

TRANSMITTING VARIABLE-BIT-RATE VIDEOS ON CLUSTERED VOD SYSTEMS

Chow-Sing Lin

School of Elec. Engineering and Computing
University of Central Florida

Min-You Wu and Wei Shu

Dept. of Elec. and Computer Engineering
University of New Mexico

ABSTRACT

Due to the burstiness, scheduling VBR streams and yet achieving high resources utilization on a clustered video server has been considered a difficult problem. In this paper, we propose a simple but effective VBR scheduling algorithm for generating conflict-free network transmission schedules on clustered VoD systems. Experimental results show that this novel scheduling algorithm can achieve near 100% network utilization with a negligible block miss rate.

1. INTRODUCTION

The motivation of developing efficient scheduling algorithms for delivering Variable-Bit-Rate (VBR) encoded video streams is that given the same perceived video quality, Constant-Bit-Rate (CBR) encoding produces a data rate significantly higher than the average rate of the corresponding VBR encoding for action movies [2]. Therefore, scheduling algorithms for VBR encoded video that achieve near 100 percent link bandwidth utilization, while keeping losses at a negligible level, can allow for significantly more video connections than CBR-encoded video. However, it has been observed by several researchers [4, 3] that compressed VBR video traffic, such as MPEG, typically exhibits burstiness over multiple time scales. Such an unpredictable burstiness complicates the design of efficient real-time mechanisms capable of achieving high resource utilization.

To address this problem, techniques for reducing rate variability and hence increasing the efficiency of VBR video delivery are intensively studied in many research works [8, 4, 6]. Among these techniques, in this paper we are only interested in the one termed smoothing by *prefetching* (or *work-ahead* in [8]) where the underutilized bandwidth during transmission of low-rate portions is utilized to prefetch high-rate portions. Given a VBR video stream, it can be partitioned into a sequence of Constant-Time-Length (CTL) video blocks. According to its rate variability, each video block may have different length. Then, the smoothing effect can be performed in a manner of prefetching lengthy video blocks as soon as possible. In this paper, we adaptively apply *Least-Laxity-First (LLF)* policy to select video blocks for delivery, in which longer video blocks have the tendency to possess higher scheduling priorities.

In addition to smoothing VBR streams for reducing rate variability, a stream scheduling algorithm must solve the network conflict problem. In this paper, we propose a simple but effective greedy approach to generate a schedule which guarantees no conflict for interconnection network transmission.

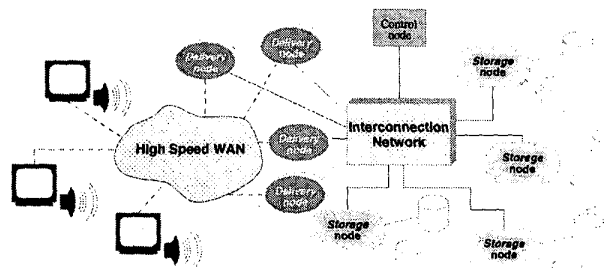


Figure 1: Diagram of a Clustered Server.

2. OVERVIEW OF SYSTEM ARCHITECTURE

The prototype of our clustered video server, *Odyssey* [5], is a cluster of PCs connected by a Fast Ethernet switch, as shown in Figure 1. There are three kinds of nodes in *Odyssey* which are *storage node*, *delivery node* and *control node*. The *storage node* is responsible for storing video data, retrieving requested data blocks and sending them to delivery nodes. In addition, partitioned video blocks are wide striped among storage nodes in a round-robin fashion to balance the workload. Each storage node deals with its own disk scheduling algorithm to provide enough bandwidth. In *Odyssey*, the SCAN algorithm is used for disk scheduling. A ping-pong buffer for each stream is applied to decouple system modules of disk access and network transmission so that they can be considered separately. The *delivery node* is responsible for taking requests from clients then forwarding them to the control node. Video blocks from storage nodes are buffered in delivery nodes, where they are re-sequenced if necessary, and then sent to clients. The *control node* provides the functionality of admission control, stream scheduling and server-level synchronization. *Odyssey* also employs a *flat* architecture where a logical storage node and a logical delivery node are mapped to a single physical node, called *processing node*.

