1

IMPROVING THE INTERACTIVE RESPONSIVENESS IN A VIDEO SERVER

A. L. Narasimha Reddy

Dept. of Elec. Engg.
214 Zachry
Texas A & M University
College Station, TX 77843-3128
reddy@ee.tamu.edu

ABSTRACT

In this paper, we will address the problem of designing an interactive video server employing multiprocessors. An interactive video server needs to provide quick response to user's requests. When the video stream is a sequence of short video clips and if the order in which these clips are played out is determined by the user, the video clips need to be started up fairly quickly to give an impression of seamless or near-seamless delivery. We will address this problem of providing quick turnaround response to user's requests while guaranteeing smooth play out of already scheduled streams.

1 INTRODUCTION

In this paper, an interactive video server is a system where the play out of a video stream is dependent on the nature of the input from the user. Interactive video servers are envisioned to be employed in a number of areas such as video training, travel planning, real-estate business and interactive movies. These applications typically involve playing a series of predetermined video clips. The order of play out of these clips is dependent on the input from the user. For example, in real-estate business, the application can put the user in a simulated car to let the user observe the neighborhood of a house so that the customer may have a feel of the surroundings before visiting the house. Similarly, travel planning can have a series of video clips of sights in a city and the user can plan on his/her tour by observing these video clips according to the user's travel plan.

In these interactive systems, it is necessary to move from one video clip to the next video clip selected by the user seamlessly or near-seamlessly. Even though the video stream is controlled by the user in these applications, it is not expected that the user requires interactivity at the level of a video game. The goal in this paper is to design a system with sub-second response times i.e., the next video clip is played out on the user's monitor in less than a second from his selection of the next clip. The level of service required in such a system falls somewhere in the middle of the spectrum from what is expected in a video-on-demand service and an interactive video-game service. In contrast, in a video-on-demand server, the user may be willing to tolerate a few seconds/minutes delay for the startup of a movie stream and in an interactive video game service the user requires frequent service of low latency turn-around response.

2 THE PROBLEM

A number of nodes in the video server system act as *storage nodes*. Storage nodes are responsible for storing video data either in memory, disk, tape or some other medium and delivering the required I/O bandwidth to this data. The system also has *network nodes*. These network nodes are responsible for requesting appropriate data blocks from storage nodes and routing them to the customers. Both these functions can reside on the same multiprocessor node, i.e., a node can be a storage node, or a network node or both at the same time. Each request stream would originate at one of the several network nodes in the system and this network node would be responsible for obtaining the required data for this stream from the various storage nodes in the system. For the rest of the paper, we will assume that every node in the system is both a storage node and a network node at the same time, i.e., a combination node. We will assume a configuration as shown in Fig. 1. Each node in the system has the same number of disks and the disk ids are distributed in a round robin fashion across the nodes in the system.

To obtain high I/O bandwidth, data has to be striped across a number of nodes. If a movie is completely stored on a single disk, the number of streams of that movie will be limited by the disk bandwidth. As shown earlier by [1], a 3.5" 2-GB IBM disk can support up to 20 streams. A popular movie may receive more than 20 requests over the length of the playback time of that movie. To

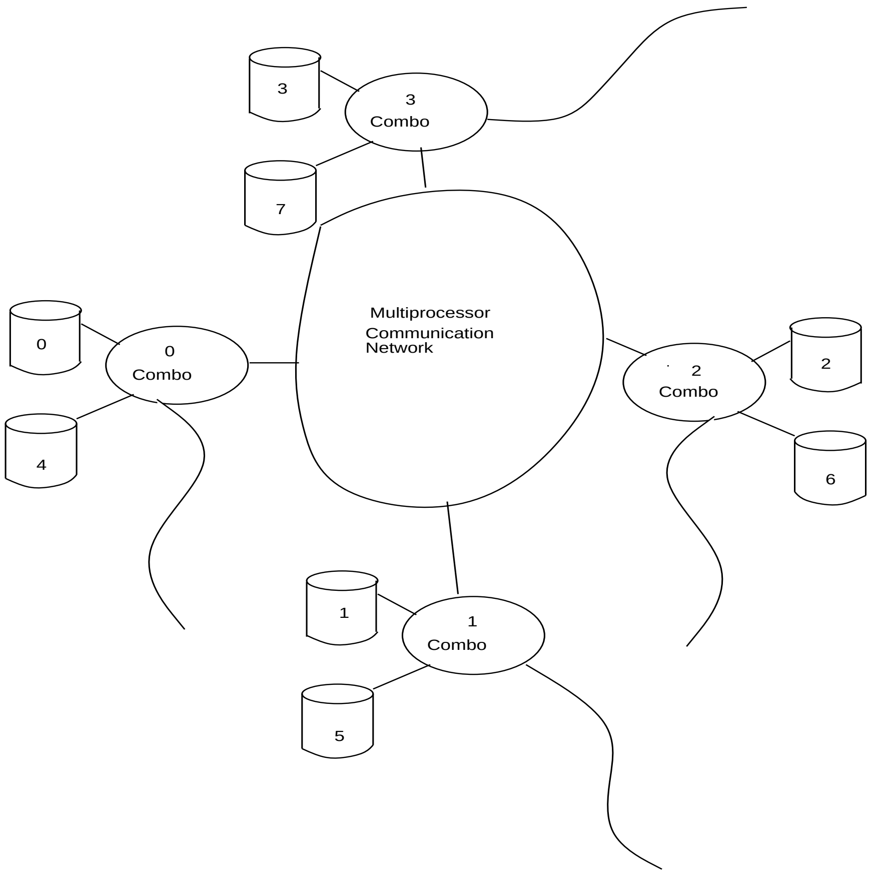

Fig. 1. System model of a multiprocessor video server.

enable serving a larger number of streams of a single movie, each movie has to be striped across a number of nodes. As we increase the number of nodes for striping, we increase the bandwidth for a single movie. If all the movies are striped across all the nodes, we also improve the load balancing across the system since every node in the system has to participate in providing access to each movie. Hence, we assume that all the movies are striped across all the nodes in the system. Even when the movies are stored in semiconductor memory, the required communication and memory bandwidths may require that the movie be striped across the memories of different nodes of the system. The unit of striping across the storage nodes is called a block. In our earlier studies on disk scheduling [1], we found that 256 Kbytes is a suitable disk block size for delivering high real-time bandwidth from the disk subsystem.

