HEWLETT PACKARD

I/O Channel and Real-Time Disk Scheduling for Video Servers

Shenze Chen, Manu Thapar
Broadband Information Systems Laboratory
HPL-96-126
August, 1996

multimedia, VOD, real-time disk scheduling

A video server is one of the key components in providing the "video-on-demand" service. This service allows the delivery of movies (or games, sports, news, etc.) to homes on viewers' requests. Video servers are different from traditional file servers or database systems in the following major aspects: they have large storage capacity, must support intensive I/O traffic, large transfer size, and real-time disk I/O. Therefore, the video server I/O subsystem design is critical to its success. The current most popular I/O interface is SCSI. It has some problems, such as an unfair channel arbitration protocol, which are detrimental in an I/O intensive environment.

In this paper, we study the I/O scheduling issues at two levels: at the channel level, we present a fair SCSI channel scheduling mechanism, and at the disk level, we propose a new window-based real-time disk scheduling algorithm. Both the fair scheduling mechanism and the real-time scheduling algorithm are designed for the video server and other real-time environments. The performance of the proposed scheduling mechanism and algorithm is studied by using a video server simulator built in HP Labs, which simulates the SCSI bus arbitration, disk scheduling activities, memory buffer management, etc. The results show that a server using these scheduling mechanisms can save 50% of the memory buffers, or with the same amount of buffer, the system performance (measured by the maximum number of concurrent movie streams supported) can be improved by up to 35% as compared to regular SCSI and the conventional elevator disk scheduling algorithm, depending on the configurations. Using the video server simulator, we also studied some interesting related issues, such as the appropriate number of disks to attach to each SCSI channel, etc.

Presented at *The Sixth International Workshop on Network and Operating System Support for Digital Audio and Video,* Zushi, Japan, April 1996.

1. Introduction

Video server is one of the key components in providing the "video-on-demand" service. This service delivers movies (or games, sports, news, etc.) to homes immediately on viewers' requests. Video servers differ from traditional file servers or database systems in the following major aspects on data storage and retrievals:

- *Large Storage Capacity*

  Each compressed movie needs about 2 to 3 GB (giga-bytes) of storage space, depending on the coding bit-rate. For a large scale video server, which stores hundreds or even thousands of movies, hundreds giga-bytes or tera-bytes of storage space will be required;

- *Intensive Disk I/O Traffic*

  Typically, if a two hour movie is invoked to playback, it will generate more than 10,000 I/O requests during its playback period, depending on the request size. At a 3 Mbit/sec of stream bit-rate and 128K bytes of I/O transfer size, if 100 concurrent movie streams are requested, then the video server needs to sustain 300 I/O requests per second;

- *Large I/O Transfer Size*

  Disk utilization is directly related to the I/O transfer size. In general, disk service time consists of three components: seek time, rotational latency, and data transfer time from disk media to the disk buffer. Among the three, only the data transfer time directly contributes to the disk utilization.

$$Disk\_Utilization = \frac{Data\_Transfer\_Time}{Seek\_Time + Rotational\_Latency + Data\_Transfer\_Time}$$

  Therefore, in order to achieve high disk utilization (as well as I/O channel utilization), in an I/O intensive video server environment, the I/O transfer size is typically set to large (> 64K bytes), as compared to the 8K transfer size for conventional file servers, or 2K transfer size for database systems;

- *Real-Time Disk I/O*

  Movie streams must be continuously played back. Any discontinuity will result in a glitch on a viewer's screen. Therefore, in a video server, in order to provide good "quality-of-service", movie data must be retrieved from storage disks in a real-time fashion. The primary I/O performance metric for such a system is no longer the "mean I/O response time", as used for the traditional file

servers or transaction processing systems. Instead, we are more concerned about the number of I/O requests that miss their deadlines.

Based on these observations, video servers, or general media servers, are actually data movement engines, since their main responsibility is to move a huge amount of data from storage to distribution networks. For such servers, the I/O subsystem design is critical to the success of the entire system.

Currently, the most popular disk I/O interface is the "Small Computer Systems Interface", or SCSI [1] in short. While SCSI provides satisfactory performance for many applications such as transaction processing and NFS file servers, it has some disadvantages when used in video servers. One well-known issue is the SCSI unfair bus arbitration problem. I/O devices on a SCSI bus have different priorities, depending on their SCSI bus ID. If several devices arbitrate for the bus simultaneously, the winner is always the one with the highest priority. This may cause a serious problem for video servers, since in this environment, I/O requests carry time constraints and requests to low priority devices are more likely to miss their deadlines. The Fibre Channel (FC) interface [2,3] tries to solve this problem by defining a fair arbitration mechanism for its loop structure. Fibre Channel technology, however, is still in its early stage of development. Even when FC becomes mature, people may still want to leverage their existing SCSI devices because of the huge investment they have put in them.

In this paper, we study the I/O channel and disk scheduling issues in a video server environment. At the channel level, we present a fair SCSI channel scheduling mechanism, which tries to give each disk on a SCSI bus (we'll use the term SCSI bus and SCSI channel interchangeably in this paper) the same chance to transfer data from disk to the host. At the individual disk level, we propose a new window-based real-time disk scheduling algorithm, which tries to guarantee that each I/O request meets its deadline, and given that, seek optimization is performed within a request window. This new disk scheduling algorithm is independent of the I/O channel used, i.e., it can be applied to other storage interfaces such as Fibre Channel. In order to see the benefit achievable by using these scheduling mechanisms, a video server simulator has been built which simulates the SCSI arbitration and disk scheduling activities. The results show that a server using these scheduling mechanisms can save 50% of the memory buffers, or with the same amount of buffer, the system performance (measured by the maximum number of concurrent movie streams supported) can be improved by up to 35% as compared to regular SCSI and conventional elevator disk scheduling algorithm. Using the video server simulator, we also study some interesting related issues, such as the appropriate number of disks to attach to each SCSI channel, etc.

The remainder of the paper is organized as follows: in Section 2, we first review some previous related work; in Section 3, we propose a new real-time disk scheduling algorithm; Section 4 presents a fair SCSI channel scheduling mechanism; Section 5 describes the video server simulator model; the performance evaluation of the proposed disk and channel scheduling mechanisms is given in Section 6; and Section 7 summarizes this paper.