3. VBR STREAM SCHEDULING

3.1. Stream Partition

Given a VBR stream with the frame rate *BaseRate*, it can be partitioned as a sequence of CTL video blocks consisting of a fixed number of frames f . Therefore, the cycle time is defined as $t_c = f/BaseRate$. Each block is then further fragmented as a number of fixed size subblocks. To increase the efficiency of data transmission, the size of a subblock, s , is specified as a multiple of the size of a packet. In practice, the optimal value of s for sustaining high network utilization can be measured from experiments. Given a video block with the size of *BlkSize*, it is fragmented as $\lceil BlkSize/s \rceil$

subblocks. Also, assume that the network bandwidth is $Netbw$, then the time for transmitting a subblock is $t_s = s/Netbw$.

3.2. Block Scheduling

To ensure (Quality-of-Service) QoS, video blocks must be received in time to guarantee continuous display. In general, a deadline is associated with a requested video block to express the urgency of delivery. A video block with the shortest deadline should be selected to be retrieved and transmitted first in order to meet its real time constraint. This is the algorithm so called *Earliest-Deadline-First (EDF)*. This algorithm works fine in video servers with Constant-Data-Length (CDL) data partition. However, in video servers with CTL data partition where the size of a video block in a video stream is varied from time to time, a deadline cannot truly express the urgency of a block. The length of a video block now has to be taken into consideration in computing its deadline. Here, a term called *laxity* is used to express the urgency of a video block. Suppose a video block currently contains k subblocks and its deadline is d , then the laxity at the current time, $CurTime$, for the video block is computed as,

$$Laxity = d - CurTime - k \times t_s$$

Figure 2 shows laxities for different sizes of video blocks. The laxity explicitly specifies that how much longer a video block can be delayed to be transmitted without violating its real time constraint. A negative value of laxity indicates that the video block has missed the deadline to transmit.

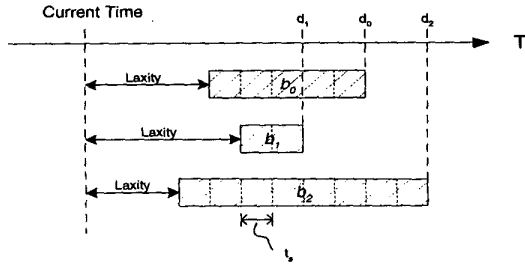


Figure 2: laxities for different sizes of video blocks

To select a subblock among requested video blocks to schedule, *Least-Laxity-First (LLF)* policy is applied. For example, in Figure 2, even though b_2 has the latest deadline, due to its longer length, it has the least laxity. Therefore, it will be scheduled first.

3.3. Generating Conflict-Free Schedules

To multiplex data transmission, a time cycle is partitioned as m time slots, where $m = \lfloor t_c/t_s \rfloor$. Assume that a clustered video server has N processing nodes, then a schedule of network transmission for one time cycle can be specified as a $N \times m$ matrix. The rows and columns in the schedule represents destined delivery nodes and time slots of requests, respectively. Each entry in the schedule consists of the information of request id, subblock number and storage node number. Different from stand alone video servers, the network transmission in clustered video servers must eliminate input and output port contentions to guarantee video blocks to be delivered in time. That is, two requirements have to be complied when generating a conflict-free schedule, which are at each time slot no two requests retrieve video blocks from the same storage

node and no two storage nodes transmit video blocks to the same delivery node.

To schedule a time slot, first, the subblock with the least laxity is selected from current requested video blocks. Then, the subblock with the least laxity in a different delivery node and a different storage node is selected. If such a subblock cannot be found in video blocks of the current time cycle, then we look at video blocks of the following time cycle until one is found. This process is repeated until a time slot is filled. In order to limit the searching depth for selecting a subblock, a *scheduling window* materialized as a set of priority queues with size w is applied. This w explicitly specifies the number of video blocks for a request can participate in the selection of subblocks. After a time slot is scheduled, the current time and laxities of video blocks in the scheduling window are updated before scheduling the next time slot. Furthermore, in case an appropriate subblock cannot be found in the scheduling window, the corresponding entry of the time slot in the schedule, termed *subslot*, therefore, is left unscheduled. Figure 3 shows an example of selection of subblocks for a time slot with $N = 3$.

