Resolving Cycle Extension Overhead Multimedia Data Retrieval[1]

Youjip Won and Kyungsun Cho

Abstract: In this article, we present the novel approach of avoiding temporal insufficiency of data blocks, jitter, which occurs due to the commencement of new session. We propose to make the sufficient amount of data blocks available on memory such that the on-going 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 playback 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.

Keywords: disk scheduling, cycle, jitter, multimedia, streaming, zoned disk, buffer management

I. Introduction

1. Motivation

In this article, 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. We propose a model to compute the amount of data blocks to be loaded to avoid startup jitter. In this work, our modeling and scheduling effort is targeted towards multi-zoned disk environment.

2. Related works

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. [5] 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. [8] proposed an analytical model for multi-zoned disk and performed physical experiment of the file system performance in zoned disk. [6] proposed the method of placing video files depending on its access frequency. [9] showed that multi-zoned disk exhibits significant improvement in throughput and proposed optimal partitioning scheme to achieve maximum transfer rate. [11] examine the behavior of low power disk whose platter does not rotate when the system is idle and model the data block loading operation for multimedia data retrieval.

II. Multimedia disk scheduling in zoned disk

The objective of multimedia disk scheduling is to retrieve the data blocks while providing a certain playback bandwidth to individual streaming session. Multi-zoning is a technique adopted by hard disk manufacturers to increase the capacity of the disk. The objective of zoning is to exploit the storage capacity in the cylinder with larger perimeter using the constant linear bit density. 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.

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

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.

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

Solving Eq. 1 and Eq. 2, the length of cycle T can be obtained as in Eq. 3. Details can be found in [5]. ε in Eq. 3 corresponds to the total disk head movement.

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

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

III. Starting new session in zoned disk

In the streaming server, the number of ongoing playbacks or the aggregate playback bandwidth from the disk subsystem dynamically changes. 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. The extension of cycle entails the temporal insufficiency of data blocks and subsequently causes jitters to on going streams.

Fig.1 illustrates the occurrence of jitter in ongoing stream when cycle length is extended due to the start of new session. Top half of Fig. 1 illustrates sequence of data block loading from the disk. Initially, the disk subsystem is servicing three

Manuscript received: June 18, 2001, Accepted: Dec. 18, 2001.

Youjip Won and Kyungsun Cho: Division of Electrical and Computer Engineering Hanyang University, 133-791, Seoul, Korea.(yjwon/goodsun@ece.hanyang.ac.kr)

※This work is funded by Korea Sanhak Foundation in 2001 and by Hanyang University, Korea made in the program year of 1999.

streams, s_1, s_2 and s_3 and the respective cycle length is denoted by P_1. From the third cycle, the cycle length is extended to accommodate new streaming session. Data blocks loaded in the extended cycle are available for playback only after t_3. The bottom half of Fig.1 illustrates playback sequence of s_3. The amount of data blocks in memory fetched in the second cycle is only for the P_1'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. In this work, we assume that the data blocks are placed using the placement strategy proposed in [5]. 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.

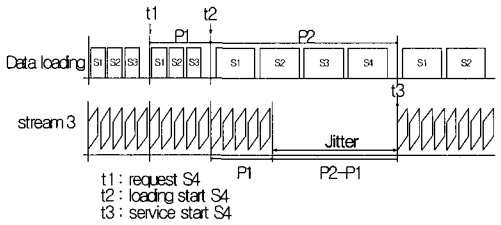


Fig. 1. Extension of cycle and jitter.

IV. Pre-Buffering

1. Concept of pre-buffering

There are two different ways of loading the data blocks. The first approach is to load sufficient amount of data blocks *prior to* starting the service. The amount of preloaded data blocks should be sufficient so that the user can consume these data blocks when cycle extension occurs. The length of the cycle and the amount of data blocks read in a cycle is set as small as possible while satisfying the continuity requirement of Eq. 3. We call the operation of loading the data blocks prior to starting the service as *pre-buffering*. The problem of this approach is long startup latency. 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. Then, we can accumulate a certain amount of data blocks 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. The advantage of the second approach is relatively short startup latency. However, it is possible that the streaming session has not accumulated sufficient data blocks when the new streaming request arrives.

Fig. 2 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. 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. However, the amount of data blocks available at t_1 is for c_n's playback and the data blocks loaded in the newly extended cycle is available after t_2. The streaming session suffer from jitter for duration, $c_{n+1} - c_n$.

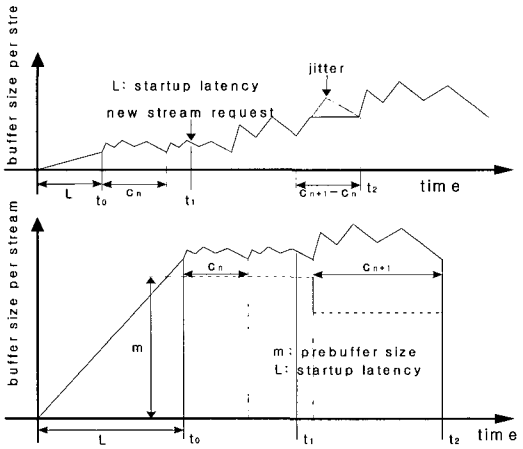


Fig. 2. Preloading of data blocks.