When the video-on-demand server is operational, requests for starting new movie streams arrive at random times and already scheduled streams complete service at random times. For serial playback of a movie, once the first block of the movie is scheduled, the rest of the movie can be delivered without any interruption [2]. In that case, we are concerned with the latency to deliver the fist block of the movie to the customer's desktop. It is likely that a few seconds delay to start a movie is acceptable to the customer. In this paper, we address a more difficult problem of ensuring short latencies between different sequences of video in an interactive video stream. In an interactive video, the sequence of video playback can change based on the customer's input. When the video playback sequence is changed, it is necessary that the following data block be delivered to the customer within a very short time to preserve the feel of interactivity. In this paper, we address this problem of achieving short, specifically sub-second, response times for these requests.

Recent work [3, 1, 4, 5, 6] has looked at disk scheduling in a video server. File systems for handling continuous media have been proposed [7, 8, 9, 10]. Pause/resume and reverse/forward type interactivity is addressed in some of the recent work [11, 12, 13]. Most of this work on interactivity is concerned about the impact of changes in bandwidth due to pause/resume and reverse/forward operations. Our work here addresses the problem of improving the latency for interactive requests in an environment where bandwidth requirements are constant.

3 STREAM SCHEDULING

We will assume that time is divided into a number of 'slots'. The length of a slot is roughly equal to the average time taken to transfer a block of movie from the disk to a buffer on the storage node.

The time taken for the playback of a movie block is called a frame. The length of the frame depends on the block size and the stream rate. For a block size of 256 Kbytes and a stream rate of 200 Kbytes/sec, the length of a frame equals $256/200 = 1.28$ seconds. We will assume that a basic stream rate of MPEG-1 quality at 1.5Mbits/sec is supported by the system. When higher stream rates are required, multiple slots are assigned within a frame to achieve the required delivery rate for that stream. But it is assumed that all the required rates are supported by transferring movie data in a standard block size (which is also the striping size).

For a given system, the block size is chosen first. For a given basic stream rate, the frame length is then determined. Slot width is then approximated by dividing the block size by the average achievable data rate from the disk. This value is adjusted for variations in seek and latency delays. Also, we require that frame length be an integer multiple of the slot width. From here, we will refer to the frame length in terms of number of slots per frame 'F'.

Now, the complete schedule of movies in the system can be shown by a table as shown in Fig. 2. The example system has 4 nodes, 0, 1, 2, and 3 and contains 5 movies A, B, C, D, and E. The distribution of movies A, B, C, D, E across the disks 0, 1, 2, and 3 is shown in Fig. 2 (a). For example, movie E is distributed cyclically across disks in the order of 2, 1, 0, and 3. For this example, we will assume that the frame length $F = 3$. Now, if movie needs to be scheduled at node 0, data blocks need to be read from disks 2, 1, 0 and 3 to node 0 in different slots. This is shown in Fig. 2(b) where the movie is started in slot 0. Fig. 2(c) shows a complete schedule of 4 requests for movies E, C, B, and E that arrived in that order at nodes 0, 1, 2, 3 respectively. Each row in the schedule shows the blocks received by a node in different time slots. The entries in the table indicate the movie and the id of the disk providing service for that request. Each column should not have a disk listed more than once since that would constitute a conflict at the disk. A movie stream has its requests listed horizontally in a row. The blocks of a single stream are always separated by F slots, in this case $F = 3$. Node 0 schedules the movie to start in time slot 0. But node 1 cannot start its movie stream in slot 0 as it conflicts with node 0 for requesting a block from the same disk 2. Node 2 can also schedule its movie in slot 1. Node 3 can only schedule its movie in slot 2. Each request is scheduled in the earliest available slot. The movie stream can be started in any column in the table as long as its blocks do not conflict with the already scheduled blocks. The schedule table is wrapped around i.e., Slot 0 is the slot immediately after Slot 11. For example, if another request arrives for movie E at node 2, we can start that request in time Slot 3, and schedule the requests in a wrap-around fashion in time Slots 6, 9, and 0 without any conflict at the disks. The schedule table has FN slots, where N is the number of disks in the system.

2(a). Movie distribution on disks:

Movie/Blocks	0	1	2	3
A	0	1	2	3
B	1	3	0	2
C	2	0	3	1
D	3	2	1	0
E	2	1	0	3

2 (b). Schedule for movie E:

Slot 0	1	2	3	4	5	6	7	8	9	10	11
E.2			E.1			E.0			E.3		

2(c). Complete schedule:

Req. node	Slot 0	1	2	3	4	5	6	7	8	9	10	11
0	E.2			E.1			E.0			E.3		
1		C.2			C.0			C.3			C.1	
2		B.1			B.3			B.0			B.2	
3			E.2			E.1			E.0			E.3

Fig. 2. An example movie schedule.

