A Novel Deadline Driven Disk Scheduling Algorithm for Multi-Priority Multimedia Objects

Ibrahim Kamel
Panasonic Information and Networking Tech. Lab.
Princeton, NJ08540
ibrahim@research.panasonic.com

T. Niranjan
Transarc Corporation
Pittsburgh, PA 15219
Niranjan@transarc.com

Shahram Ghandeharizedah
Dept. of Computer Science
University of Southern California
shahram@usc.edu

Abstract

In this paper we introduce a new deadline driven disk scheduling algorithm designed for multimedia servers. The proposed algorithm supports real time requests with multiple priorities, e.g., those for different object classes in digital library applications. The proposed algorithm enhances utilization of disk bandwidth by (a) maintaining one queue for all requests, and (b) optimizing the seek time. Prior schemes, collectively termed "Multi-queue schemes", maintain a separate queue for each priority group and optimize the performance of the high priority requests only. When compared with our proposed scheme, our technique provides approximately two order of magnitude improvement in meeting the deadline of low priority requests. In addition, this algorithm provides both a better disk utilization and a better average response time. Under certain conditions, our algorithm violates the deadline of a few high priority requests (less than 5 out of a million requests).

1 Introduction

Multimedia applications are growing rapidly due to the advances in computer hardware and software technologies. In particular, the advancement in mass storage, video compression, and high-speed networks have made it feasible to provide multimedia services, such as news broadcasting, digital libraries, and video on demand [LS94]. With these applications, disk bandwidth is a scarce resource that must be managed intelligently. This is because when a disk is invoked to read a data block, it performs a certain amount of useful (transferring of data) and wasteful (seek and rotational latency) work. The duration of wasteful work must be minimized in order to enhance performance. (In a recent article [SSH99], Stankovic et. al., identify scheduling techniques and resource management in support of multimedia data as an important research direction that is not addressed sufficiently.) While disk technology continues to improve in terms of both speed and storage capacity, application requirements are also increasing at either the same or faster pace. For example, both the news broadcasters and the entertainment industry have started to use high quality video format, e.g., MPEG 4:2:2, DVCPRO. The bandwidth requirements to display a single video clip encoded in either MPEG 4:2:2 or DVCPRO ranges between 20-60 Megabits per second (Mbps).

Multimedia servers are used to store a wide variety of object types, e.g., text, images, and video. Different object types have different deadline requirements and priorities. Continuous media objects, e.g., audio and video clips, have strict deadline constraints. Their data blocks must be delivered within a predetermined time range. These deadlines are mandated by the display rate of the object. If a request is not serviced within its deadline, it is considered lost, resulting in a possible discontinuity of display at a station. Such disruptions and delays are collectively termed *hiccups*.

A digital library application may consist of both continuous media and traditional data (termed non-real-time) such as text, still images, ASCII text and scanned documents. Non-real-time object requests do not have deadlines and are given a lower priority than video and audio objects. However, these requests must be served in a manner that prevents starvation.

Not all real-time object requests have identical priorities [RS97]. Some are more important than others. For example, with MPEG encoded video, a video clip is represented as a sequence of I, P, and B frames. Frames are organized into groups of pictures. $I - frame$ is an *Independent frame* that can be decoded independently from other frames. While P and B frames both need the previous I frame for the decoding process. In this sense I frames are more important than P and B frames. Losing an I frame is more disruptive than losing either P or B frames. A disk scheduling algorithm that gives a higher priority to the retrieval of I frames results in a better Quality of Service (QOS).

As another example, it is well known that humans are more sensitive to faults in audio than in video. If it is inevitable to discard data packets, it might be better to discard video packets in favor of delivering audio packets.

As a final example from video on demand applications, the service providers may offer a range of services with higher quality service provided at higher prices. For example, there might be two level of services: high-quality and best-effort. Customers ordering high-quality services are expected to be charged more than those ordering best-effort streams. One way to honor the high-quality services is to assign different priorities to object requests and implement scheduling techniques that service them in a manner that maximizes profits.