2. Related Work

In this section, we review previous work on the I/O scheduling. Disk scheduling has been a research topic for decades. Early work was focusing on reducing disk access times. One of the most popular algorithms is the "elevator" or circular SCAN algorithm [4] that tries to minimize the average disk access time. It, however, does not take into account the time constraints (if any) of I/O requests when scheduling. Real-time disk scheduling research started to appear in literature in late 80's and early 90's. In the following, we will review several of these algorithms. SCSI channel scheduling, however, to our knowledge, has not appeared in literature.

In an earlier work on real-time disk scheduling [5], Abbott presented an algorithm for real-time transaction processing systems, called "Feasible Deadline SCAN (FD-SCAN)". In this algorithm, the track location of the request with the earliest feasible deadline is used to determine the scan direction. A deadline is feasible if it is estimated that it can be met. At each scheduling point, all requests are examined to determine which has the earliest feasible deadline. After selecting the scan direction, the arm moves towards that direction and serves all requests along the way. The issues with FD-SCAN are: (a) its run-time overhead may be high since it needs to check if the previous scan target is still feasible, and if not, a new direction must be determined; and (b) if there are many requests between the arm position and the target request that has the earliest deadline, then this request is likely to miss its deadline. More recently, Reddy [6] studied a hybrid of SCAN and *earliest-deadline-first* (EDF) algorithm for multimedia systems, called SCAN-EDF. In this algorithm, requests are normally served in EDF order. If there are more than one requests have the same deadline, then the SCAN algorithm is applied on these requests. Apparently, the effectiveness of SCAN-EDF depends on how many requests carry the same deadline. If a server dynamically issues I/O requests, then the chance for more than two requests to have the same deadline may be small. In this case, the algorithm reduces to EDF. On the other hand, if a server uses "service round", and all requests during a round are assigned the same deadline, then the algorithm reduces to SCAN. Therefore, the behavior of this algorithm really depends

on how deadlines are assigned to I/O requests. The "Group Sweeping Scheme (GSS)" was studied in [7] and [8] for multimedia systems. In this scheme, movies retrievals are serviced in "service rounds". Within each round, requests are partitioned into groups and these groups are served in a fixed order. The SCAN algorithm is applied within each group. The number of groups (or the group size) can be tuned to achieve a better performance. This algorithm is appropriate when all of the disks over which movies are striped across are accessed simultaneously by each request. This requires I/O transfer size to be very large, which implies large memory buffers are needed in order to make this algorithm work efficiently. In addition, a study on disk scheduling for traditional systems can be found in [9]. Other related work on the video server I/O subsystem design, such as multimedia storage subsystem design, video layout on disks, and buffer management, can be found in [10,11,12,13,14,15,16].

3. Real-Time Disk Scheduling

3.1. A New Real-Time Disk Scheduling Algorithm

In this subsection, we present a new window based real-time disk scheduling algorithm, called RT-WINDOW, for video servers, which tries to overcome the drawbacks and limitations of existing real-time and non-real-time algorithms.

The RT-WINDOW algorithm works in the following way:

- All arriving I/O requests to a disk join a queue (e.g., the IO_Q shown in Figure 3) which is sorted according to request deadlines.

  As shown in the following figure, request *ri* carries deadline *di* and has a cylinder address *ci*. Request *r1* has the earliest deadline, *r2* has the next earliest deadline, and so on, with *d1* < *d2* < *d3* ...

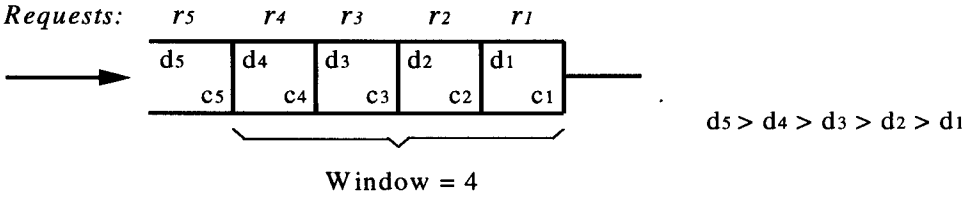
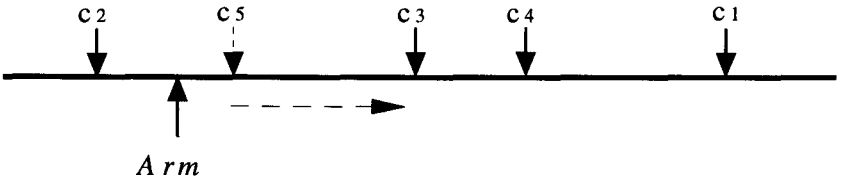


Figure 1. The IO_Q Structure of RT-WINDOW Algorithm.

- We define a window of size *n* to be the first *n* requests in front of the queue;

- If the remaining life time of *d1* is less than a *Threshold*, serve *r1*; Otherwise, scan towards *r1* and schedule anyone within the window.

  The *Threshold* above is an estimated worst-case response time for *r1*, which will be discussed in more detail in the next subsection.



As an example showing in above figures, the window includes four requests, *r1* to *r4*.. Depending on the deadline tightness of request *r1*, the algorithm decides whether to serve *r1* or not. If *r1* has some room to wait, then the algorithm scans towards *r1* and selects *r3* for service. Notice that: (a) *r2* is not selected because it is not in the direction towards *r1*, and (b) *r5* is not selected because it is not in the window.

3.2. Main Advantages of the New Algorithm

This window based new disk scheduling algorithm has the following advantages:

- *Real-Time Feature:*

The algorithm tries to make each request meet its deadline, which differentiates it from all of the non-real-time algorithms. At each scheduling point, if the remaining life-time for the request with earliest deadline, *r1*, is very close, then the algorithm will serve *r1* immediately.

- *Seek Optimization*:

Seek optimization is performed, but only when the request with the earliest deadline is guaranteed to meet its deadline. This is quite different from the existing real-time algorithm EDF in which no seek optimization is performed. This is also in contrast to the FD-SCAN algorithm, in which the seek optimization is always performed regardless of the time constraint of *r1*. In that case, *r1*, the request with the earliest deadline, is more likely to miss its deadline.