When the system is running to its capacity, each column would have an entry for each disk in the system. It is observed that the schedule has a permutation of disk ids in each slot. If we specify F such sets for the F slots in a frame (j = 1,2,...F), we would completely specify the schedule. If a movie stream is scheduled in slot j in a frame, then it is necessary to schedule the next block of that movie in slot j of the next frame (or in $(j + F)$ mod FN frame) as well. Once the movie distribution is given, the schedule of transfer in slot j from disk s_i automatically determines that the next block of this stream be transferred in the next frame from disk s_{i+1}, where s_{i+1} is the disk storing the next block of this movie. Hence, given a starting entry in the table (row, column specified), we can immediately tell what other entries are needed in the table. It is observed that the F slots in a frame are not necessarily correlated to each other, but there is a strong correlation between two successive frames of the schedule. It is also observed that the length of the table (FN) is equal to the number of streams that the whole system can support.

The notation introduced in this paper is very similar to the notation used by us earlier in solving the network scheduling problem in a multiprocessor video server [2]. The slot sizes used in the earlier paper were based on the communication requirements and the slot sizes used here are based on the disk service times. These two solutions need to be combined together for a complete solution to the scheduling problem in a multiprocessor video server.

We will assume that the movies are striped across all the disks in the system in a round-robin order. The starting disks can be different for different movies. This distribution is shown to be quite effective in [2]. This distribution results in (a) balanced load across the disks in the system, (b) regular communication patterns that simplify communication scheduling and (c) simpler scheduling of movie streams in the system. Moreover, this distribution simplifies the problem of movie scheduling to scheduling the first block of the movie stream [2].

In this paper, we classify requests into two categories: **urgent** and **non-urgent**. Urgent requests are those requests for which latency of response should be as short as possible. Requests to transition from one video clip to the next one during an already-in-progress stream can be termed an urgent request. Urgent requests typically have sub-second response time requirements. Non-urgent requests are requests to start a new movie stream or a new video sequence. Non-urgent requests can endure a few seconds delay. In this paper, we will consider different mixes of urgent and non-urgent requests in the request stream. In this paper, we propose and evaluate techniques to meet latency targets for urgent requests.

3.1 System Model

We assume double buffering at the customer's desktop or presentation device. With double buffering, while a block of data is being consumed, another block of data is being filled into the second buffer such that there is always a block of data waiting to be consumed when one block is finished. This helps in maintaining the seamless delivery of video data to the consumer.

For video-on-demand applications, once the first block of data is scheduled for service, we can predict the entire schedule of service until the movie is completely played out. This predictability makes it easier to provide a seamless delivery to the consumer since the blocks can be read ahead of time before they are needed. However, for more interactive applications described earlier such as training videos, the predictability of the sequence of blocks is occasionally destroyed by the user's input. At these times, when the user's response determines the next block of data to be played out, the seamless delivery guarantee is harder to guarantee and this is the problem that is being addressed in this paper.

This problem is somewhat similar to the problem of running the processor pipeline without any breaks when we encounter branch instructions in the instruction stream. The user's input may change the sequence of blocks to be read, similar to the way a branch instruction may change the sequence of instructions to be executed. The compilers typically use various strategies to predict the flow of instructions after a branch to keep the pipeline running with as few breaks as possible. Similar strategies can be applied in this context for predicting the flow of video sequences in an interactive video. The approach described here is somewhat different and can be applied in conjunction with such predictive strategies.

In our system, urgent requests and non-urgent requests are scheduled differently. For non-urgent requests, the first block of the movie is not played out until the second block of the movie is scheduled for service at the disk. Since we utilize a block size of 256 Kbytes, this may translate to a latency of 1.28 seconds for an MPEG-1 stream playing at the rate of 200 KB/sec. This latency is quite tolerable at the beginning of the movie. Moreover, since the blocks are presented to the user without any break, the user can observe the movie without any breaks in service once the movie stream is started. However, if the video sequence needs to be changed in the middle of the presentation of an interactive video, this latency would be intolerable and would appear as a glitch in service. For these urgent requests, we schedule the play out of video data immediately after the block is read out. This minimizes the latency of transition from one video sequence to the next.

Consider a disk with a deliverable bandwidth of say 4 MB/sec. A block of 256 Kbytes can be read in 64 msec. If the data is immediately played out at the customer's desktop, the observed latency is only about 64 msec and it is quite tolerable.

When data is distributed in a round-robin fashion, each video stream is assigned a slot of service at the disks at the correct time for each request block of that stream. This is guaranteed when we schedule the very fist request block of that stream. Urgent requests however change the video sequence and hence may change the next disk that has to service this video stream. During that time slot, another request may already have been scheduled at that disk. Hence, scheduling this block for an already scheduled stream requires special attention. This block cannot be scheduled as if this is the first block of a new stream because of the tight latency requirements for this block's service.

Let's define **latency target** as the target latency to be achieved by urgent requests. This latency target is specified in terms of slots. To allow for variations in service time due to seek and latency delays of the disk and the communication delays in getting the request block to the user, latency targets have to be a multiple number of slots. In this paper, we will consider latency targets of 5 slots and up, giving us minimum latencies in the range of 300 ms. A **frame** is defined earlier to be the number of slots required to play out a request block at the consumer's desktop. With a disk bandwidth of 4 MB/sec and a stream rate of 200 KB/sec, a frame is equal to $4/0.2 = 20$ slots. The latency for non-urgent requests is 20 slots after the first block of the video data is scheduled. The latency for non-urgent requests is at least 20 slots. We will show later from simulations that the latency can be much higher than this number due to finding the required disk busy for several slots after the request arrives.