Laxity updating

Assume that $Laxity_{old}$ and $Laxity_{new}$ represents laxities of a video block before and after a time slot has been scheduled. Then, updating the laxity of a video block can be considered in two folds. First, if one of subblocks in a video block has been scheduled, then the laxity of the video block is kept the same without updating, i.e., $Laxity_{new} = Laxity_{old}$. On the other hand, for updating the laxity of a video block with no subblock scheduled, two situations need to be considered. If its laxity is not equal to zero, then the laxity should be shorten by a time slot, i.e., $Laxity_{new} = Laxity_{old} - t_s$; Or, if its laxity equals to zero, i.e., it becomes negative after updating, then a subblock is dropped and the laxity of the video block again is set to be zero. After updating, if no more subblock in a video block is left for scheduling, the next video block corresponding to that request is added to the scheduling window.

Selecting a victim subblock to drop

When a video block misses its deadline, a victim subblock in the block has to be dropped. In order to keep the best possible service, we should select the one which has the least impact on the quality of video. That is, the data layout of a video block has to be taken into consideration. Dropping subblocks is similar to rate reduction in scalable video servers where the data containing fundamental information of a video block is always stored at the front portion and transmitted first. For example, a video block can be partitioned into a *low resolution* component and a *high resolution* component which are resided at the front portion and the back portion of a video block, respectively. Such a data partitioning in a transform block can be done by dividing frequency domain coefficients between the two components similar to the approach proposed in [9]. The other approach is to layout a video block by grouping frames with the same dependency to support different rates for MPEG video streams [1]. For example, assume a video block contains a standard GOP pattern of $IB_1B_2P_1B_3B_4P_2B_5B_6P_3B_7B_8P_4B_9B_{10}$. By grouping frames with the same dependency, the storage layout becomes $IP_1P_2P_3P_4B_1B_2B_3B_4B_5B_6B_7B_8B_9B_{10}$.

Both data layouts imply that the selection of a victim subblock should start from *the last one* such that the best possible quality presentation can be achieved. This approach is applied in the paper.

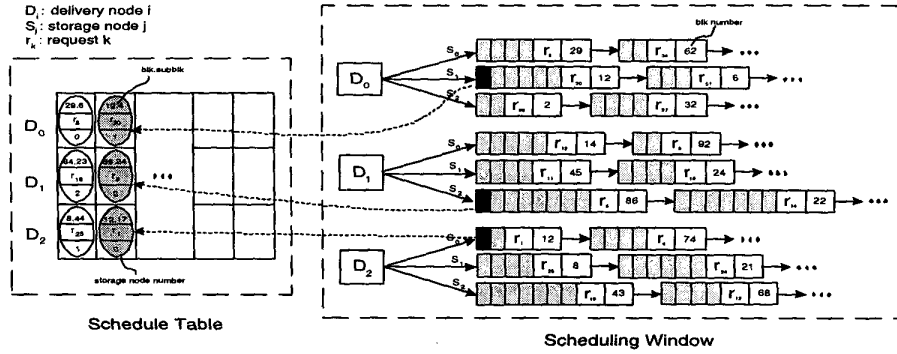


Figure 3: The selection of subblocks for a time slot with $N = 3$

4. PERFORMANCE EVALUATION

4.1. Traces and Data Partition

To create a fair testing data set in our simulation, we use four separate MPEG traces which give the number of bits in each frame from University of Wuerzburg [7]. These traces are named *Lambs*, *Mr.Bean*, *Simpsons* and *Talk* with the order of peak/mean rate from high to low. Each of the video clips was compressed with the GOP pattern IBBPBBPBBPBB at a frame rate of 24 f/s. There are 40000 frames in each trace, corresponding to about 28 minutes. The traces obtained were of fairly low bit rate. To simulate MPEG-1 streams, they were magnified to 1.375 Mb/s based their average bit rates while relative weights and burstiness of the frames are still preserved. These traces are then partitioned as a sequence of video blocks, each of that contains two GOPs (24 frames) which represents one second display time. With 40000 frames, each trace is partitioned as $\lfloor 40000/24 \rfloor = 1666$ blocks. Figure 4 shows the block size trace of *Lambs*.