In addition, (a) the RT-WINDOW algorithm only scans within the window, i.e., the request being selected for service always has a relatively tight time constraint, whereas FD-SCAN will select anyone on the road, even though its deadline is still far away; this window mechanism will reduce the chance

for requests with tight time constraints to miss their deadlines; and (b) since the arm always moves towards the *r1*, after serving someone on the road, the remaining seek distance to *r1* is always reduced.

- *Flexibility*:

The RT-WINDOW algorithm has two adjustable parameters, *Threshold* and *Window-Size*. The performance of the algorithm can be tuned by carefully adjusting these two parameters.

Threshold: it is used to decide whether or not to serve the request with earliest deadline. One possible value for this parameter could be the estimated worst case disk I/O response time, which should include the service time (disk access time + channel transfer time) of this request plus that of other requests to be served before the one with the earliest deadline.

Window-Size: it decides the algorithm's behavior. Decreasing window size will put more weight on requests' time constraints rather than seek optimization. In the extreme case of Window-Size=1, the algorithm degenerates to the EDF algorithm. On the other hand, increasing window size will put more weight on the seek optimization than time constraints. If this Window-Size parameter is set to a very large value (e.g. 1000), then the algorithm will degenerate to FD-SCAN if it decides to scan. Experimental results show that a window size of 3 to 5 is appropriate for a wide range of workloads, and within this range, the overall system performance is not very sensitive to the window size.

Notice that, unlike SCAN-EDF whose effectiveness depends on the workload, i.e., if there are no requests carrying the same deadline, it degenerates to EDF, the new RT-WINDOW algorithm is totally independent of the workload. It can be set to behave like EDF or FD-SCAN by adjusting the algorithm's parameter(s). This provides great flexibility in using the algorithm.

- *Complexity*:

This algorithm can be implemented easily. One queue is maintained for each disk in the operating system kernel, which is sorted according to the request deadline. New requests issued by application programs to a disk are inserted into the corresponding queue. This insertion can be performed in parallel with disk operation(s). When the disk is ready to serve the next request, one request is dequeued according to the algorithm and sent down to the disk. Since the algorithm defines a fixed (small) window size, the computational complexity for selection is on the order of O(1), which guarantees that RT-WINDOW can be used as a on-line real-time disk scheduling algorithm.

In the discussions above, we presented the RT-WINDOW scheduling algorithm that tries to guarantee real-time constraint at each individual disk. Typically multiple disks are connected to an I/O channel to

share the channel bandwidth. Therefore, in order to achieve an optimal design of I/O subsystems, we also need to consider the channel scheduling issue, which will be addressed in the next section.

4. SCSI Channel Scheduling

4.1. The SCSI Unfairness Problem

Unfair arbitration is a known problem in the SCSI protocol. Each device on a SCSI channel has a unique device ID. During the arbitration stage, each device which is ready to transfer data from/to host(s) presents its ID on the SCSI bus. The winner of the arbitration is determined by the ID numbers present on the bus. Typically, there is one host on the bus. It has the highest priority and takes the address of 7. The device with ID 6 has the next highest priority and device 0 has the lowest priority. For SCSI-2 channels, which support up to 16 devices, the priority order could be 7, 6, ..., 0, 15, 14,..., 8. In a highly loaded system, if high priority devices continue to arbitrate, the low priority devices may starve.

To provide video-on-demand service, a video server is usually attached a large pool of disks which are used for the storage of movies. In this case, it is impractical to provide a dedicated channel for each disk. During playback, multiple movie streams are continuously retrieved from the disks and put into intermediate buffer memory in the server. In order to guarantee the quality of service, each I/O to the disk must carry a time constraint, i.e., it must complete before the data it retrieves is required for the playback. Obviously, any starvation on low priority disks will cause requests to these disks to miss their deadlines, resulting in unacceptable quality.

SCSI Bus ID	Target Bus Queuing Time (ms)	Distribution of Lost Request (%)
5	4.61	3.9
4	5.60	5.7
3	7.11	9.1
2	9.03	14.4
1	11.03	24.7
0	12.66	41.9

Table 1 : The SCSI Unfair Arbitration Problem.

In Table 1, we present some results obtained from a video-on-demand server simulator. The simulator model will be further described later in Section 5. Here we only show the seriousness of the SCSI unfair arbitration problem. As shown in the Table, a total of six disks are attached to each SCSI channel. The workload (number of concurrent streams) was intentionally set to be high enough for some of the I/O requests to miss their deadlines. As we see in this case, the average waiting time to win the arbitration for the lowest priority disk 0 is 2.7 times of that for the highest priority disk 5. Among those requests which missed deadlines, about 42% are missed at the lowest priority disk 0, which is 10 times higher than that at the highest priority disk 5.

Clearly, the SCSI unfair arbitration nature will cause some problems to the video server. The server needs either more memory to buffer the data or has to reduce the number of concurrent streams.

4.2. A Fair SCSI Channel Scheduling Mechanism

In this subsection, we present a fair SCSI channel scheduling mechanism. This mechanism takes advantage of the PRE-FETCH SCSI command which has not been widely used so far.

Modern Disk Structure and the Pre-fetch Operation

To help understanding the mechanism, we first describe the modern SCSI disk structure. As shown in Figure 2, a modern SCSI disk contains an internal data buffer, and the typical buffer size ranges from 256K to 1M bytes. This buffer is segmented to support multiple I/O transactions. For example, if the transfer size is 128K, then a 512 KB buffer can be divided into 4 segments to support up to 4 concurrent I/O transactions.

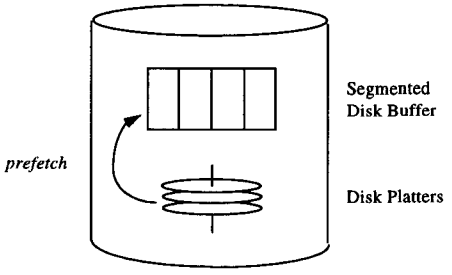


Figure 2. Modern SCSI Disk Structure.

