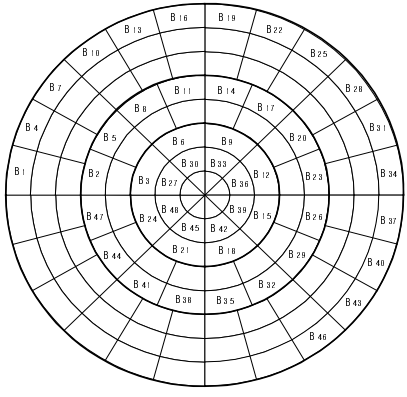

Fig. 3. Round Robin Block Placement in Zoned Disk

3.2. Data Block Placement in Zoned Disk

A number of approaches have been proposed to effectively utilize the variable transfer rate of zoned disk for multimedia data retrieval. The simplest approach is to use the average transfer rate of multi-zoned disk[12]. 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[10] proposed to place the data blocks to each zone in round-robin fashion. In this work, we assume that the data blocks are placed using the placement strategy proposed in [10]. Fig. 3 illustrates placement of the data blocks in multi-zoned disk. The disk consists of three zones in Fig. 3. 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. 4 illustrates the number of data blocks retrieved in a cycle for a stream when the playback rate is 1.5 Mbits/sec and data block size is 4 KByte. The amount of data blocks loaded in a cycle is slightly larger in multi zoned disk than in the single zone disk since the number of data blocks loaded in a cycle is integer multiples of the number of zones.

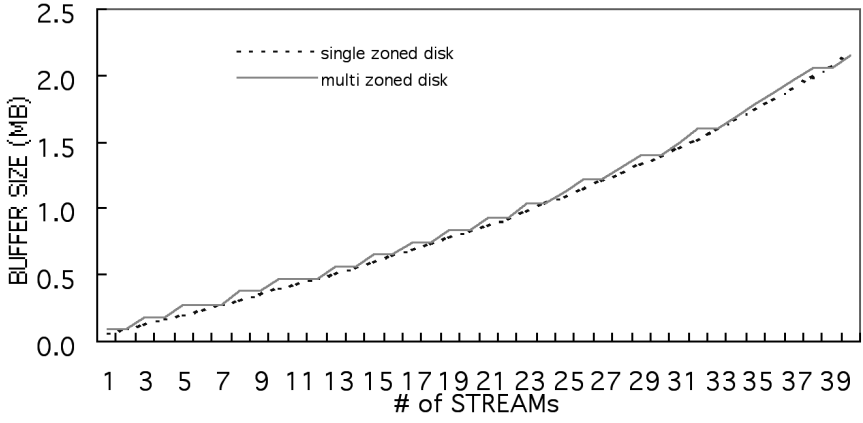

Fig. 4. Synchronization Buffer Size: Multi Zoned vs. Single Zoned Disk

4 Resolving the Cycle Extension Overhead

4.1 Pre-buffering

As illustrated in Fig. 2, ongoing sessions may suffer from the temporal insufficiency of data blocks when the cycle length is extended. If each stream has sufficient amount of data blocks available, it can survive cycle extension. In this work, we propose to load the “sufficient” amount of data blocks to memory prior to start the service. We call the operation of loading the data blocks prior to starting the service as *pre-buffering*. In practice, pre-buffered data blocks can reside in the server side or in the user side. We presently assume that pre-buffering happens in server end, but we

expect that there is not much difficulty in extending this idea to user end pre-buffering.

Fig. 5 illustrates how preloading can resolve the insufficiency of data blocks when extending a cycle. X and y axis denotes the time and the amount of data blocks in memory, respectively. *L* and *m* denote the service start up latency and the amount of data blocks loaded prior to start the service. c_i denotes the length of the cycle to service *i* number of streams and let us assume that there are *n* number of ongoing streams at time t_0. Cycle length is c_n, initially. New service request arrives at t_1. As a result of new request arrival, cycle length is extended to c_{n+1} to accommodate the new session. Prior to starting service, disk subsystem pre-buffers a certain amount of data blocks in addition to data blocks for c_n's playback. *m* denotes the amount of data blocks which should be available prior to starting service. When the cycle length is extended to accommodate the new stream, system requires c_{n+1}'s worth of data blocks to avoid any jitter. By preloading a certain amount of data blocks, it is possible to avoid temporal insufficiency of data blocks.

