Real-Time Parallel Disk Scheduling for VBR Video Servers[†]

Özgür Ertuğ Mahesh Kallahalla Peter J. Varman
Department of Electrical and Computer Engineering
Rice University
Houston TX 77251

Abstract

We introduce a framework for real-time I/O scheduling for multiple-disk parallel I/O systems. The framework is used to model a video server delivering VBR encoded video data with real-time requirements. The video streams are assumed to be stored in CDL format and distributed across multiple disks.

We present a novel algorithm RT-OPT for optimally prefetching blocks into the server buffer. We show that for any fixed disk-head scheduling policy, RT-OPT is guaranteed to produce a feasible schedule, in which all blocks meet their deadlines, if one exists. Simulations show that the number of clients supported by RT-OPT is superior to intuitive algorithms like GREED-EDF that aggressively keep the disks busy fetching in order of deadlines.

1 Introduction

Video-on-Demand (VOD) systems are distributed server-network-client architectures that provide real-time video service to multiple clients with specified quality of service (QoS) guarantees. Video data is conventionally stored in compressed form on parallel disk arrays. The main goal of a VOD server is to deliver the requested video clips, stored on secondary storage, to the clients in a timely manner honoring QoS guarantees such as delay, delay jitter and maximum frame-loss probability.

Video data compressed in MPEG format inherently results in a variable bit rate (VBR) stream. To simplify storage allocation, avoid fragmentation, and ease buffer management such compressed video streams are stored in constant-sized data blocks. Such constant data length (CDL) storage schemes are preferred over constant time length (CTL) schemes [2]. In this paper, we consider VOD systems that store VBR video data in CDL format on multiple disks. Given a fixed set of resources such as disk bandwidth, storage volume, and main memory buffer, the video server can deliver a limited number of streams while providing the specified QoS guarantee. The primary design goal is to maximize the number of clients that can be concurrently served with the guaranteed QoS.

The storage subsystem is in general the bottleneck for real-time delivery of video due to the high I/O latencies incurred by secondary storage devices. Parallel I/O systems incorporating multiple disks have the potential to alleviate this problem by providing a higher disk bandwidth. However harnessing the raw disk bandwidth to decrease the latency seen by individual blocks requires sophisticated prefetching and caching techniques [4]. The problem is compounded by the necessity for real-time guarantees in a VOD server.

In a parallel I/O system, idle disks can be used to prefetch video blocks concurrently with demand I/Os from other disks. These prefetched blocks are held in the buffer until their deadlines, when they can be transferred to the client. However, deciding which blocks to prefetch is a non-trivial task. Blocks that are prefetched long before their deadline occupy buffer space wastefully, reducing the effective buffer size, and decreasing the number of clients that the server can handle; on the other hand delaying a prefetch potentially wastes disk bandwidth, causing blocks downstream to miss their deadlines and violating QoS guarantees.

We present a framework for real-time parallel I/O. In Section 2, we present our parallel I/O model. A real-time, parallel-disk scheduling algorithm RT-OPT that maximizes the number of clients that can be supported, is presented in Section 3. For any fixed disk-head scheduling policy [6], we show that RT-OPT is optimal in that

[†]Supported in part by the National Science Foundation under grant CCR-9704562, a grant from the Schlumberger Foundation, and a Graduate Research Fellowship from Texas Instruments.
To appear in Proceedings of CSI'99

if in its schedule, any block misses its deadline then there *does not exist any schedule* in which all blocks can meet their deadlines. In Section 4 we present simulation results for RT-OPT using traces of MPEG video streams, by empirically determining the number of clients that can be supported for different QoS and server parameters.