In all examples, deadlines alone are not sufficient to model application requirements for disk requests. At peak system loads, if it is inevitable to discard some of the data requests, it desirable for the disk scheduling algorithm to discard those with lowest priority.

In this paper, we propose a new disk scheduling algorithm that can handle multiple priority requests in addition to disk head movement and request deadline. Our algorithm is based on a heuristics that takes a more relaxed view of priorities. All requests are inserted in a common queue. Our algorithm increases the disk utilization by serving the requests in a SCAN order whenever possible. If it is impossible to meet all deadlines, requests with least priorities are dropped. For our target applications, our scheme offers significant performance advantages when compared with the Multi-queue scheduling scheme (described in Section 3). While we demonstrate that this tradeoff results in superior performance for multimedia servers, it is not necessarily appropriate for those applications that associate an infinitely high price to high priority requests, e.g., military.

The rest of the paper is organized as follow. Section 2 surveys previous studies on deadline driven disk scheduling algorithms. Section 3 describes the traditional Multi-queue scheduling algorithm. Our new disk scheduling algorithm for multiple priority requests is described in Section 4. The performance evaluation and comparison experiments are explained in Section 5. These results demonstrate that our algorithm is a superior alternative to the multi-queue algorithm. Brief concluding remarks are offered in Section 6

2 Related work

Several studies have investigated scheduling of requests in the context of real-time systems, e.g., Abbott and Gracia-Molina [AGM90], Chen et al. [ea91], etc. Others have investigated the issue in the context of multimedia servers, e.g., Reddy and Wyllie [RW94], Gemmell and Christodoulakis [GC92], Kenchammana and Srivastava [KS97]. None of these considered the role of different priority levels associated with different requests.

For multimedia applications, there are two classes of disk scheduling algorithms. The first, termed cycle-based [GM98], guarantees delivery of data in support of hiccup-free display, e.g., Group Sweeping Scheduling (GSS) [CKY93]. Cycle-based algorithms partition requests into groups and sort requests in each group according to the physical location of the data on the disk. They typically do not associate a deadline with the request. Instead, they break a cycle into slots with one slot supporting a single display.

The second class of algorithms, termed deadline driven, provides only soft real-time guarantee. They honor the deadline of the requests but do not guarantee meeting all deadlines. These algorithms along with an appropriate admission control algorithm can give a statistical real-time guarantee. Generally speaking, deadline driven algorithms provide for a higher disk utilization [SM98]. Example algorithms include: Earliest Deadline First (EDF) [SP89], SCAN-EDF [RW94], and

SCAN-RT [KI96]. None of these studies considered the role of different priority levels associated with different requests.

Interactive multimedia applications have motivated their own scheduling algorithms. For example, Chang and Garcia-Molina proposed BubbleUp [CGM97] to minimize initial latency for interactive requests. With this technique, all periodic requests are served in SCAN order, however, when the first request of a new stream arrives, it is inserted at the head of the queue for immediate service. Wijayaratne and Reddy [WR99] proposed a similar algorithm that provides different performance level guarantees for interactive and periodic requests. The algorithm differniates between two types of applications:audio/video applications and interactive applications. Audio and video applications produce sequence of requests (termed "periodic requests") that arrive periodically to the disk with prespecified deadline. Interactive application, e.g., interactive games, issue requests in an unpredictable manner (termed "interactive requests"). The algorithm maintains the queue in SCAN order. It divides the periodic requests into groups and schedules the interactive requests at the head of each group. Once can conceptualize this algorithm as a specialization of our proposed technique where the number of priority levels is limited to two. In addition to supporting muliple (more than two) priority levels, our algorithm strives to schedule requests in a manner that minimizes violation of the SCAN order.