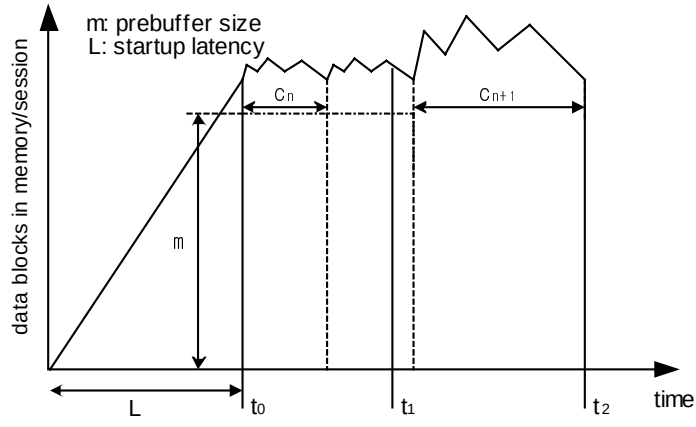

Fig. 5. Pre-buffering

There are two important issues in pre-buffering. The first issue is to determine the amount of data blocks for preloading, *m*. The second issue is to determine the time to start the service. With pre-buffering, it is unavoidable that user experiences longer start-up latency since the playback starts only after sufficient data blocks are accumulated in memory. By keeping more data blocks in memory, ongoing session becomes more robust against jitter caused by cycle extension. The ideal situation is to maximize the amount of data blocks available in memory data while minimizing the startup latency. Unfortunately, start-up latency and the amount of pre-buffering data are mutually dependent factors and we have to sacrifice one at the cost of the other. If the server starts to dispatch the data blocks before sufficient data blocks are loaded in memory, it is more likely that the service session suffer from the jitter caused by cycle extension.

In this article, we introduce two approaches in pre-buffering the data blocks. The first approach is to finish pre-buffering prior to start service. We call it as *simple pre-buffering*. This approach is shown in Fig. 5. The length of cycle and the amount of data blocks read in a cycle is set as small as possible while satisfying the continuity

requirement(Eq. 3). The problem in this approach is long startup latency. User has to wait several of cycles for the service until sufficient amount of data blocks(*m*) becomes available in memory. The second approach is to make the length of cycle large enough so that the amount of data blocks read is larger than the amount of data blocks consumed in a cycle. We call this approach as *incremental pre-buffering*. In incremental pre-buffering, data blocks are accumulated in memory as the playback proceeds since the amount of the data blocks read in a cycle exceeds the amount of data blocks consumed in a cycle. These surplus data blocks will be used when cycle extension occurs. In incremental pre-buffering, service can start immediately and thus the user experiences relatively short startup latency. It is possible that the streaming session has not accumulated sufficient data blocks when another new streaming request arrives. Then the ongoing stream will suffer from jitter.

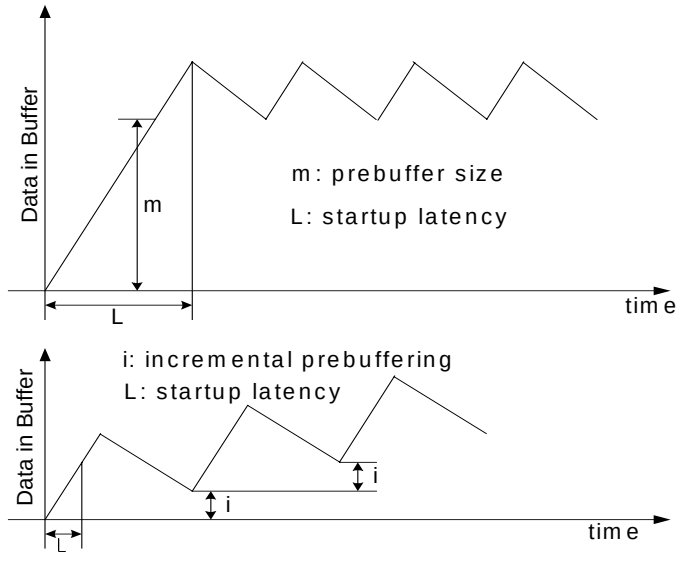

Fig. 6. Simple Pre-buffereing vs. Incremental Pre-buffereing

5 Simple Pre-buffering

5.1 Block Retrieval in Simple Pre-buffering

The length of the cycle and the amount of data blocks read in a cycle is proportional to the disk bandwidth utilization. More importantly, this amount increases very fast as the disk bandwidth utilization approaches 100%. The asymptotic slope follows $1/(1-\rho)$, where ρ is disk bandwidth utilization[10]. It is advised not to fully utilize the bandwidth of the disk due to its excessive buffer overhead. Actual limit on the