3.2 Techniques for reducing the startup latency

Consider a non-urgent request that is already scheduled. The request is scheduled for service 'F' slots before it will be consumed at the customer's desktop. Let us call this time between the actual time of consumption and the time of scheduled service, **slack time**. Because of various delays at the disk, and in the communication medium, there is a **safe slack time** below which the needed block cannot be guaranteed to be available for consumption at the customer's desktop at the required time. If the safe slack time is $S < F$, then the scheduled non-urgent request can be safely scheduled later, delayed up to F-S slots, without violating the safe slack time period. Fig. 3. shows the differences in handling the urgent and non-urgent requests. The time between consumption

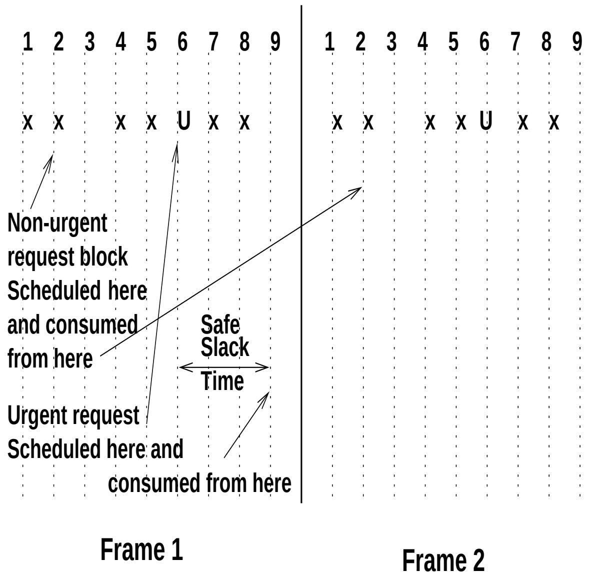
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
x x x x U x x
x x x x U x x
Non-urgent request block Scheduled here and consumed from here
Safe Slack Time
Urgent request Scheduled here and consumed from here
Frame 1
Frame 2

Fig. 3. Differences in handling urgent and non-urgent requests.

and the scheduling of the request block is shorter (equal to safe slack time) for urgent blocks and is equal to the width of a frame for non-urgent request blocks. This observation enables us to devise techniques for reducing the stream startup latency for urgent requests.

When an urgent request arrives, first we search for an open slot at the disk that has to service this request. If there is an open slot within the latency target time of this request, we will schedule the request in that slot. Else, an open slot at that disk is identified. Non-urgent requests are moved within their slack times to create a new open slot close to the arrival time of the urgent request. This is called a 'backward move' since a non-urgent request scheduled at time t is moved backward in time to $t + \delta$ to accommodate the newly arrived urgent request. An example backward move is shown in Fig. 4. Fig.4. shows the schedule in a single frame, the length of the frame being 10 slots. Request U arrives at time 2, but we cannot find an open slot until slot 7 which is beyond the latency target of 3 slots. In this case, the request scheduled in slot 5 is moved backward to slot 7 to make slot 5 open for the new request. In this example, it is assumed that it is safe to move backward up to 2 slots.

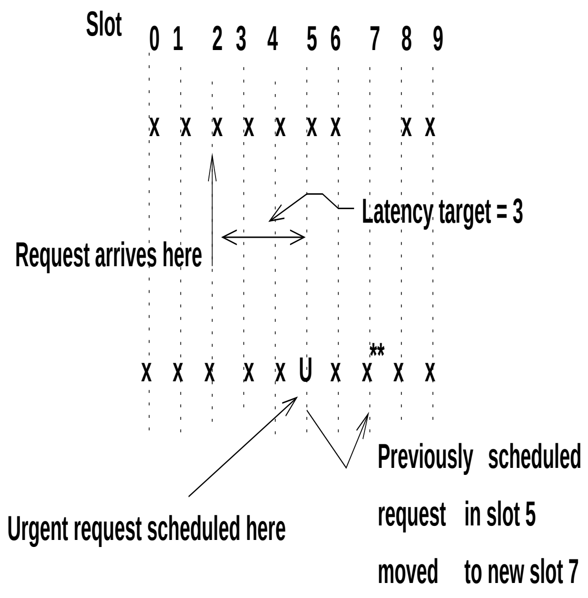

Fig. 4. Example backward move to achieve latency target.

The request now scheduled in slot 7 cannot be moved backward any further to accommodate any further urgent requests. The request now in slot 7 is marked with two asterisks to denote the fact that the request is already moved 2 slots.

A sequence of backward moves may be required to create an open slot within the latency target time of the new urgent request. This is shown in Fig. 5. If the urgent request arrives at time slot 0, again we find that there is no open slot within the target latency period. In this case, we make a series of backward moves, from slot 5 to slot 7 and then from slot 3 to slot 5 to create an open slot within the target latency period. Again we mark the two requests now in slots 5 and 7 with two asterisks to show that they are already moved 2 slots.

Since the non-urgent requests are moved within the schedule table without violating their safe slack times, those requests will still safely meet the deadlines.