Our proposed technique belongs to the deadline driven class of algorithms. When compared with previous algorithms, the proposed algorithm considers priority of requests from a global perspective. Traditionally, multiple priority requests are organized in multiple queues, one for each priority level [CRL89]. Each queue is sorted separately to optimize the head movement. High priority requests are always serviced prior to low priority requests. They service low priority requests only when higher priority queues are empty. While this achieves optimal performance for highest priority requests, it usually results in (a) a lower disk utilization because there is no global optimization for the seek time, and (b) longer delays for low priority requests, resulting in potential starvation.

3 Multi-queue scheme

Traditional treatment for multiple priority requests is to give strict precedence for high priority requests at the expense of low priority requests [CRL89]. This requires maintaining multiple queues, one for each priority level (see Figure 1.) Within each priority queue, requests can be sorted in different ways.

- One way is to sort the requests in each queue based on their arrival time (First In First Out, FIFO)
- Requests can be sorted according to their deadlines (the time by which the requested data must be serviced). In this case, each queue can be viewed as Earliest Deadline First, EDF.
- Requests are sorted according to their positions on the disk platters in order to maximize the utilization of disk bandwidth. SCAN is a good example of this approach.
- Sorting the requests according to both their deadline and the position on the disk platters, termed SCAN-EDF [RW94].

Requests from a queue of priority p_j are scheduled only if there is no pending request in any queue of priority p_i, where $p_i > p_j$. In general, this scheme results in an optimal performance, in terms of both response time and loss rate, for high priority requests. However, this is at the expense of low priority requests, and when the overall performance of the system is considered, it is not necessarily optimal.

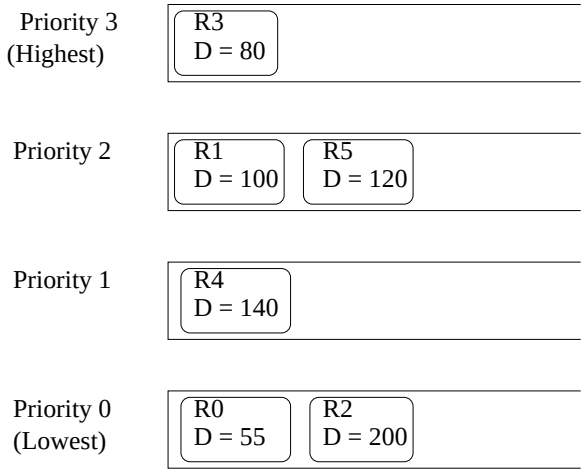


Figure 1. Multi-queue scheme keeps each priority group in a separate queue

In our video-on-demand pricing application, the Multi-queue scheme will try to fulfill high priority requests regardless of how many lower priority requests

miss their deadlines. This might result in a poor overall performance of the system. This is best explained with an example. In Figure 1, it is assumed that there are four different levels of priorities 0-3, where priority 3 is the highest and 0 is the lowest. Each request has its own priority. There are six outstanding requests $R_0 - R_5$ with different deadlines D and priorities as shown in Figure 1. The deadline is the time range (in msec) within which the request should be retrieved. For simplicity, let us assume that each request requires 20 msec of service time (to retrieve the data from the disk). Request R_0, a low priority request, has a deadline of 55 msec. R_0 will not be serviced until R_3, R_1, R_5 and R_4 are all serviced in order. By that time, request R_0 would have been dropped because its deadline has expired.

However, if the scheduling algorithm had scheduled R_0 immediately after R_3 then it would have serviced all requests with no deadline violations. The Multi-queue scheme optimizes for the deadline of those requests that fall in the same priority level. It services the different priority queues in strict order according to their priority level without considering either (a) the deadline of requests at a different priority level or (b) the impact of seeks from a global perspective. In this paper, we propose a new scheme which maintains one queue for all request. It optimizes the disk head movement and attempts to meet the deadline of all requests with the objective to meet the deadline of those requests with the highest priority.