maximum number of concurrent streams needs to be much lower than the physical capacity of the disk. In simple pre-buffering scheme, the server selects the upper bound on aggregate disk bandwidth(or equivalently disk bandwidth utilization). The issue of determining the reasonable bound of disk bandwidth utilization in multimedia streaming environment is quite subjective topic and we do not discuss this topic in detail here. When new request arrives, admission control module checks whether it can accept the new request. One of criteria for call admission is if the sum of playback rates of ongoing sessions including the newly arrived request is less then the predefined upper bound for disk bandwidth utilization.

In simple pre-buffering, playback starts only after we load sufficient amount of blocks in memory. We have to provide clearer notion on the term "sufficient". Given the upper bound on disk bandwidth, we can compute length of cycle and the amount of data blocks read in a cycle when the disk utilization reaches its upper bound(Eq. 1, Eq. 3). Let T_{max} and B_{max} be the length of cycle and the amount of data blocks read when the disk is fully utilized by its upper limit. The length of the cycle will be extended with the arrival with the new request until disk reaches its service limit. Thus, if individual session initially loads the data blocks which is required to survive T_{max}'s playback, then it will not be affected by the extension of a round. In simple pre-buffering, the playback starts after B_{max} amount of data blocks are fetched into memory. Let T_i and B_i be the cycle length and the amount of data blocks retrieved in a cycle for a stream when there are i concurrent streams, respectively. Given that there are i number of streams, the server retrieves B_i amount of data from the disk and dispatch them to user in each cycle. Cycle extension keeps occuring until the number of concurrent session reaches the upper limit. To survive the cycle extension, B_{max} amount of data blocks needs to be available in memory for each stream. Total buffer size for individual session is the sum of pre-buffered data and the amount of data retrieved in each cycle. It can be formulated as in Eq. 4. i denotes the number of concurrent sessions.

$$Buffer_i = B_{\max} + B_i = (T_{\max} + T_i) \times r_{display} \tag{4}$$

5.2 Latency in Simple Pre-buffering

Individual session suffers from longer service startup latency because client can consume the data blocks only after B_{max} amount of data blocks becomes available in memory. L in Fig. 5 corresponds to session start-up latency in pre-buffering. We like to examine the length of startup latency in more detail. Let us assume that with the arrival of new request, the cycle length is extended to T_i. The amount of data blocks loaded in each cycle is B_i. The new stream can start only after B_{max} amount of data blocks are accumulated in memory and thus, the start up latency, L, can be formulated as in Eq. 5.

$$L = \left\lceil \frac{B_{\max}}{B_i} \right\rceil * T_i \tag{5}$$

6 Incremental Pre-buffering

As we observed in Eq. 5, start-up latency can be non-negligible if it is required to preload B_{max} amount of data blocks prior to starting service. To improve the startup latency, we propose a scheme called *Incremental Pre-buffering*. In *Incremental Pre-buffering*, we exploit the fact that if the cycle length is longer than the minimum cycle length requirement in Eq. 3, a certain fraction of cycle becomes idle, and subsequently idle fraction of a cycle can be used to accumulate the data blocks for each session to survive the cycle extension. In *Incremental Pre-Buffering*, the server does not have to wait until the B_{max} amount of data blocks is available in memory. It can start playback as soon as the sufficient amount of data blocks for single cycle's playback duration becomes available. Let us examine the characteristics of incremental pre-buffering in more detail. Let T be the length of cycle and we like to use $1-\alpha(0<\alpha<1)$, fraction of each cycle for incremental pre-buffering. Then, continuity requirement of Eq. 2 needs to be re-stated as in Eq. 6.

$$\alpha T \geq \sum_{i=1}^{S}\sum_{j=1}^{Z}\frac{n_i \times b_i}{Z \times r_j} + \sum T_{seek} + \sum T_{latency} + T_{fullseek} \tag{6}$$

Eq. 6 states the condition that total time to retrieve all the data blocks for T 's playback should be than αT . Eq. 6 and Eq. 1 together implies that the amount of data blocks read in a cycle should be sufficient for T's playback(Eq. 6), and time to read all the data blocks for a cycle should be less than αT(Eq. 2). In each cycle, $T - \alpha T$ remains unused and thus can be used to accumulate the data blocks. Solving Eq. 6 and Eq. 2, we can obtain the amount of data blocks to be loaded in each cycle as in Eq. 7. Details of derivation step can be found in [15].