Usually, when a READ command is received from the host, the disk controller first checks the buffer to see if the requested data is already in the buffer. If it is not, the controller starts to fetch the data from the disk media and put into the buffer. When the data is in the buffer, the controller begins to

arbitrate the SCSI bus and send data from the internal buffer to host memory across the SCSI bus. The PRE-FETCH command behaves the same as the READ command except when data is in the buffer, the disk controller does not transfer the data to host. In this case, if the buffer is not reused, the next READ command to this data finds it in the buffer and no media access takes place.

The Channel Scheduling Mechanism

The SCSI channel scheduling mechanism utilizes the pre-fetch operations described above, i.e., for each read request received from applications, the kernel will issue a PRE-FETCH command to the disk. After the pre-fetch finishes, READ commands are issued to disks on the same bus in a round-robin fashion, as shown in Figure 3.

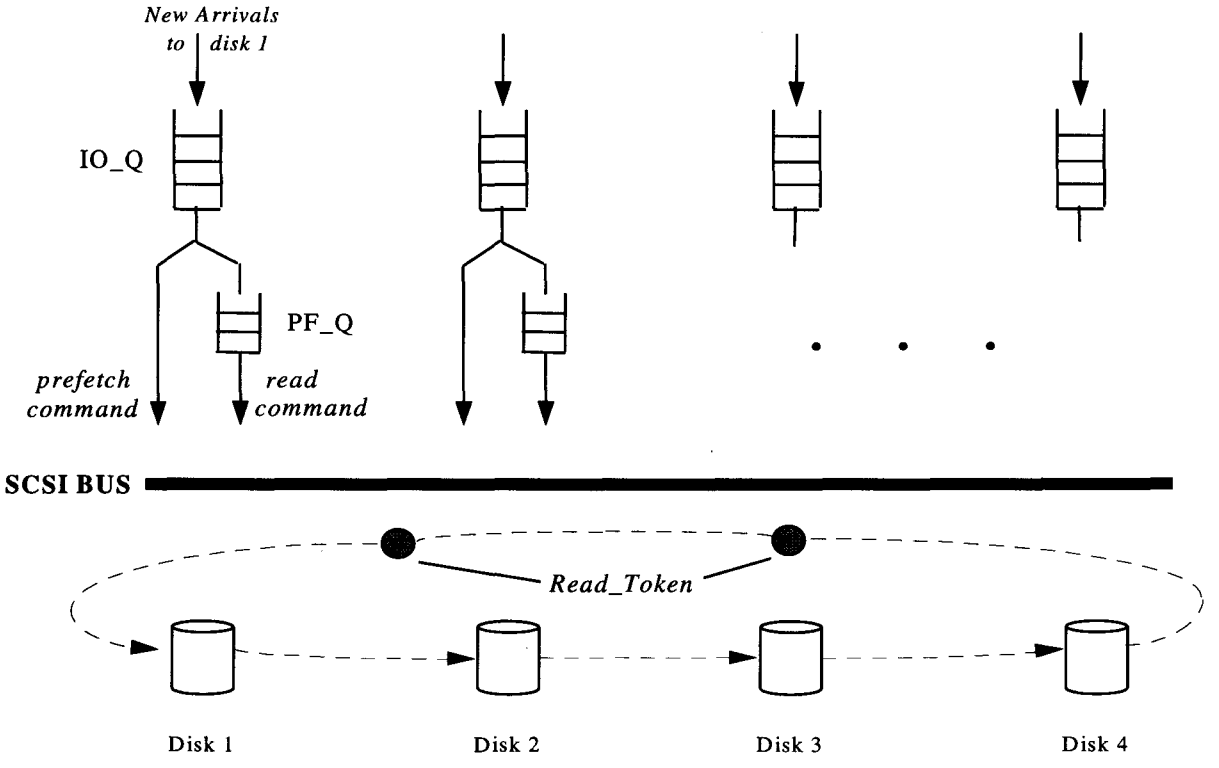


Figure 3. The SCSI Channel Scheduling Mechanism.

From the figure, we see that there is one IO_Q maintained for each disk, and new requests join this IO_Q when received by the kernel. When a read request is scheduled for service (by the disk scheduling algorithms described in Section 3), a PRE-FETCH command is issued to the disk, and a stub (or buffer header structure) of the request enters a pre-fetch queue (PF_Q) with tag PF_UNDONE.

When the pre-fetch finishes, the tag is changed to PF_DONE. There are two Read_Token's passing around all disks attached to a SCSI bus. The kernel can issue a READ command to a disk only when it holds a Read_Token and there is a request waiting in the associated PF_Q with tag PF_DONE. We selected two Read_Token's instead of one in order to have better utilization of the bus bandwidth. Typically, even when the requested data is already in the disk buffer, a READ command may still experience 800 us to 1 ms of disconnect time between the time the READ command arrives at the disk and the disk begins to arbitrate the bus for data transfer. If there is only one Read_Token, the bus will sit idle for most of the disconnect period per read, and therefore the bus bandwidth is wasted. On the other hand, if three or more Read_Tokens are used, the SCSI unfairness problem will come back again. (Notice that with the two token system, bus access is guaranteed to be fair, since when one disk disconnects, the other disk can always get on the bus.)

The more specific algorithm for channel scheduling is given below.

ALGORITHM:

Initialization:

(1). set all access_right = yes;
(2). set all Read_Token = idle;
(3). set all num_outstanding = 0;

New Arrival:

```
(1).    if ( num_outstanding < MAX_OUTSTANDING ) {
                issue_prefetch_command( );
                enqueue( PF_Q ) with tag PF_UNDONE;
                num_outstanding ++ ;
        } else {
                enqueue( IO_Q );
        }
```

Pre-Fetch Done:

```
(1).    mark PF_DONE;
(2).    if ( access_right == yes  AND  there is an idle Read_Token ) {
                remove from PF_Q;
                access_right = no.
                Read_Token = busy;
                issue_read_command( );
        }
```

Read Done:

```
(1).    if ( IO_Q not empty ) {
```

```
            /* perform real-time disk scheduling and dequeue( IO_Q ) here */
            new_req = disk_scheduling( );
            issue_prefetch_command( );
            enqueue( PF_Q ) with tag PF_UNDONE;
        } else {
            num_outstanding -- ;
        }
        Read_Token = idle;                                  /* release Read_Token */

(2).    Loop:
        {
          pass the Read_Token to next disk on bus;
          if (access_right == yes  AND  there is one in PF_Q with tag PF_DONE) {
            remove from PF_Q;
            access_right = no;
            Read_Token = busy;
            issue_read_command( );
            exit Loop;
          }
        } (Until all disks on the bus have been checked )          /* end Loop */

(3).    if ( no read  issued in (2) ) {
            reset all access_right to yes;
            if ( there is one with PF_DONE in any disk found in (2) ) {
                go back to (2);
            }
        }
```

Figure 4. The SCSI Channel Scheduling Algorithm.