4 The new algorithm: one-queue scheme

In this section we propose a novel deadline driven disk scheduling algorithm that services requests with the objective to maximize the system output bandwidth. Our algorithm attempts to service low priority requests with tight deadlines as long as the deadlines of the high priority requests are not violated. A request is considered to be "lost" if the system fails to meet its deadline. To maximize the disk utilization and minimize the seek time, the queue is maintained in SCAN order whenever possible. The new scheduling algorithm can be conceptualized to provide an on-line solution to an optimization problem with three dimensions (see Figure 2): 1) priority, 2) disk head movement, and 3) deadline.

Let the requests in the disk queue be R_0, R_1, ..., R_n, where R_0 is at the head of the queue. The queue

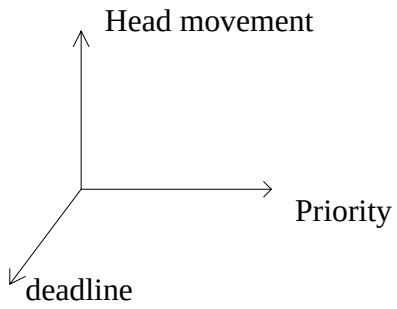


Figure 2. Representation of the optimization space

is composed of one or more SCAN cycles. Requests that belong to one SCAN cycle are sorted in ascending or descending order with respect to the disk head position. We define the state of the scheduling queue to be *valid* if and only if all the requests in the queue are serviced in order with no deadline violations. Initially, the queue is assumed to be in a valid state (none of the deadlines are violated). Let R_{new} be a new request that falls in between R_i and R_{i+1}. After the insertion of R_{new}, the algorithm ensures that the queue remains in a valid state. This is accomplished in an efficient manner. In particular, we do not attempt to enumerate all possible valid schedules in order to choose the best one because such an approach is computationally expensive (NP complete as a function of the number of requests) and unrealistic.

If insertion of R_{new} results in an invalid state (at least one request will loose its deadline), the algorithm takes a set of steps to restore validity with the objective to satisfy as many of the following conditions as possible:

1. insertion of R_{new} should not violate the deadline of a higher priority request,
2. the number of requests that are lost should be minimal,
3. restoration of validity should not sacrifice the bandwidth efficiency by either scrambling up the SCAN order or creating a new SCAN order.

Algorithm outline: We first attempt to tentatively insert R_{new} in SCAN order. If this results in a valid state then the algorithm terminates successfully(ideal scenario).

In the following discussion, we assume that the potential insertion of R_{new} in SCAN order leads to dead-

line violation of one or more requests. There are two possibilities:

- the deadline of R_{new}, itself, is violated,
- or the deadlines of some other requests, already in the queue, are violated.

For the sake of simplicity, let us assume that there is only one deadline violation. Handling deadline violation of multiple requests is a straightforward extension.

If the deadline of R_j (R_j could either be a pre-existing request, or R_{new}, itself) has been violated during the insertion, moving a request from the segment of the queue R_0, R_1, ..., R_{j-1} to the tail of the queue will benefit R_j (either the deadline violation of R_j will disappear, or it will be missed by a shorter amount). However, the migrated requests might have their own deadlines violated because of the migration. Therefore, we carefully choose low priority requests (with relaxed deadlines) and move them back in the queue. This process might push R_{new} itself back in the queue, specially when it has the lowest priority.

The requests in the queue are examined in the order of priorities, lowest first. Requests with identical priorities are examined in the order of deadlines, most relaxed first. Requests are moved to the tail of the queue, one by one, until the schedule becomes valid. It is understood that the request that is pushed back might lose its deadline. Effectively, this leads to low priority requests with relaxed deadlines being migrated to the back of the queue. Thus, the algorithm attempts to rearrange the queue order to avoid possible deadline violation. If requests must be lost, our algorithm drops those with the lowest priority.