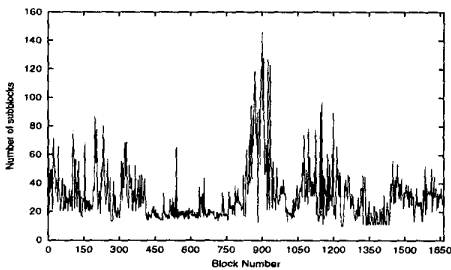


Figure 4: Block size trace of *Lambs*.

The simulated network environment is Fast Ethernet with TCP whose MTU is known as 1500 bytes. In TCP, if excluding the first 40 bytes for the protocol header, only 1460 bytes left in a packet are available for data transmission. Since our preliminary measurement from a Fast Ethernet switch shows that the optimal size of a subblock is between 2 to 16 packets, without losing generality, in the experiment each subblock contains only 4 packets. That is, $s = 4 \times 1460 = 5840$ bytes. The statistics of those MPEG block size traces in terms of the number of subblocks are shown in Table 1. We note that after the partition, the average bit rate of these video streams is increased from 1.375 Mb/s to 1.397 Mb/s due to the internal fragmentation.

Table 1: The Statistics of MPEG Block Size Traces

Trace	Average	Max	Min	Peak/Mean
<i>Lambs</i>	31.359	146	10	4.656
<i>Mr.Bean</i>	31.349	122	9	3.892
<i>Simpsons</i>	31.356	102	5	3.252
<i>Talk</i>	31.359	81	15	2.582

4.2. Experimental Results

The set up of the simulation is described as follows. Without losing generality, we assume that the sustained link bandwidth for Fast Ethernet is 80 Mb/s. Then each link can statistically deliver up to $\lfloor 80/1.397 \rfloor = 57$ MPEG-1 video streams. In the simulation, 57 requests are assumed to arrive in each delivery node simultaneously. Each request randomly picks one of those four movies. The starting block number and storage node number are independent and uniform distributed over $[0, 1665]$ and $[0, N-1]$, respectively. Each request is mandatory to watch the full length of a movie regardless of where a movie being started. If a starting block number for a request is k , then the last requested block number shall be $(k + 1665) \bmod 1666$. After the 1666th video block is scheduled, the next requested block is wrapped around to the first block of the movie.

The schedulability of the algorithm can be measured in terms of *miss rate* which is calculated as the total number of missed subblocks divided by the total number of subblocks requested by 57 N clients. During the simulation, no flow control is applied so that infinite buffer space in a client is assumed.

Figure 5 shows the impact of various sizes of scheduling window on miss rate over N with no pre-buffering ($p = 0$). As it shown, miss rates drop dramatically after increasing w from 1 to 2. Miss rates are not significantly improved when w is greater than 2.

Figure 6 shows the network utilization over N with $p = 0$. During the simulation, the total number of unscheduled subslots is recorded as ns . Then, the network utilization be calculated as, $1 - ns/(N * m * 1666)$. The resultant curve shows near 100% network utilization over N .

Pre-buffering video data before actual playback may further reduce miss rate. With pre-buffering video data of p time cycles, the laxity of a subsequent video block can be relaxed to $Laxity + p * t_c$. Figure 7 shows miss rate over various length of pre-buffering time

Table 2: The distribution of dropped subblocks per late block with $N = 4$.