There have been a number of different approaches for storage and retrieval of real-time video data. Disk-head scheduling and single-disk data placement were addressed in [6, 8]. For the SPIFFI video-server, [3] proposed a real-time priority-based disk scheduling algorithm that partitions requests into priority classes based on the nearness of their deadlines. In [1, 2] admission and resource allocation algorithms for VBR-CDL video streams were studied in a single-disk system based on the concept of fetching in rounds. While simplifying the admission control algorithm, these techniques under-utilize the buffer and consequently can handle fewer clients. In [5], caching methods to reuse blocks of popular movies were proposed and analyzed. None of these works attempts to optimize the use of buffer in a multiple-disk situation among multiple disks.

2 Real-time Model

We consider a server-network-client architecture of a video server. The server consists of an array of D parallel disks and a main memory of size M blocks. A block is the unit of access from the disks. Video streams are VBR encoded and stored in CDL format so that blocks are of fixed size, and therefore the number of frames per block is variable. Our framework places no restriction on the placement of blocks on disks. The clients are assumed to have buffers to hold the current block being played back, and to mask network delays.

The temporal evolution of each block is modeled by a *timeline*, and indicates the state of the block between the start of its I/O and the end of its playback. The timeline consists of disk I/O time, time spent in the server buffer, network transmission delay, time spent in the client buffer, processing delay at client, and playback. The playback deadline for each block, is fixed by the frame rate and encoding of the previous blocks. The decompression time is also fixed and depends on the encoding of that block. The network transmission time is assumed to be known for every client. The disk I/O time is determined by the disk-head scheduling policy and disk characteristics. All these components together place a deadline on the latest time an I/O for a block can be initiated by the disk system.

From the server's viewpoint the model can be simplified by incorporating the network delay and client-side decompression and processing into the playback interval. Further to stress the server we assume minimal client features: a client only buffers the current block. This makes the client-buffer residence time zero.

The input to the scheduling algorithm is a string of blocks B_i, each with its associated disk access time L_i (IO End - IO Start), and deadline D_i, the latest time at which the block can complete its I/O. A block for which IO End is less than D_i stays in the server buffer till D_i at which time it is transferred to the client. The scheduling algorithm then determines the time at which each block should begin its I/O.

For illustration consider an intuitive scheduling algorithm GREED-EDF. GREED-EDF, aggressively keeps the disks busy, fetching in the order of deadlines. The blocks that cannot meet their deadline are dropped. Consider an I/O system with 3 disks and a buffer of 2 blocks. Let blocks a, b, and c with deadlines 1, 3, and 4 respectively, be requested from disks 1, 2, and 3 respectively. Let the I/O time for blocks a and b be 1 each while that of c be 4. Since the deadline of b (3) is earlier than that of c (4), GREED-EDF fetches blocks a and b initially. Thus an I/O for c cannot start before time 1, which causes it to miss its deadline. On the other hand we could initiate an I/O for c at time 0 and start fetching b only at time 1 which can allow all deadlines to be met. This schedule is constructed by RT-OPT.

3 Algorithm RT-OPT

Algorithm RT-OPT generates a schedule for a set of blocks, given their disk access times and deadlines. The access times are obtained by assuming a particular disk scheduling policy for each disk. Intuitively, RT-OPT operates as follows. The blocks are first tentatively scheduled so that each block's IO End time coincides with its deadline. Next RT-OPT iterates through the set of blocks in order to resolve conflicts. The first type of conflict occurs due to overlapping disk access intervals on a single disk. Since only one I/O can be in progress from a single disk at any time, the disk access interval of one of these blocks must be moved earlier in time to remove the overlap. The chosen order of access from the disk determines which will stay and which will be prefetched. A block is pushed back the minimal amount, so that

Algorithm RT-OPT

for d=1 to Number of disks /* Initialization */
    for b = 1 to last block on disk d
        Assign $\text{StartIO}(b) \leftarrow \text{StartPB}(b) - \text{IOTime}(b)$ and $\text{EndIO}(b) \leftarrow \text{StartPB}(b)$
        call $\text{PushDisk}(d, b)$
At any time t
    Fetch $F(t)$ blocks from $E(t)$ giving preference to blocks with earlier I/O Start time