Figure 3-a shows an example of a disk queue that contains several requests with different priority values range from one to five. Recall that higher value means higher priority. In this figure, the deadline of each request is denoted as D. Each request requires a fixed service time of 20 $msec$[1]. To insert a new request with priority value = 3, the deadline of R_3 get violated. Hence, the algorithm rearranges requests. Among the requests in front of R_3, R_2 has the least priority and the most relaxed deadline (Figure 3-b.) Therefore, R_2 is moved to the tail of the queue. Figure 3-c shows the queue after reorganization. If R_2 is scheduled successfully at that position, the algorithm terminates. In case the deadline of R_2 is shorter, it is possible that R_2 could be lost and removed from the queue.

The outline of the proposed algorithm is as follow:

1. insert the new request R_{new}, according to the scan order, in the last scan cycle in the disk queue.
2. if no deadline is violated then terminate.
3. if the deadline of a request R_j, R_j might be a pre-existing request or R_{new}, is violated then:
   - find the request with the lowest priority, R_{low}, among all the requests in front of R_j and move it to the tail of the queue (reschedule in cycles behind R_j),
   - if several candidate requests exist then choose the one with the most relaxed deadline,
   - recompute the service time of the affected requests (due to change in seek times),
   - if the deadline of R_{low} is violated, declare it lost and remove it from the queue
4. go to step 2

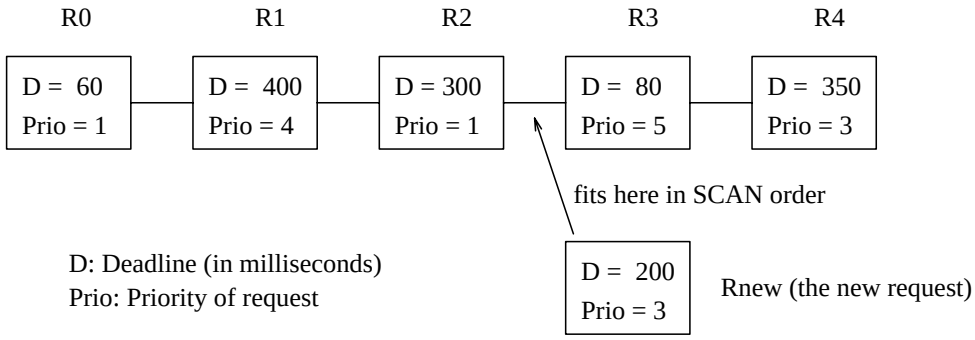
5 Experiments

The details of the experiments undertaken to evaluate the performance of our proposed algorithm, the results, and analysis are described in this section. The experiments are based on a simulator that models PanaViSS server. PanaViSS is a video server built by Panasonic [IT94] for broadcasting applications. The following subsection describes the general architecture of the PanaViSS video server.

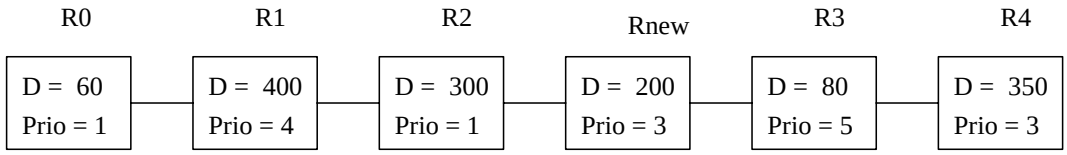
5.1 PanaViSS architecture

PanaViSS video server consists of several building blocks that are connected through an ATM backbone network. The main blocks are: a) the Media Segment File Server (MSFS) that stores the video objects, and b) the Sequence Control Broker (SCB) that controls the reading of the video objects from MSFS and their transmission to display stations. Media Segment File Server (MSFS) and the Sequence Control Broker (SCB) are combined into one object called PanaViSS Unit (PU). The main components of the system architecture are shown in Figure 4.