In bottom half of Fig. 2, prior to starting service, disk subsystem prebuffers 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, we can avoid temporal insufficiency of data blocks. However, prebuffering mechanism increases the startup latency.

2. Amount of data blocks to preload

The important issue in prebuffering is to determine the appropriate amount of data blocks for prebuffering. 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%. 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 capacity of the disk.

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. max denotes the maximum number of concurrent streams. Given that there are i number of streams, the server retrieves B_i amount of data blocks from the disk and transfers same amount of data blocks to end 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 amount of buffer for individual session is the sum of and B_i and can be formulated as in Eq. 4.

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

V. Pre-Buffering strategy

1. Pre-Buffering and latency

If each stream has B_{max} amount of data blocks in memory, the stream is free from the temporal insufficiency of data blocks in cycle extension. However, the individual session can

suffer from longer service startup latency because client can consume the data blocks only after B_{max} amount of data blocks becomes available in memory before starting service. L in Fig. 2 corresponds to session start-up latency in pre-buffering. We like to examine the length of startup latency when the server preloads B_{max} amount of data blocks prior to starting service. Let us assume that with the arrival of new stream, the cycle length is extended to T_i. Assuming that the playback rate for the individual session is the same, each ongoing stream is allocated same fraction of a cycle to load the data blocks, i.e. T_i /i fraction of a cycle will be dedicated to individual streams to load the data blocks. Each stream loads B_i amount of data blocks. The newly arrived stream can start the playback only after B_{max} amount of data blocks are accumulated in memory. Thus, the start up latency, τ s can be formulated as in Eq. 5.

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

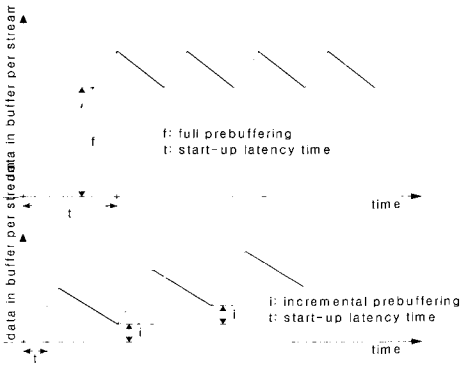


Fig. 3. Full pre-buffering vs. Incremental pre-buffering.

2. Incremental pre-buffering

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 consequently, idle fraction of a cycle can be used to accumulate the data blocks for each session to survive the cycle extension. In Incremental Prebuffering, 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 prebuffering in more detail. Let T be the minimum length of cycle to satisfy the continuity requirement for ongoing service session. We like to extend the length of cycle by extension factor, $\alpha(\alpha>1)$, such that a certain fraction of cycle can be used for accumulating the data blocks. Then, the continuity requirement of Eq. 1 needs to be re-stated as in Eq. 6.

$$\alpha T \times r_{display} \le n_i \times b_i \tag{6}$$

Eq. 6 states the condition that the amount of data blocks read in each cycle should be sufficient for αT_i's playback. However, the Eq. 2, i.e. total time to retrieve all the data blocks, should still be satisfied. Eq. 6 and Eq. 2 together implies that the amount of data blocks read in a cycle should be sufficient for αT_i's playback(Eq. 6), and time to read all the data blocks for a cycle should be less than T_i(Eq. 2). In each cycle, $\alpha T_i - T_i$ 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 [7].

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

Let us examine the relationship between the amount of data blocks accumulated in each cycle and α. $(\alpha-1)$ 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, $(\alpha-1)$ 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, $b_{prebuffer}$ can be formulated as in Eq. 8. n and r_{max} denotes the number of sessions and maximum transfer rate of the disk.

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

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 prebuffering 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 $(\alpha-1)T$ and this relationship can be formulated as in Eq. 9.

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

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. 3 characterizes the difference between the *Full* prebuffering and *Incremental prebuffering*. Full prebuffering 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 prebuffering* 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 prebuffering, it is possible that new request arrives before the sufficient amount of data blocks becomes available.

VI. Simulation study

We perform simulation based experiment to examine the overhead of prebuffering. The disk is modeled after IBM Deskstar 34 GXP(27.3 GB,7200 RPM). The transfer rate of the disk ranges from 13.8 MByte/s to 22.9 MByte/s. There are 17494 numbers of cylinders and the number of zones is 12.