$$n \geq \frac{O(n)r}{\alpha - \left(\frac{\sum_{i=1}^{n} r_i}{Z} \times \sum_{j=1}^{Z}\frac{1}{B_j}\right)} \times \frac{1}{b_i} \tag{7}$$

Let us examine the relationship between the amount of data blocks accumulated in each cycle and α. *(1-*α*)* fraction of the cycle will be used to incrementally *preload* the data blocks. Since retrieving the additional data blocks from the same file does not entail any additional disk head movement overhead, *(1-*α*)* fraction of cycle can be effectively dedicated to transferring the data blocks. In single zoned disk, we can compute the amount of data blocks preloaded for individual session in each cycle, $b_{pre\text{-}buffer}$, as in Eq. 8. n and r_{max} denotes the number of sessions and maximum transfer rate of the disk.

$$b_{prebuffer} = \frac{\left((1-\alpha)T\right)}{n} r_{\max} \tag{8}$$

In multi-zoned disk, amount of preloaded data blocks in each cycle needs to be the integer multiples of the number of zones since the server retrieves the same number of blocks from each zone. We like to obtain the amount of data blocks which can be preloaded in each cycle under multi-zoned disk. Let n, m, B_i and z be the number of sessions, the number of blocks preloaded for a single stream in a zone, maximum transfer rate of zone i and the number of zones, respectively. In a single zone, m number of blocks are loaded for pre-buffering for a stream, and thus mb/B_i

corresponds to the time to accumulate the data blocks for a single stream in a single zone where *b* denotes the block size. The total time to *accumulate* the data blocks in a cycle should be less than *(α-1)T* and this relationship can be formulated as in Eq. 9.

$$0 < \frac{n\alpha\left(\sum_{i=1}^{z}\frac{mb}{B_i}\right)}{(1-\alpha)T} \le 1 \tag{9}$$

The amount of preloaded buffer for individual session for each cycle can be formulated as in Eq. 10 where m_{max} in Eq. 10 is largest *m* satisfying Eq. 9.

$$b_{prebuffer} = m_{\max} zb \tag{10}$$

Fig. 6 characterizes the difference between the *Simple Pre-Buffering* and *Incremental Pre-Buffering*. Simple pre-buffering has longer startup latency since the start of service is delayed until the B_{max} amount of data blocks are accumulated on memory. On the other hand, *Incremental pre-buffering* incrementally accumulates the data blocks in each cycle and thus enables the session to start service right after the first cycle completes. However, in incremental pre-buffering, it is possible that new request arrives before the sufficient amount of data blocks are accumulated in memory and thus ongoing streams get exposed to jitter.

7 Simulation Study

We perform simulation based experiment to verify the effectiveness of the proposed prebuffering strategy. The disk is modeled after IBM Deskstar 34 GXP with storage capacity of 27.3 GB. The transfer rate of the disk ranges from 13.8 MByte/s to 22.9 MByte/s depending on the position of disk head. There are 17494 numbers of cylinders and the number of zones is 12. The disk rotates at 7200 RPM.

7.1 Pre-buffering Overhead

Loading sufficient amount of data blocks entails a certain overhead from the aspect of memory as well as from the aspect of startup latency. We first examine the amount of data blocks which is required to be preloaded. The size of pre-buffer depends on the predefined service limit of the disk subsystem. We examine the pre-buffer size ATSC compressed stream(19.2 Mbits/sec) and MPEG-1 compressed stream(1.5Mbits/sec). The disk used in the simulation can theoretically support upto seven ATSC compressed streams and 95 MPEG-1 compressed streams, respectively. The buffer size allocated to individual session is sum of the size of pre-buffered data and the amount of the data blocks retrieved in a cycle. Fig. 7 illustrates the buffer size for each stream when the upper bound on the disk service limit is set to 70, 80, and 90, respectively. As can be seen in Fig. 7 and Fig. 8, buffer size varies widely depending on the upper bound of the concurrent number of streams. In Fig.7, i.e. in case of MPEG1 compressed stream with 1.5 Mbits/sec playback rate, per stream buffer size is 4 Mbyte when the number of concurrent streams is 70. However, the buffer size

increases six times when the maximum number of concurrent streams is 90. We can observe the similar phenomenon when the playback rate of the stream is 9 Mbits/sec. Fig. 8 illustrates the buffer size when the maximum number of streams is 5, 6, and 7 respectively. When the maximum number of streams is 5, the buffer size is 3.5 Mbyte. However, when the maximum number of concurrent streams is 7 which is the largest number of concurrent stream supported by the disk, the buffer size increases by eight times.