As explained earlier, the urgent requests operate with their slack times close to the safe slack times so that the latency targets can be met. However, it is

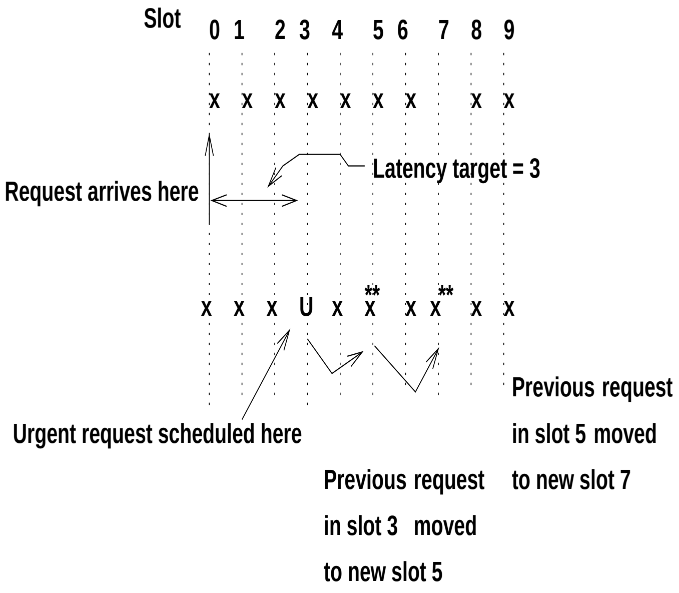
Slot
0 1 2 3 4 5 6 7 8 9
x x x x x x x x x
Latency target = 3
Request arrives here
** **
x x x U x x x x x x
Previous request in slot 5 moved to new slot 7
Urgent request scheduled here
Previous request in slot 3 moved to new slot 5

Fig. 5. Sequence of backward moves to achieve latency target.

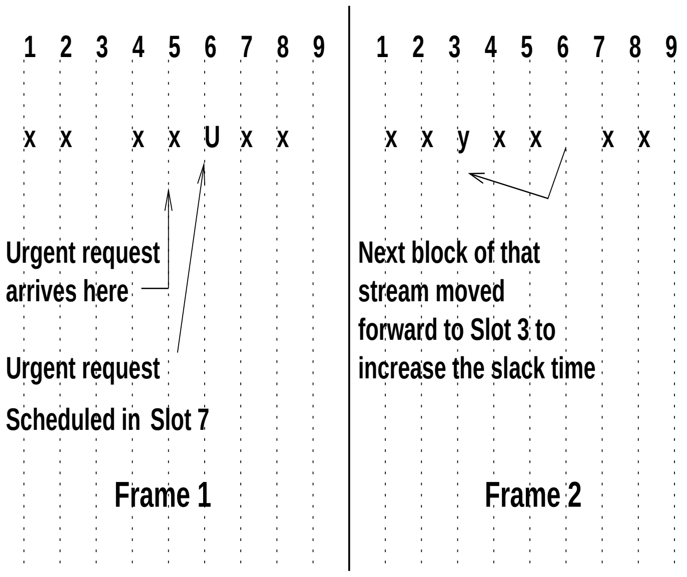
1 2 3 4 5 6 7 8 9
x x x x U x x
Urgent request arrives here
Urgent request Scheduled in Slot 7
Frame 1
1 2 3 4 5 6 7 8 9
x x y x x x x
Next block of that stream moved forward to Slot 3 to increase the slack time
Frame 2

Fig. 6. Example forward move to increase the slack time.

to be noted that the requests of a stream arriving in sequence after an urgent request can be made into non-urgent requests since their order is known ahead of time. Normally, when a request is scheduled for service at time t, the next request of that stream is scheduled for service at time $t + F$. In such a case, if the slack time of the urgent request is 'f' slots, then the requests of that stream that follow will only have a slack time of 'f' slots even though these requests are known ahead of time. To make the backward move technique work well for the urgent requests, we move the requests following an urgent request of a stream forward. This 'forward move' technique increases the slack time safely to as high a value as possible up to 'F' slots. This forward move technique turns the consecutive requests of a stream following an urgent request into non-urgent requests. Fig. 6 shows an urgent request U scheduled in slot 6 of frame 1. The next request block of that stream is moved from slot 6 to slot 3 in frame 2 to increase the slack time of that stream.

Forward move is applied aggressively to increase a scheduled stream's slack time to 'F' slots immediately after it is scheduled. Backward moves are applied to already scheduled streams to make an open slot available at the disk for an arriving urgent request. These two techniques, forward move and backward move, are applied when appropriate within the system to achieve the latency targets of requests.

3.3 Simulations

We simulated a system with 16 nodes, each node with 10 disks. Each disk is assumed to be able to sustain a bandwidth of 4 MB/sec. MPEG-1 streams of 200 KB/sec were considered. At these stream rates, each disk can support up to 20 streams/sec or each node can sustain 20*10 = 200 streams/sec. With the 16 nodes in the system, the system can sustain up to 200*16 = 3200 streams/sec. We assumed that the system is running up to 80% of capacity i.e., 0.8*3200 = 2560 streams are running on the system. The system is started with zero streams running initially. Then random requests for movie streams are generated until 2560 streams are generated and scheduled. Once the system reached this operating point, simulations involved (1) stopping an already scheduled stream at random and (2) starting a new movie stream at random. This would continue maintaining the operating capacity at a constant 80%. Hence, all the results reported here are obtained when the system is running at 80% capacity. When a random stream is stopped, the slots that it would have taken for service are marked open. The random new request can be either an urgent request or a non-urgent request. The percentage of the new requests that are urgent requests is varied from 0% to 100% in increments of 20%. If the new request is a non-urgent request, it is always scheduled in the first open slot at the disk where it requires service.

Different experiments are carried out for scheduling the urgent requests. In the first experiment, urgent requests are treated just like non-urgent requests and no latency reduction techniques are applied. In the second experiment, only forward move technique is applied and hence only non-urgent requests are moved to make room for the new request. In the third experiment, backward move technique is applied along with the forward move technique. In each experiment, 50,000 requests are generated and scheduled after the system reached an operating point of 80% capacity.