In this algorithm, each disk attached to a SCSI bus is initially given an access right. Once a disk gets a chance to transfer data to host, its right is canceled until all other disks on the same bus which are waiting for data transfer have a chance to do so. This guarantees fair accesses for all disks on bus.

The number of outstanding requests to each disk is controlled by a kernel parameter MAX_OUTSTANDING. The purpose to having multiple outstanding requests to each disk is to keep the disks busy so that disk bandwidth can be better utilized. For example, when a previous request is transferring data to the host, the next one waiting in the disk can start to pre-fetch data from disk media to the disk buffer. On the other hand, too many outstanding requests won't help either. As long as when a disk's turn arrives, there is one outstanding request which has been pre-fetched, it will be good enough. In addition, if there are many outstanding requests to a disk, they will queue up in the disk and

the disk may reorder the execution sequence of these requests which may be undesirable since it may delay the execution of some requests with very tight deadlines. Finally and more importantly, the number of outstanding requests cannot exceed the total number of buffer segments in a disk, since it may cause the data in a buffer segment be overwritten before being transferred to host. For example, if the disk buffer size is 512 KB, which represents the case of today's majority SCSI disks, and the I/O transfer size is 128 KB, then the number of outstanding requests can not be more than 4. Based on the above discussions, we recommend the MAX_OUTSTANDING be 2 or 3.

One of the drawbacks of the above SCSI channel scheduling mechanism is that for each read request received from applications, the kernel needs to issue two SCSI commands, which doubles the total number of commands across the SCSI bus as compared to regular SCSI systems. The extra bus overhead for the PRE-FETCH command and Status phase, however, is relative small (less than 100 us) as compared the data transfer time across the bus (about 7 ms for 128KB transfer). We do point out that this extra overhead will increase when I/O transfer size decreases. This implies small transfer sizes will have a negative impact on the performance of this mechanism. Therefore, transfer sizes of less than 64 KB are not recommended.

Another issue that needs to be addressed is whether the Status byte which is sent from a low priority disk to host notifying a PRE-FETCH command finished can get starved because of channel contention. Fortunately, this is not a serious problem in this design. Typically, any disk may have its turn for data transfer. If there is a Status byte for a PRE-FETCH command waiting to be sent to host, the transfer of the Status byte can always be done before the data transfer. Our simulation also verified this scenario.

5. The Video Server Model

In order to study the behavior of the channel and disk scheduling algorithms, we have built a video server simulator which simulates the activities of a server when retrieving data from disks to the server's movie buffer. The video server model is shown in Figure 5.

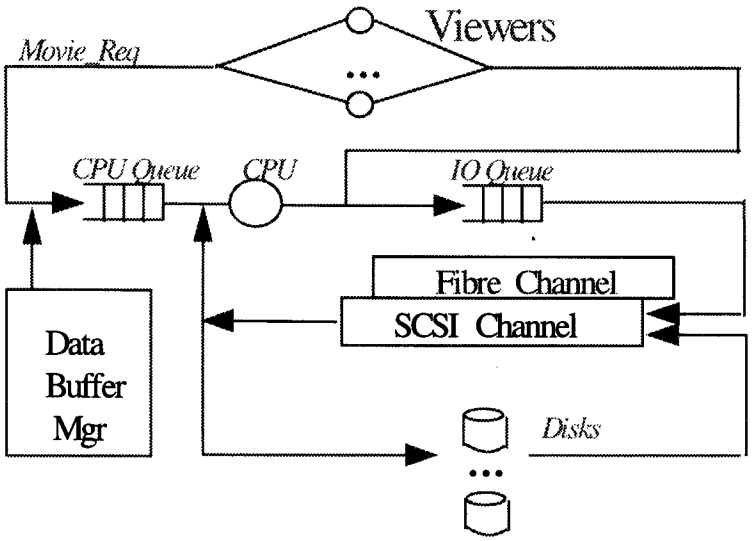


Figure 5: The Video Server Model.

In this model, there are multiple viewers randomly generating viewing requests. Movies are pre-allocated on disks according to some movie layout strategies. Movie lengths are uniformly distributed between 100 and 120 minutes. Each viewing request selects a movie to watch, and therefore all of the I/O requests generated by a movie stream are directed to certain locations on the disks. A viewing request can also select a play-back rate for different display quality. For simplicity, however, the results reported below use a fixed stream bit-rate of 3 Mbit/sec, and there are no fast-forward or fast-backward operations. When a movie finishes, the viewer goes back to a "think" state for a random period of time and then generates another viewing request.

The video server model contains many components, such as buffer management, disk data layout, disk scheduling, CPU scheduling, SCSI channel, disk module, etc. Since in this paper we are focusing on the disk and channel scheduling, we only use some simple strategies for the other components in the simulator. Issues, such as video data layout, alternative storage interfaces, movie buffer management, are studied in other papers [3][14].

Data Striping: Movies are striped across all disks attached to the server, with striping size equals to the I/O transfer size (which is fixed to 128K bytes in this study).

Workload and Buffer Management: When a customer viewing request is received by the server, it allocates a main memory buffer space for the stream. The buffer space is divided into segments, with segment size the same as the I/O transfer size (128K bytes). The segments for a stream are retrieved

and played-back in round-robin order. When a segment has been played-back, the server issues the next I/O immediately to fill in the freed segment. Thus, with constant bit-rate, the generating of I/O requests for each stream is triggered by the Segment_Empty signal periodically with the period equals to the play-back time of a single segment, and each I/O requests carries a deadline, which is equal to the play-back time of remaining segments assigned to the stream. Obviously, the play-back time of each segment is a function of the stream bit-rate and the segment size. For example, at a 3Mb/sec stream bit-rate and 128K segment size, if the stream is assigned a 512KB buffer space, then each I/O for this stream carries a deadline of 1 second.