Procedure PushDisk(disk d, block b)
if $b \neq 1$ and $\text{EndIO}(b-1) > \text{StartIO}(b)$ then
    Assign $\text{EndIO}(b-1) \leftarrow \text{StartIO}(b)$ and $\text{StartIO}(b-1) \leftarrow \text{StartIO}(b) - \text{IOTime}(b-1)$
    call $\text{PushDisk}(d, b-1)$

Figure 1: Algorithm RT-OPT

its IO End coincides with the IO Start of the next block to be fetched from that disk. Finally RT-OPT checks for buffer violations in the schedule. At any time the total number of blocks being fetched or which are in the server buffer cannot exceed M, the size of the server buffer. The blocks are scanned in increasing order of time, and blocks whose I/O initiation would violate the buffer constraint are dropped.

The pseudo-code for algorithm RT-OPT is shown in Figure 1. In the description, StartPB(b) denotes the playback start time PB Start for block b; similarly StartIO(b) and EndIO(b) denote IO Start and IO End times for block b. At any time t let $I(t)$ represent the set of idle disks, and $F(t) = \min\{N(t), |I(t)|\}$, where $N(t)$ is the number of free blocks in the server buffer. Let $E(t)$ be a set of blocks, from idle disks, with the earliest I/O start time among all blocks from that disk satisfying $t + \text{IOTime}(B_i) \leq \text{StartPB}(B_i)$.

RT-OPT's policy of fetching blocks as late as possible minimizes the buffer occupancy of each block and consequently the probability of buffer overflow for a given buffer size. Using this we can formally prove that RT-OPT is optimal in the sense that if it drops some blocks then every other algorithm must also drop at least one block for the same data placement, buffer size and single disk-head scheduling policy. As an admission control method RT-OPT can determine whether a client can be admitted by checking if a schedule exists in which each block meets its deadline.

4 Simulations

The performance of RT-OPT was evaluated using a trace-driven simulator based on our real-time model. We used public-domain MPEG compressed video traces described in [7]. The frame rate for these traces is 24 fps and each trace contains 40000 frames. Using data on the coded frame sizes for each of the video sequences, we created CDL blocks of 64 Kbytes from 5 representative streams: Bond, Terminator, Lambs, Dino and Starwars. The blocks were then striped across the disks in a round-robin fashion. To model the probabilistic access structure of clients we used the Zipf distribution, with a skewness factor of 0.271. Clients were assumed to access the chosen clip from a random starting frame; wrapping around when the end of a stream is reached. The results reported correspond to accesses to 50000 blocks. We assumed EDF scheduling at individual disks, and I/O times uniformly distributed between 16 and 44ms. In all cases averaged results over several Monte Carlo trials are reported.

We first consider the case in which all blocks are required to meet their deadlines and measure the number of clients that can be supported. Figure 2 shows the the number of clients serviced versus buffer size for 64 disks. RT-OPT does substantially better than GREED-EDF over a wode range of buffer sizes. Similar trends were observed with smaller number of disks.

The scalability of the algorithms RT-OPT and GREED-EDF with number of disks is illustrated in Figure 3. The scalability of RT-OPT is better than that of GREED-EDF, and close to linear at all buffer sizes: M = 32, 64, 128 and 256. For brevity only results with M =256 are shown in the figure. The performance of GREED-EDF saturates quickly; that is, GREED-EDF requires a much larger amount of buffer to exploit the available disk bandwidth. In contrast RT-OPT supports a substantially larger number of clients

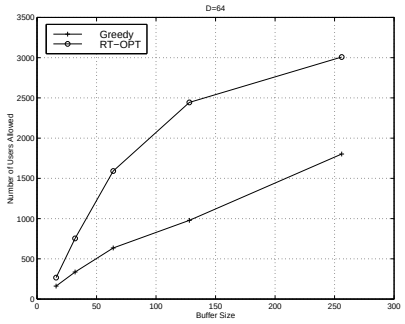

Figure 2: Clients Serviced vs. Buffer Size

with modest buffer requirements.