Startup latency of a new request is measured by the time it took to schedule the new request. Each experiment is run with a latency target. When latency reduction techniques are applied, scheduled requests are moved around to schedule an urgent request within the latency target if possible. If it is not possible to achieve the latency target for an urgent request, it is scheduled as close to the request time as possible. Time is measured in slots. For guaranteeing some reasonable response time to user, we used the 99-percentile latency i.e., the latency observed by 99% of the requests is below this measured value. This measure is the primary evaluation measure for the proposed techniques. For each latency target, achieved 99-percentile latency is measured. This gives us an indication about how often the latency reduction techniques could achieve the desired targets.

3.4 Results

No latency reduction

Fig. 7. shows the results obtained when no latency reduction techniques are applied. Since all requests, urgent or non-urgent, are treated the same, we observe that all the different curves with varying percentages of urgent requests observe the same latency. Since no latency reduction techniques are applied, the requests are scheduled at the first open slot. A 99-percentile latency of 66 slots means that 99% of the requests can be scheduled within 66*64ms = 4.2 seconds. For urgent requests during the transition from one video sequence to another in an interactive video, a response delay of 4.2 seconds would be unacceptable. However, this delay is quite acceptable when starting a new movie stream.

Backward moves

Fig. 8. shows the effectiveness of the backward move technique of delaying the non-urgent requests to accommodate the urgent requests. The x-axis shows the target latency threshold of the latency reduction technique and the y-axis shows the actual value of the 99-percentile latency achieved. When latency reduction is not effective, the target latency is not achieved. When the latency reduction is effective, the 99-percentile latency is less than or equal to the target.

When non-urgent requests form a higher percentage of the request stream, there are more non-urgent requests that can be moved backward to accommodate the new urgent request. Hence, the latency targets are met more easily when non-urgent requests form a higher percentage of the request stream. As the percentage of the urgent requests increases, we have fewer non-urgent requests and the backward-move latency reduction efficacy decreases. When the request

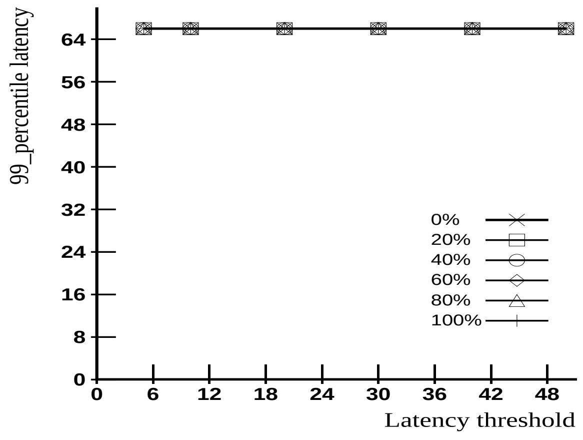

Fig. 7. Observed 99-percentile latency.

stream consists entirely of urgent requests, this technique has no effect and we observe a 99-percentile latency of 66 slots that we observed earlier.

Both techniques

Fig. 9. shows that the combination of these two techniques is clearly effective. Except for one case, the 99-percentile latency is below or equal to the latency target. In the one case of 100% urgent requests and a latency target of 5 slots, 99-percentile latency is higher at 9 slots and the required latency target of 5 slots could not be met. For this simulation, after an urgent request is scheduled, next request block of that stream is moved forward as far as possible without exceeding the F slots limit from its consumption time. If this is effective, successive blocks of this stream have a larger slack time than the minimum allowed and hence these requests can be moved backward to accommodate an urgent request that arrives at a later time. Backward move technique alone was inadequate to achieve the target latencies when there are higher percentages of urgent requests.

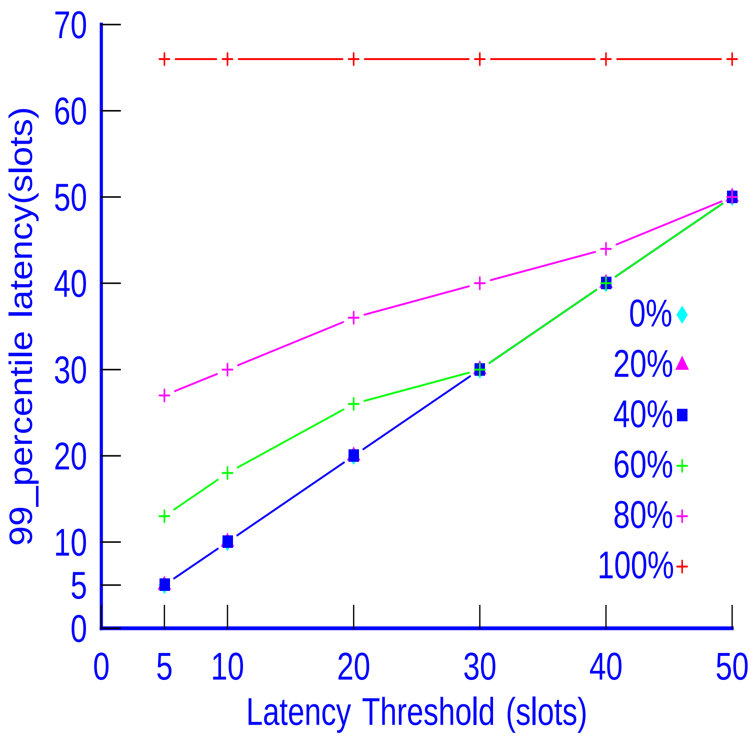

Fig. 8. Observed 99-percentile latency with backward move technique.