For the RT-WINDOW disk scheduling algorithm parameter Window_Size, as mentioned before in Section 3.2, a window of 3 to 5 is appropriate and the system performance is not very sensitive to the window size within this range. Thus we fixed the Window_Size to be 5 for the results reported below. The other parameter, Threshold, could be estimated according to the disk parameters, the number of disks per channel, and MAX_OUTSTANDING, the maximum number of outstanding requests allowed to each disk (which is fixed to 2). For simplicity, we fixed the Threshold to be 350ms for all of our experiments. A slightly better performance can be expected if this parameter is fine tuned according to the system configurations. The other simulation parameters are listed in Table 2.

System Parameters:	
Movie Stream Bit Rate	3 Mbit /sec
Movie Length	100 - 120 min.
Num. of F/W SCSI Channels	4
Num. of Disks	8 - 60
I/O Transfer Size	128K
Buffer Size per Stream	256K - 1M

Table 2. Simulation Parameters.

In the simulator, we simulated four parallel fast/wide SCSI channels. Movies are striped across the four channels so that each channel receives the same workload. Two classes of disks are used by the simulator: popular drives and fast drives. The popular drive category includes drives with 5400 RPM (Rotations Per Minute), such as HP C2247, IBM 0644, Seagate ST32430, Micropolis 1936AV, DEC DSP 3105, etc. The fast drive category includes the state-of-art high end 7200 RPM drives, such as

IBM DFHS series, Seagate ST32550, HP C3330, Micropolis 3243AV, etc. Table 3 shows the disk parameters used by the simulator.

	Popular Drive (HP C2247)	**Fast Drive (IBM DFHS)**
Avg. Seek Time (ms)	10.5 ms	8.0 ms
Data Transfer Rate	3.1 - 5.3 MB /sec	9.6 - 12.6 MB /sec
RPM	5400	7200

Table 3. Disk Parameters.

6. Performance Results

In this section, we report some simulation results obtained from the video server simulator described above. The performance metric here is the total number of concurrent streams the server can support. We vary the number of disks attached to the server and total amount of buffer assigned to each stream as parameters. The results of SCSI channel scheduling and real-time disk scheduling are compared to regular SCSI systems using elevator (C-SCAN) disk scheduling algorithm, with the same number of disks and the same size of buffers.

The performance results show the maximum number of concurrent movie streams that can be serviced without any I/O request missing its deadline. Each data point is obtained by averaging over 6 runs. Within each run, 500 movies are viewed by multiple viewers. Since a movie lasts 110 minutes on the average, at 3 Mbit/sec stream rate and 128KB I/O transfer size, each movie stream generates about 20,000 I/O requests. Hence, each data point obtained from the simulator corresponds to 6×10^7 I/O requests issued without a single missed its deadline. While in this study, we targeted at a zero-miss ratio for I/O requests, it will also be interesting to know how the system performs if we allow some (small) I/O miss ratios. Due to space limitations, we'll leave this topic for a separate study.

6.1. Performance of Channel and Real-Time Disk Scheduling

Figure 6 illustrates the performance results of three scenarios: regular (unfair) SCSI system coupled with conventional C-SCAN disk scheduling algorithm, fair SCSI system as a result of using the above channel scheduling mechanism coupled with conventional C-SCAN algorithm, and fair SCSI system coupled with our real-time disk scheduling algorithm RT-WINDOW.

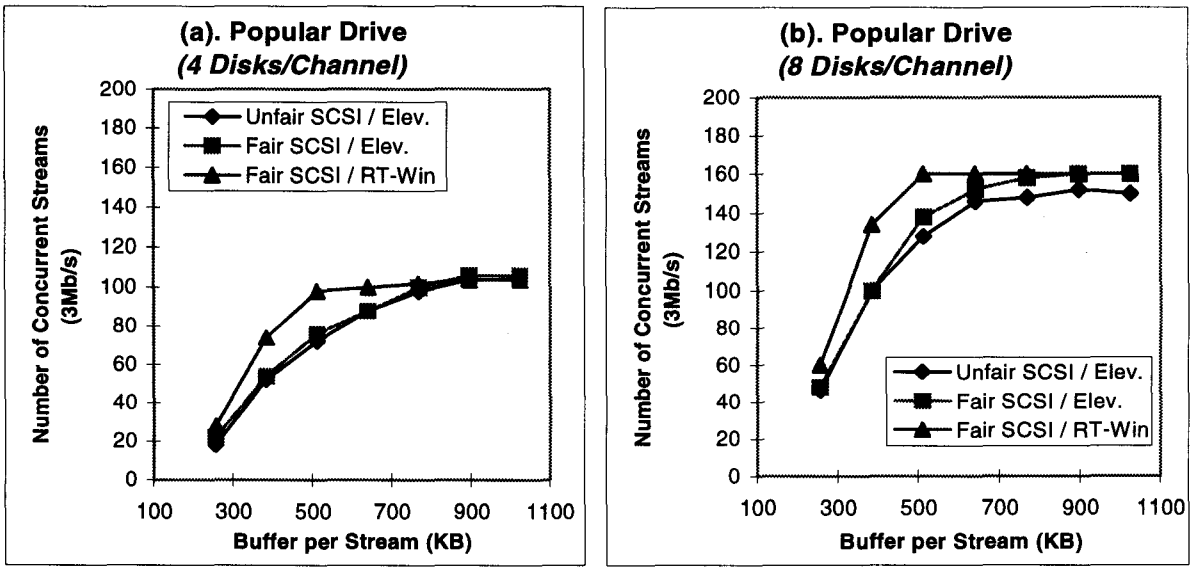


Figure 6. Performance of Channel and Real-Time Disk Scheduling (Popular Drive).