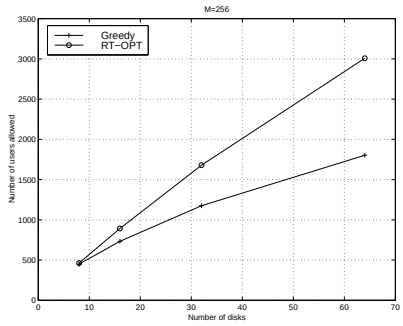

Figure 3: Scalability with number of disks

Finally in Figure 4 the two algorithms are compared by determining the number of clients that can be supported with a given block dropping probability. The maximum drop rate permitted was fixed at 2%, and M was fixed at 256. For brevity only results for 64 disks are presented, the trend is similar for 8, 16 or 32 disks also. RT-OPT can support substantially more number of clients at a given drop probability. The improvement was noticed to be greatest with larger number of disks when the need for efficient buffer management becomes more important.

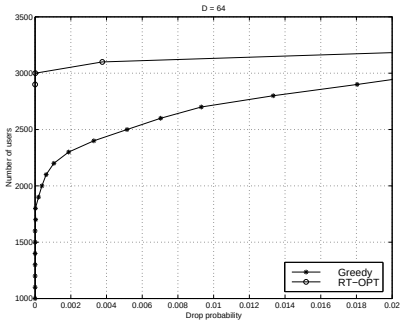

Figure 4: Clients Serviced vs. Drop Probability

5 Conclusions

We introduced a framework for incorporating real-time constraints on block accesses in a parallel I/O system. The framework was used to design prefetching and scheduling algorithms for real-time playback of multiple video streams in a multiple-disk, parallel-I/O-based video server.

We introduced a scheduling algorithm RT-OPT for prefetching blocks into the server buffer. RT-OPT is shown to be optimal, in the sense that if it drops a block then there exists no schedule in which all blocks meet their deadline. Using traces from MPEG compressed video clips we empirically observed the performance of RT-OPT. By dynamically multiplexing the buffer among different clients and disks optimally, RT-OPT was shown to be highly scalable with the number of disks with only modest buffer requirements. The number of clients supported was shown to be superior to intuitive but suboptimal algorithms like GREED-EDF that attempt to keep the disks busy and fetch in order of deadlines.

References

[1]E. W. Biersack, F. Thiesse, and C. Bernhardt. Constant Data Length Retrieval for Video Servers with Variable Bit Rate Streams. In *Proc. of Conf. on Multimedia*, pp 151–155, 1996.

[2]Ed Chang and A. Zakhor. Admission Control and Data Placement for VBR Video Servers. In *Proc. of Intl. Conf. on Image Processing*, pp 316–329, 1994.

[3]C. .S Freedman and D. .J DeWitt. The SPIFFI Scalable Video-on-Demand System. In *Proc. of SIGMOD'95*, pp 352–363, 1995.

[4]M. Kallahalla and P. Varman. Optimal Read-Once Parallel Disk Scheduling. In *Proc. of ACM Wkshp. on I/O in Parallel and Distributed Systems*, 1999.

[5]B. Ozden R. Rastogi and A. Silberschatz. Demand Paging for Video-on-Demand Servers. In *Intl. Conf. on Multimedia Computing and Systems*, pp 264–272, 1995.

[6]A. L. N. Reddy and J. C. Wyllie. I/O issues in a Multimedia System. *Computer*, 27(3):69–74, 1994.

[7]O. Rose. Statistical Properties of MPEG Video Traffic and their Impact on Traffic Modelling in ATM Systems. Research Report Series 101, University of Wurzburg, Institute of Computer Science, February 1995. ftp://ftp-info3.informatik.uni-wuerzburg.de.

[8]P. S. Yu, M-S. Chen, and D. D. Kandlur. Grouped Sweeping Scheduling for DASD-based Multimedia Storage Management. *Multimedia Systems*, 1(1):99–109, 1993.