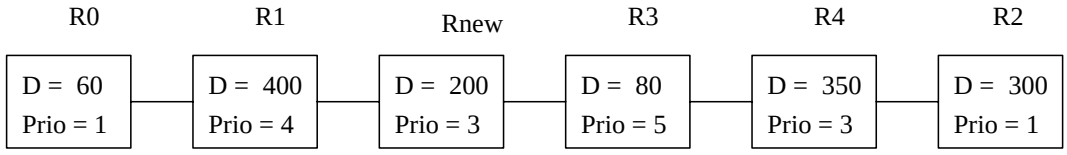
[1] This is a simplification because the service time depends on the seek time which is a function of the disk head position and the physical location of the block referenced by the request.



(a) Before the insert



(b) After the insert: Rnew violates the deadline of R3



(c) After reorganization

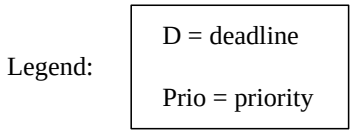


Figure 3. The state of the scheduling queue: (a) before the new request is inserted, (b) after the new request has been tentatively inserted, and it is discovered that it violated the deadline of R_3, and (c) after reorganization

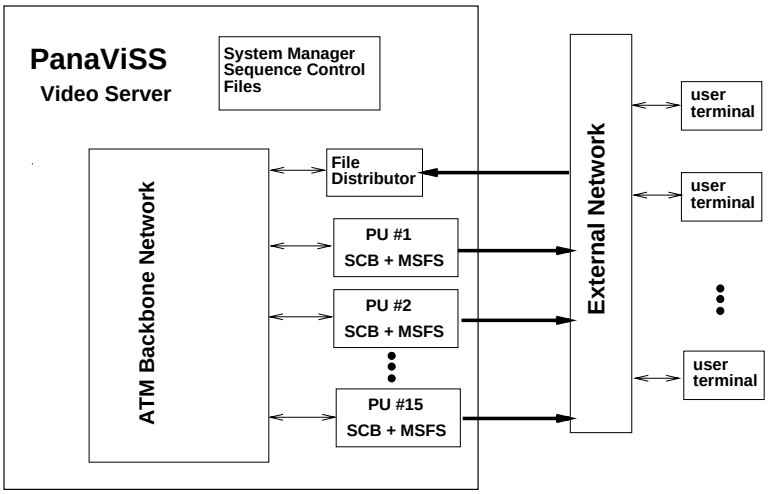


Figure 4. PanaViSS system architecture.

The following is a brief description of the functions of the important components.

System Manager (SM) is the database server. In the current design, an MPEG-encoded video is broken into fixed-length media segment file (MSF) blocks and randomly distributed across all MSFSs. The pointers to all the MSFS are kept in the Sequence Control Broker (SCB), which is managed by the system manager (SM).

File Distributor (FD) is responsible for striping a new movie file evenly across all PUs.

PanaViSS Unit (PU) is the building block of the video server. There are 15 PUs. Each one contains one Sequence Control Broker (SCB) and one Media Segment File Server (MSFS).

Sequence Control Broker (SCB) acts on behalf of users to maintain a video playback stream. During the initialization of the video playback session, the SCB fetches the file table for the requested movie. During the playback, the SCB sends the read requests to the MSFS on behalf of the user in a way to meet the end user capacity and requirements. The SCB is also responsible for handling virtual VCR function requests.

Media Segment File Server (MSFS) stores and retrieves video segment files at the request of the SCBs.

ATM Switch connects all the server components together.

External Network connects the video server to the end users.

User Terminal is the set-top box that is responsible for the following functions: decoding MPEG-encoded video data, providing user interface for virtual VCR function requests and communicating with the SCB.