In Figure 6 (a), 4 disks are attached to each fast/wide SCSI channel. From the figure, we observe that applying only channel scheduling does not help, since in this case, disks are the bottleneck and SCSI channel utilization is still relative low. Therefore the channel contention is not a serious problem. When buffer size per stream increases, the number of concurrent streams increases until the disk bandwidth limit is reached. On the other hand, if the conventional C-SCAN algorithm is replaced by the real-time disk scheduling algorithm, RT-WINDOW, the system performance can be significantly improved, i.e., the same performance can be achieved by using only half buffer memory, or when the same amount of memory (512 KB per stream) is used, the RT-WINDOW can perform 35% better. When more disks are added to a SCSI channel, the bottleneck starts to shift from disks to the channel. As shown in Figure 6 (b), where 8 disks are attached to each channel, when channel contention increases, the channel scheduling becomes beneficial, which could outperform regular SCSI by 6.7%. The real-time disk scheduling algorithm adds one more level of improvement and requires much less memory buffers. The peak system performance (where the SCSI channel has been saturated) can be achieved by using 512 KB buffer for each stream. At that point, it performs 16% better than fair SCSI with C-SCAN algorithm and 25% better than regular SCSI with C-SCAN algorithm.

6.2. Using Fast Disk Drives

When fast disk drives are used, individual disk bandwidth increases, and therefore less disks may saturate a SCSI channel. Meanwhile, by using fast drives, the disk service time (including seek, rotational latency, and the data transfer time from disk media to disk buffer) is reduced, which will result in less memory buffer required to serve the same workload. Figure 7 shows the same experiment as the previous subsection except that the 7200 RPM IBM DFHS drives are used instead of 5400 RPM popular drives. From Figure 7(a), we observe that the SCSI contention begins to show up when 4 such fast disks are used. This is quite different from the case of using slower 5400 RPM drives as shown in the last subsection.

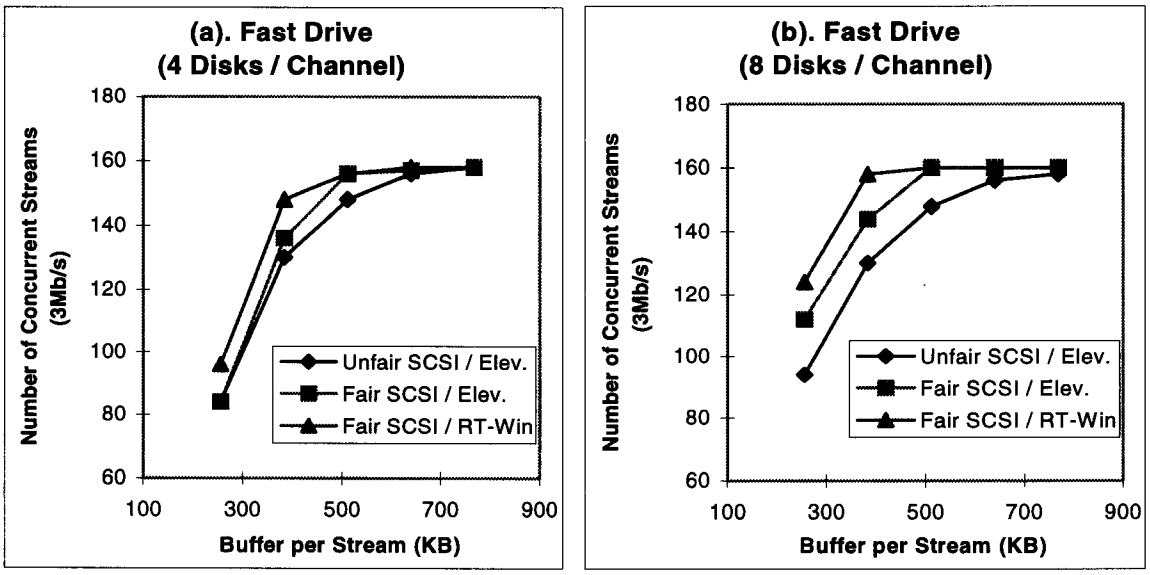


Figure 7. Performance of Channel and Real-Time Disk Scheduling (Fast Drive).

From both (a) and (b), we see the system using SCSI channel scheduling coupled with real-time disk scheduling continues performs the best. 384 to 512 KB of memory buffer per stream can lead the system to reach its peak performance, whereas 896 to 1024 KB of buffer is required for a system using regular SCSI and conventional C-SCAN disk scheduling algorithms. When 384 KB is allocated to each stream, which corresponds to the room for retrieving three requests given the request size of 128 KB, the Fair-SCSI / RT-WINDOW system outperforms the Unfair-SCSI / Elevator system by 12% and 22% for the cases of (a) and (b), respectively.

6.3. Varying Number of Disks per Channel

In previous figures, we fixed the number of disks per channel to 4 and 8, and varied the buffer size per stream as a parameter. Now we fix the buffer size and study the performance impact of varying the number of disks per SCSI channel. The results are shown in Figure 8. The buffer size of 512 KB per stream is used for popular drives and 384 KB for fast drives. With these amount of buffer, the Fair-SCSI / RT-WINDOW system starts to reach its peak performance.

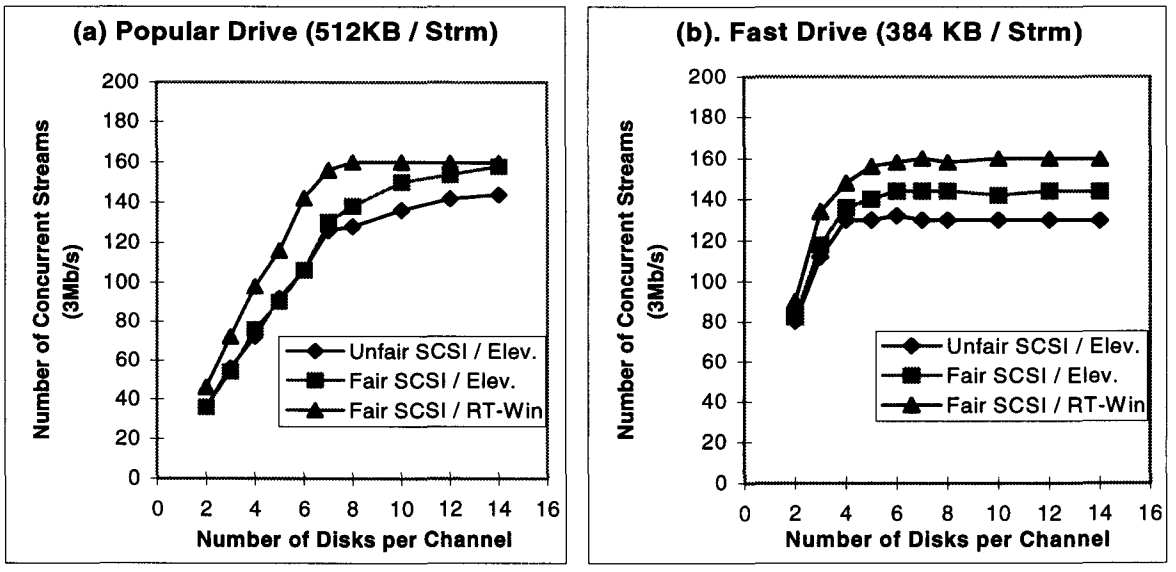


Figure 8: The Performance Impact of Varying Number of Disks per Channel.