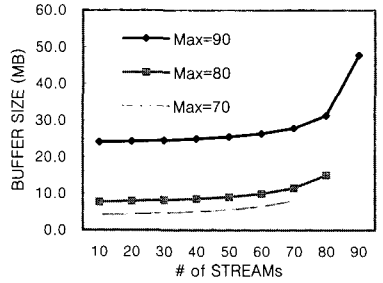


Fig. 4. Prebuffer Size(1.5 Mbits/s stream)

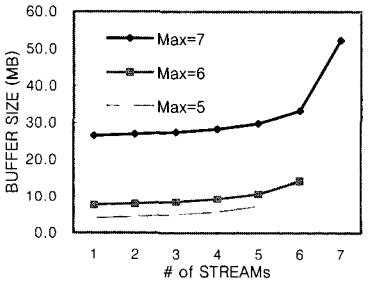


Fig. 5. Prebuffer Size(19.2 Mbits/sec)

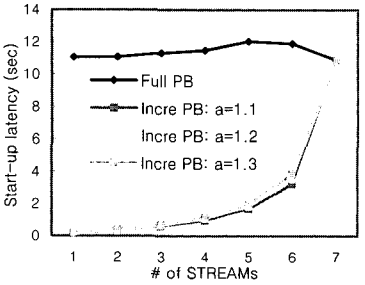


Fig. 6. Start-up latency with 19.2 Mbps.

1. Prebuffering overhead

The size of prebuffer depends on the predefined service limit of the disk subsystem. We examine the prebuffer 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 is size of prebuffer plus the size of the data blocks retrieved in a cycle. Fig. 4 illustrates the buffer size for each stream when the disk service limit is set to 70, 80, and 90, respectively. As can be seen in Fig. 4 and Fig. 5, buffer size varies widely depending on the upper bound of the concurrent number of streams. In Fig.4, 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. 5 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.

2. Start-Up latency

Fig. 6 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 the incremental prebuffering, 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 prebuffering when the maximum number of concurrent streams is set to five or six.

VII. Summary

In this article, we present the novel approach of avoiding temporal insufficiency of data blocks, *jitter*, which occurs due to cycle extension. We propose 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. We propose two ways of making the sufficient amount of data blocks available in memory. The first approach, *Full Prebuffering*, is to load the sufficiently large amount of data blocks in memory prior to start service. In the second approach, *Incremental Prebuffering*, each session retrieves more data blocks than it consumes in a cycle. The main advantage of incremental prebuffering 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 prebuffering and incremental prebuffering. 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] D. R. Kenchammana-Hosekote, and J. Srivastava, "Scheduling continuous media on a video-on-demand server," *In Proc. of International Conference on Multi-media Computing and Systems*, Boston, MA, USA, 1994.

[2] J. Gemmell, "Multimedia network file servers: multi-channel delay sensitive data retrieval," *In Proc. of ACM Multimedia,* Anaheim, CA, USA, Aug. 1993.

[3] B. Ozden, et al., "A low-cost storage server for movie on demand databases," *In Proc. of the 20th Int'l. Conf. on Very Large Data Bases*, pp. 594-605, Santiago, Chile,

September 1994.

[4] Y. Won, and Y. S. Ryu, "Handling sporadic tasks in multimedia file system," *In Proc. of ACM Multimedia,* Los Angeles, CA, USA, *Oct.* 2000.

[5] S. Ghandeharizadeh, S. H. Kim, and C. Shahabi, "Continuous display of Video objects using multi-zone disks," *Second International Baltic Workshop on DB and IS*, 1996.

[6] R. Tewari, et al., "Placement of multimedia blocks on zoned disks," *In IS&T/SPIE Conference on Multimedia Computing and Networking* (MMCN 1996). San Jose, California, USA, Jan. 1996.

[7] Y. Won, *Minimizing Intermedia Interference in Integrated File System*, Submitted for Publication

[8] Rodney Van Meter, "Observing the effects of multi-zone disks", *Usenix 1997 Technical Conference*, Anaheim, CA, Jan 1997.

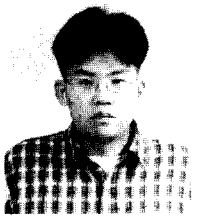
[9] P. K. C. Tse, C. H. C. Leung, "Improving multimedia systems performance", *ACM Multimedia Systems Journal*, vol 8, p.47-55, January, 2000.

[10] A. L. N. Reddy and J. C. Wyllie, "I/O issues in a multimedia systems", *IEEE Computer*, March, 1004, pp. 69-74.

[11] Anindya Neogi, Ashish Raniwala, Tzi-cker Chiueh, "Phoenix: A low-power fault-tolerant real-time network-attached storage device", *ACM Multimedia Conference*, Orlando, FL, USA, Nov. 1999.

Youjip Won

Youjip Won received BS and MS degree from Department of Computer Science, Seoul National University, Seoul, Korea in 1990 and 1992, respectively. He received his Ph.D from Department of Computer Science, University of Minnesota, Minneapolis, USA in 1997. He worked for Intel Corp., as server performance analyst. Since 1999, he is on the board of faculty members in Department of Electrical and Computer Engineering, Hanyang University, Seoul, Korea. His research interests include Multimedia Systems, High Speed Network, Performance Modeling and Analysis, Database.



Kyungsun Cho

Kyungsun Cho received the B.S. degree and the M.S. in Department of Electrical and Computer Engineering, Hanyang University, Korea, in 1999 and 2001, respectively. He is a senior engineer at research center of Net&TV Co., Korea. His research interests include multimedia, digital TV, and media compression technology.