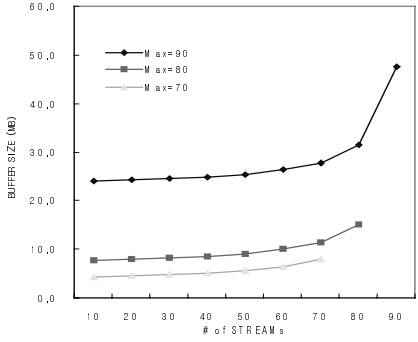

Fig. 7. Pre-buffer Size(1.5 Mbits/s stream)

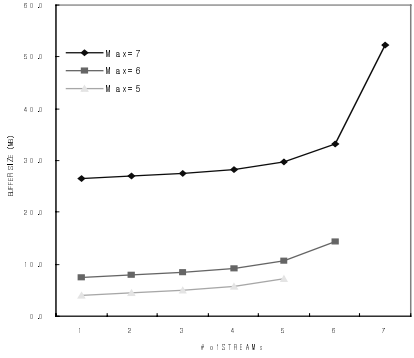

Fig. 8. Pre-buffer Size(19.2 Mbits/sec)

The worst case buffer requirement is the same for "pre-buffering" and "incremental pre-buffering". However, incremental pre-buffering reduces the startup latency of the client since the streaming server dispatches the data blocks after the first cycle. This is to be discussed in detail in section7.2.

7.2 Start-Up Latency

Fig. 9 illustrates the startup latency for ATSC compressed stream. When the maximum number of concurrent streams is 7, startup latency is 10.5sec. With the maximum number of streams being 5 and 6 respectively, the startup latency is 1.3 sec and 2.5 sec, respectively, in simple prebuffereing. In the incremental pre-buffering, the start up latency is not governed by the maximum number of concurrent streams, but is proportional to the number of concurrent sessions. The startup latency is actually longer in the incremental pre-buffering when the maximum number of concurrent streams is set to five or six.

8 Summary

While cycle based disk scheduling for multimedia data retrieval provides effective utilization of disk bandwidth, ongoing streams get exposed jitter when the cycle is extended due to commencement of new streaming session. In this article, we present the novel approach of avoiding temporal insufficiency of data blocks, *jitter*, which