As can be seen from the figure, 7 to 8 of the popular drives per SCSI channel is needed in order to achieve the best performance, whereas 4 to 5 of the fast drives per channel will be appropriate. At these points, the disk bandwidth and the SCSI channel bandwidth utilization is roughly balanced. After theses turning points, when more disks are added to each SCSI channel, the disk bandwidth utilization as well as the overall system performance will be limited by the SCSI channel bandwidth. Therefore, when the number of disks per channel increases, the overall system performance will remain the same, but the disk utilization will decrease. Again, the Fair-SCSI / RT-WINDOW system provides the best performance as compared the other two systems.

7. Summary

In this paper, we studied the SCSI channel scheduling and real-time disk scheduling issues in a video server environment, in which video data needs to be retrieved from each disk within a certain time constraint. First, we presented a window based real-time disk scheduling algorithm, called RT-WINDOW. This algorithm tries to guarantee each disk I/O request to meet its deadline, and given this, it performs seek optimization within a pre-defined small window. Second, we illustrated a channel scheduling mechanism to solve SCSI unfairness problem, which is inherited from the SCSI channel arbitration process. This mechanism uses the SCSI PRE-FETCH command to grab data from disk media into disk buffer. Then data from each disk's buffer is scheduled to transfer to the host across SCSI channel in a round-robin fashion.

The performance of the channel and real-time disk scheduling has been studied by using a video server simulation model, which includes many components such as memory management module, CPU scheduling module, disk scheduling module, SCSI interface module, etc. Since we focus on the SCSI interface and disk scheduling in this paper, we fixed the simulation parameters in other modules. For SCSI channel, we simulated the SCSI arbitration, Command, Data, and Status phases. The results of using the channel and RT-WINDOW real-time disk scheduling are compared with that of using regular SCSI and conventional elevator algorithm under the same workload and system configuration. With these new scheduling algorithms, the maximum number of concurrent movie streams supported by servers could be achieved by only using half of the memory buffers. When the same amount of memory buffer is used, these scheduling algorithms can improve the system performance by up to 35%.

Acknowledgment:

The authors wish to thanks John Wilkes for helpful discussions on the SCSI channel accesses.

References:

[1] ANSI Standard X3T9.2/86-109 *Small Computer System Interface-2 (SCSI-2),* Oct. 1991.

[2] ANSI Standard X3T11/93-275 *Fibre Channel Arbitrated Loop (FC-AL),* rev 4.1, Feb. 1994.

[3] Chen, Shenze and Thapar, Manu, "*Fibre Channel Storage Interface for Video-on-Demand Servers*", Proc. of Multimedia Computing and Networking'96, San Jose, CA, Jan. 1996.

[4] Peterson, J. L. and Silberschatz, A., *Operating System Concepts,* 2nd Ed. Addison-Wesley Publishing Company, 1985.

[5] Abbott, R. and Garcia-Molina, H., *"Scheduling I/O Requests with Deadlines: A Performance Evaluation,"* Proc. of Real-Time Systems Symposium, pp.113-124, 1990.

[6] Reddy, A.L.Narasimha and Wyllie, James C., *"I/O Issues in a Multimedia System,"* Computer, 27(3):69-74, 1994.

[7] Yu, P., Chen, M.S., and Kandlur, D.D., *"Grouped Sweeping Scheduling for DASD-based Multimedia Storage Management",* Multimedia System Journal, 1:99-109, 1993.

[8] Gemmell, D.J. and Han, J., *Multimedia Network File Servers: Multi-Channel Delay Sensitive Data Retrieval",* Multimedia Systems, 1(6):240-252, 1994.

[9] Seltzer, M., Chen, P., and Ousterhout, J., *"Disk Scheduling Revisited,"* Proc. of the Winter'90 USENIX Conf., pp.22-26, Jan. 1990.

[10] Vin, Harrick M. and Rangan, P.Venkat, *"Designing a Multiuser HDTV Storage Server,"* IEEE J. on Selected Areas in Comm. Vol. 11, No. 1, pp.153-164, Jan. 1993.

[11] Rangan, P.Venkat and Vin, Harrick M., *"Efficient Storage Techniques for Digital Continuous Multimedia,"* IEEE Trans. on Knowledge and Data Engineering, Special Issue on Multimedia Information Systems, Aug. 1993.

[12] Berson, S., Muntz, R., Ghandeharizadeh, S., and Ju, X., *"Staggered Striping in Multimedia Information Systems,"* in Proc. of SIGMOD'94, 1994.

[13] Kenchammana-Hosekote, D. and Srivastava, J., *"Scheduling Continuous Media in a Video-on-Demand Server,"* in Proc. of the Int'l Conf. on Multimedia Computing and Systems, pp.19-28, May, 1994.

[14] Chen, Shenze and Thapar, Manu, *"Zone Bit Recording Enhanced Video Data Layout Strategies,"* Proc. of MASCOTS'96, San Jose, CA, Feb. 1996.

[15] Buddhikot, M.M., Parulkar, G.M., and Cox, J.R. Jr., *"Design of a Large Scale Multimedia Storage Server,"* Computer Networks and ISDN Systems, 27, pp.503-517, 1994.

[16] Dan, Asit and Sitaram, Dinkar, *"Buffer Management Policy for an On-Demand Video Server,"* IBM Research Report RC 19347, 1994.