Dropped Subblocks	1	2	3	4	5	6	7	8	9	10
Total Percentage	33.7%	31.8%	19.3%	9.6%	2.7%	0.1%	0.6%	0.5%	0.7%	1.0%
Accum. Percentage	33.7%	65.5%	84.8%	94.4%	97.1%	97.2%	97.8%	98.3%	99.0%	100%

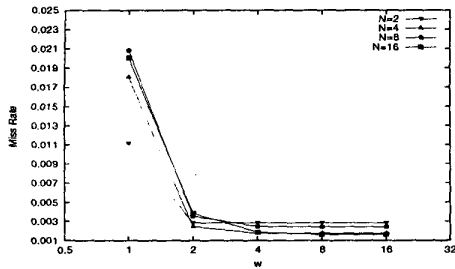


Figure 5: Miss Rates for various sizes of scheduling window over N .

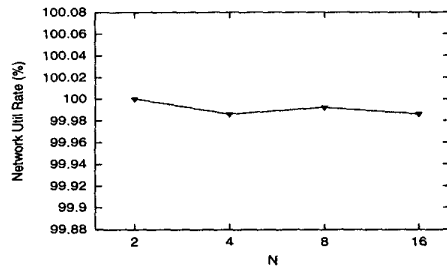


Figure 6: Network utilization over N .

cycles with $N = 4$. As it shown, the miss rate drops significantly with pre-buffering one and two time cycles. With pre-buffering video blocks of 6 time cycles, no subblock misses its deadline.

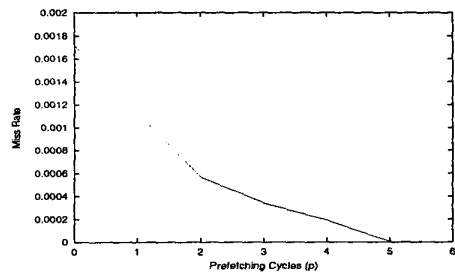


Figure 7: Miss rate over various length of pre-buffering time cycles with $N = 4$.

Table 2 shows the distribution of dropped subblocks per late block with $N = 4$. As it shown, about 65.5% of late video blocks drop one or two subblocks, and also less than 3% of them drop 5 or more subblocks. Most of victim subblocks contain only high resolution component or B frames so that the perception of video quality is expected not to be significantly affected.

5. CONCLUSION

The experimental results show the superior schedulability of the proposed scheduling algorithm for VBR streams on clustered VoD systems in terms of high network utilization and low miss rate. The distribution of dropped subblocks further implies the negligible degradation on the video presentation. With a small amount of pre-buffering, block miss can be entirely eliminated.

6. REFERENCES

- [1] E. Chang and A. Zakhor, *Variable bit rate MPEG video storage on parallel disk array*, 1st Int'l Workshop on Community Networking: Integrated Multimedia Services to the Home (IEEE Press, ed.), 1994, pp. 127–137.
- [2] I. Dalgic and F. A. Tobagi, *Characteristic of quality and traffic for various video encoding schemes and various encoder control schemes*, Csltr-96-701, Stanford University, Dept of Elec. Eng. and Comp. Sci., August 1996.
- [3] Mark W. Garrett and Walter Willinger, *Analysis, modeling and generation of self-similar VBR video traffic*, ACM SIGCOMM, 1994, pp. 269–280.
- [4] Matthias Grossglauser, Srinivasan Keshav, and David N. C. Tse, *RCBR: A simple and efficient service for multiple time-scale traffic*, IEEE/AVM Trans. on Networking 5 (1997), no. 6, 741–755.
- [5] Chow-Sing Lin, Wei Shu, and Min-You Wu, *Performance study of synchronization schemes on parallel CBR video servers*, ACM Multimedia (Orlando), Nov. 1999.
- [6] M. Reisslein and K. W. Ross, *High-performance prefetching protocols for VBR prerecorded video*, IEEE Network (Nov./Dec. 1998), 45–55.
- [7] Oliver Rose, *Statistical properties of MPEG video traffic and their impact on traffic modeling in ATM systems*, Proceedings of the 20th Annual conference on Local Computer Networks, 1995, pp. 397–406.
- [8] J. D. Salehi, Z. L. Zhang, J. Kurose, and D. Towsley, *Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing*, IEEE/ACM Trans. on Networking 6 (1998), no. 4, 397–410.
- [9] P. J. Shenoy and H. M. Vin, *Efficient support for scan operations in video servers*, The Third ACM conference on Multimedia, November 1995.