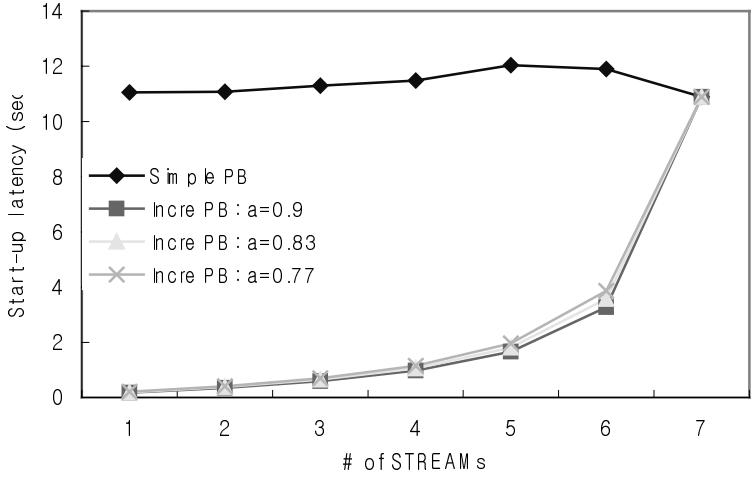

Fig. 9. Start-up latency with 19.2 Mbps

occurs due to cycle extension. We propose a technique called *pre-buffering* which is to make the sufficient amount of data blocks available on memory such that the streaming session can survive the temporal insufficiency of data blocks with cycle extension. The length of cycle keeps extended with the arrival of new stream until the number of concurrent session reaches the upper bound. We propose two ways of making the sufficient amount of data blocks available in memory. The first approach is to load the sufficiently large amount of data blocks in memory prior to start service. In this scheme, the amount of data blocks retrieved from the disk in each cycle is precisely what needs to be sent to the client. In our simulation study, we find that fully exploiting disk bandwidth capacity makes the startup latency prohibitively larger, i.e. more than 10 sec. However, if the streaming server is set not to utilize more than 60% of the disk bandwidth capacity the startup latency is within the tolerable range, around 2 sec. In incremental pre-buffering, each session retrieves more data blocks than it consumes in a cycle. Subsequently, each session can accumulate the data blocks as playback proceeds and the accumulated data blocks can be used to survive the cycle extension. The main advantage of incremental pre-buffering is relatively short startup latency. In this article, we develop an elaborate model to compute the length of the cycle which incorporate the time to preload the data blocks in both full pre-buffering and incremental pre-buffering. The result of this work can be effectively incorporated into modern streaming server design especially in file system, disk scheduling, and resource allocation module.

References

[1] R. Wijayaratne and A. L. N. Reddy, "Techniques for Improving the throughput of VBR Sreams," presented at ACM/SPIE Multimedia Computing and Neworking, 1999.

[2] P. Shenoy and H. M. Vin, "Cello: A Disk Scheduling Framwork for Next Generation Operating Systems," presented at ACM Sigmetrics Conference, Madison, WI, USA, 1998.
[3] W. J. Bolosky, R. P. Fitzgerald, and J. R. Douceur, "Distributed Schedule Management In The Tiger Video Fileserver," *ACM SIGOPS Operating Systems Review (ACM)*, vol. 31, 1997.
[4] C. Ruemmler and J. Wilkes, "Introduction to Disk Drive Modeling," *Computer*, vol. 27, pp. 7-28, 1994.
[5] B. L. Worthington, G. R. Ganger, and Y. N. Patt, "Scheduling for Modern Disk Drives and non-random workloads," University of Michigan CSE-TR-194-94, March 1994.
[6] B. Ozden, A. Biliris, R. Rastogi, and A. Silberschatz, "A Low-Cost Storage Server for Movie on Demand Databases," *Proc. of VLDB '94*, 1994.
[7] D. R. Kenchammana-Hosekote and J. Srivastava, "Scheduling Continuous Media on a Video-On-Demand Server," *Proc. of International Conference on Multi-media Computing and Systems, Boston, MA, IEEE*, 1994.
[8] P. Rangan, H. Vin, and S. Ramanathan, "Designing an on-demand multimedia service," *IEEE Communication Magazine*, vol. 30, pp. 56-65, 1992.
[9] J. Gemmell, "Multimedia Network File Servers:Multi-Channel Delay Sensitive Data Retrieval," *Proc. of 1st ACM Multimedia Conf. ACM*, 1993.
[10] S. Ghandeharizadeh, S. Kim, and C. Shahabi, "Continuous Display of Video OBjects Using Multi-Zoned Disks," *Technical report, University of Southern California*, 1995.
[11] R. V. Meter, "Observing the Effects of Multi-Zone Disks," presented at Usenix Technical Conference, San Jose, CA, USA, 1997.
[12] R. Tewari, "Placement of Multimedia Blocks on Zoned Disks," presented at IS&T/SPIE Conference on Multimedia Computing and Networking(MMCN), San Jose, CA, USA, 1996.
[13] P. K. C. Tse and C. H. C. Leung, "Improving Multimedia Systems Performance," *ACM Multimedia Systems Journal*, vol. 8, pp. 47-55, 2000.
[14] A. Neogi, A. Raniwala, and T.-c. Chiueh, "Phoenix: A Low-Power Fault-Tolerant Realtime Network-Attached Storage Device," presented at ACM Multimedia, Orlando, FL, USA, 1999.
[15] Y. Won and J. Srivastava, "SMDP: Minimizing buffer requirements for continuous media servers," *ACM/Springer Multimedia Systems Journal*, vol. 8, pp. 105-117, 2000.
[16] A. L. N. Reddy and J. C. Wyllie, "I/O Issues in a Multimedia Systems," in *IEEE Computer Magazine*, 1994.
[17] R. Rejaie, M. Handley, and D. Estrin, "Layered Quality Adaptation for Internet Video Streaming," *IEEE Journal on Selected Areas of Communications*, 2000.
[18] M.-S. Chen, D. D. Kandlur, and P. S. Yu, "Optimization of the grouped sweeping scheduling(gss) with heterogeneous multimedia streams," *ACM Multimedia '93*, pp. 235-242, 1993.