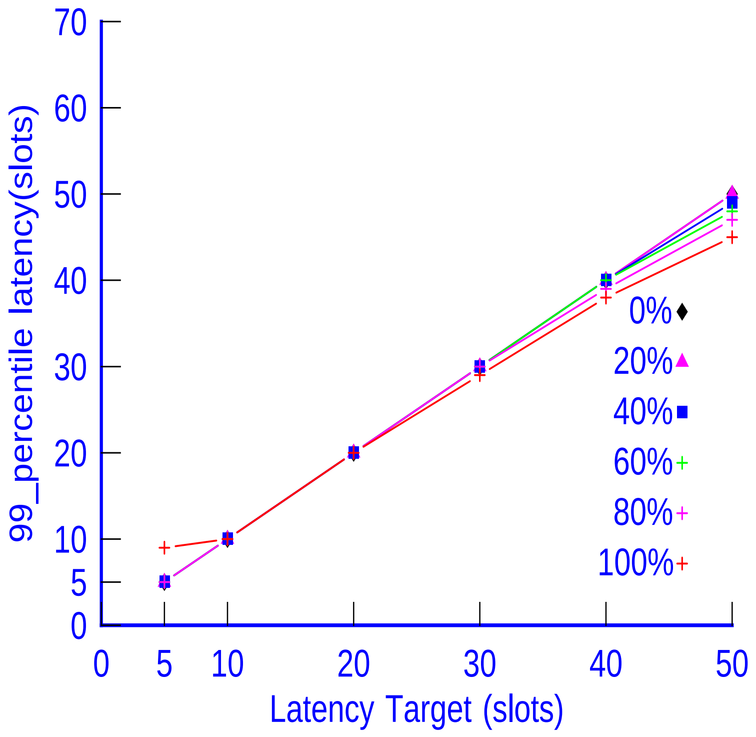

Fig. 9. Observed 99-percentile latency with both the techniques.

Latency Distribution

Fig. 10. shows the distribution of observed latencies for different requests in the request stream. When no latency reduction is applied, quite a few requests observe large latencies. For this experiment, the request stream had 80% urgent requests and the latency target was kept at 10 slots. When both the latency reduction techniques are applied, most of the requests with higher latencies are scheduled to observe a latency of less than or equal to 10 slots. Very few requests (less than 1% of the requests) have latencies higher than 10 slots. The sharp jump in frequency at exactly 10 slots is because of the fact that the latency target is kept at 10 slots. The latency reduction techniques are quite effective in changing the latency distribution such that most (99%) of the requests observe small latencies. The latency reduction techniques have clearly been effective in reducing the variance in the latencies for urgent requests.

Different disk parameters

The effectiveness of both the techniques described above is dependent on the relative bandwidths of the disks and the request streams. In earlier experiments we used MPEG-1 stream rates of 200 KB/sec and an effective disk bandwidth of 4 MB/sec. The disk hence has an effective speed ratio of 20 compared to the request streams. This determined the frame width in our experiments to be 20. Clearly, a scheduled request block cannot be moved from its slot by more than 20 slots forward or backward. We assumed in our earlier experiments that it is safe to move only 16 slots from its scheduled slot. This assumption is made to allow some communication time (4 slots) for the block to be moved from the disk to the customer's site. The effectiveness of both the proposed techniques are dependent on this parameter. The larger the frame width, the more options for moving a scheduled request block and hence the higher probability of meeting the latency target of an arriving urgent request. To see the sensitivity of our results to this parameter, the effective slack period is varied from 16 slots (used in earlier results) to 8 slots in steps of 2 slots. Fig. 11 shows the effectiveness of meeting the latency targets with varying slack periods. For these experiments, the percentage of urgent requests is kept at 80%. It is seen that the latency targets can be achieved when slack periods are greater than 12 slots. This implies that our techniques can be effective as long as the disk bandwidths are 12+4 = 16 times that of the request stream bandwidth. RAID3 type of disk arrays may be employed to maintain this speed ratio when higher request stream bandwidths need to be supported.

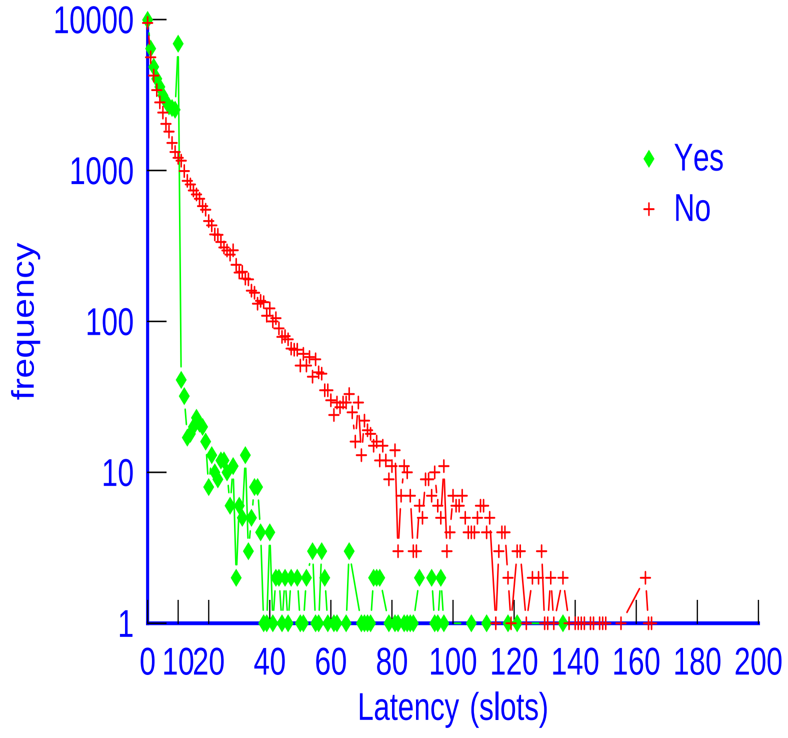

Fig. 10. Distribution of latencies with and without latency reduction.

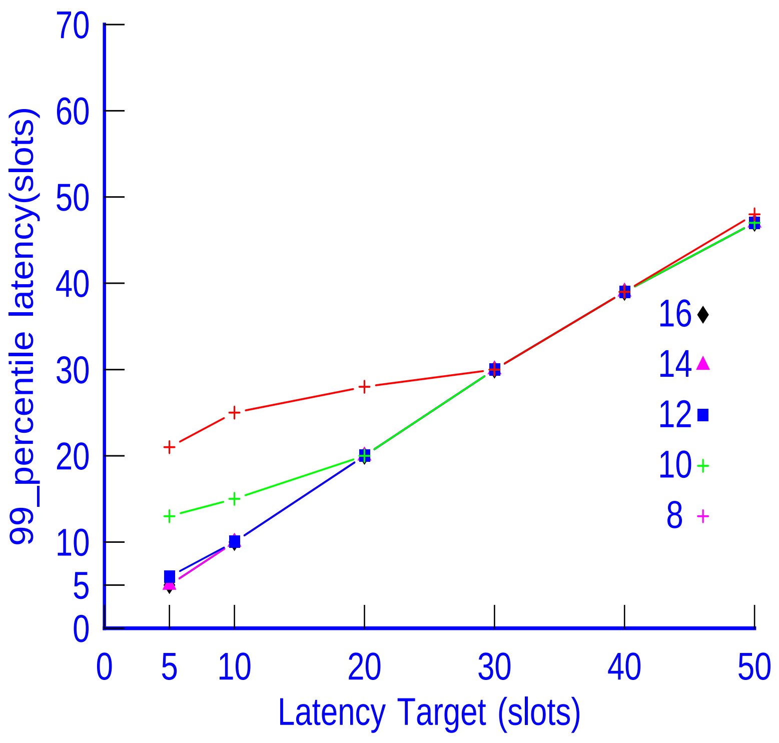

Fig. 11. Effect of the slack period.

4 SUMMARY

In this paper, we have addressed the problem of achieving short turn-around latencies for video streams in an interactive video server. We classified the requests into urgent requests and non-urgent requests based on the latencies required by the requests. We proposed two techniques for reducing the latencies of urgent requests. Through simulations, we showed that these techniques are quite effective in achieving sub-second response times for the urgent requests.

5 ACKNOWLEDGMENTS

Discussions with Jim Wyllie have contributed to the presentation of the ideas proposed in this paper.

REFERENCES

[1] A. L. Narasimha Reddy and Jim Wyllie. Disk scheduling in a multimedia I/O system. *Proc. of ACM Multimedia Conf.*, Aug. 1992.

[2] A. L. N. Reddy. Scheduling and data distribution in a multiprocessor video server. *Proc. of Int. Conf. on Multimedia Computing and Systems*, pages 256–263, 1995.

[3] D. P. Anderson, Y. Osawa, and R. Govindan. Real-time disk storage and retrieval of digital audio/video data. *Tech. report UCB/CSD 91/646, Univ. of Cal., Berkeley*, Aug. 1991.

[4] P. S. Yu, M. S. Chen, and D. D. Kandlur. Grouped sweeping scheduling for dasd-based multimedia storage management. *Multimedia Systems*, 1:99–109, 1993.

[5] J. Yee and P. Varaiya. Disk scheduling policies for real-time multimedia applications. *Tech. report, Univ. of California, Bekeley*, Aug. 1992.

[6] B. Ozden, R. Rastogi, and A. Silberschatz. A framework for storage and retrieval of continuous media data. *Proc. of IEEE Int. Conf. on Multimedia Computing and Systems*, 1995.

[7] H. M. Vin and P. V. Rangan. Designing file systems for digital video and audio. *Proc. of 13th ACM Symp. on Oper. Sys. Principles*, 1991.

[8] D. Anderson, Y. Osawa, and R. Govindan. A file system for continuous media. *ACM Trans. on Comp. Systems*, pages 311–337, Nov. 1992.

[9] R. Haskin. The shark continuous-media file server. *Proc. of IEEE COMPCON*, Feb. 1993.

[10] F. A. Tobagi, J. Pang, R. Biard, and M. Gang. Streaming raid: A disk storage system for video and audio files. *Proc. of ACM Multimedia Conf.*, pages 393–400, Aug. 1993.

[11] M. S. Chen, D. Kandlur, and P. S. Yu. Support for fully interactive playout in a disk-array-based video server. *Proc. of ACM Multimedia Conf.*, pages 391–398, Oct. 1994.

[12] K. D. Jayanta, J. D. Salehi, J. F. Kurose, and D. Towsley. Providing vcr capabilities in large-scale video server. *Proc. of ACM Multimedia Conf.*, pages 25–32, Oct. 1994.

[13] H. J. Chen, A. Krishnamurthy, T. D. Little, and D. Venkatesh. A scalable video-on-demand service for the provision of vcr-like functions. *Proc. of IEEE Conf. on Multimedia Computing and Systems*, pages 65–72, May 1995.