5.2 Performance results

For experimental purposes, we configured PanaViSS with 15 SCBs and 15 MSFSs (storage units). Each MSFS consists of 4 disks. The parameters of the disk are shown in Table 1. Seek time is modeled as the function $s_1 + s_2 d + s_3\sqrt{d}$. An average rotational latency of half the time taken for one rotation is assumed for each disk request. To implement our new scheduling scheme we used one queue for each disk. To implement the Multi-queue scheme we assumed to have one queue for each priority level.

Disk Parameters	Values
Type	Quantum XP32150
No. of cylinders	3832
Tracks/Cylinder	10
No. of zones	16
Sector size	512 Bytes
Rotation Speed	7200 RPM
Average seek	8.5 mSec
Max seek	18 mSec
Seek cost function	$0.8 + 0.002372(d) + 0.125818(\sqrt{d})$
Disk Size	2.1 GBytes
File Block Size	64 KBytes
Transfer Speed	4.9 - 8.2 MBytes/sec
Disks/RAID	5 (4 data 1 Parity)

Table 1. Disk Model

In the experiments, we assume that 82 to 98 users are simultaneously accessing each of the 15 MSFSs in PanaViSS server. This means that there is more than 1200 users accessing the server simultaneously. Each user requests to either retrieve or download a MPEG-1 stream at 1.5 Mbps. Users send read or write requests every 350 ms. Each request either reads or writes a 64 Kilobyte block. This means that each request must be serviced by the MSFS before a delay limit of 350 milliseconds. A request not serviced prior to this deadline is considered lost. Contiguity of layout is not assumed – data blocks are located randomly across disks (and across the platters in a disk). Details of the simulation model are presented in [JMA95].

The main performance metric that we measure is the fraction of read/write requests that are lost by the MSFS. Naturally, the scheduling algorithm strives to minimize the number of lost requests while respecting the priorities of requests.

The experiments were performed with 2 (numbered 0 and 1) and 4 (numbered 0, 1, 2, 3) priority levels. In the Multi-queue scheme, requests in each of the queues are independently scheduled using the SCAN-EDF algorithm. Priorities are strictly enforced — a request is serviced from a queue only if there are no pending request of a higher priority. The distribution of requests is exponential; the number of requests at one priority level (e.g., say 0) is ten times as many as that of the next higher priority level (e.g., 1). Each experiment was performed for 1 million requests.

Users per MSFS	One-queue		Multi-queue	
	P 0	P 1	P 0	P 1
82	1	0	323	0
84	4	0	600	0
86	6	0	714	0
88	7	0	1117	0
90	22	0	1244	0
92	35	3	1806	0
94	39	3	3341	0
96	60	3	4695	0

Table 2. Loss Rate for Two Priority Levels (P_i= priority i; 0 is the lowest)

Table 2 shows the number of data block losses for the environment with two priority levels. The number of priority 0 requests serviced by the One-queue scheme is approximately two order of magnitude higher than the Multi-queue scheme. Under considerable load (in terms of the number of simultaneous users) both schemes service all priority 1 requests. However, with increased system load, the One-queue scheme drops a few (less than 0.0003% of) high priority requests. This is best explained with an example. Assume a queue that consists of two requests, one from each priority level. Assume that the priority 0 request is ahead of the priority 1 request because of the scan order. When the system starts to service the priority 0 request, multiple high priority requests might collide and reference this disk subsystem (due to a random assignment of blocks), causing one of them to miss its deadline. The Multi-queue scheme prevents this from happening by servicing the high priority requests first, forcing the low priority request to be dropped. Of course the possibility of such occurrences is slim, explaining why the One-queue scheme drops such few high priority requests.

User MSFS	One-queue scheme				Multi-queue scheme			
	P0	P1	P2	P3	P0	P1	P2	P3
82	36	1	0	0	3821	0	0	0
84	67	7	1	0	6555	0	0	0
86	108	6	1	0	9047	0	0	0
88	183	10	0	0	12825	0	0	0
90	322	10	0	0	17989	0	0	0
92	408	10	2	0	26298	0	0	0
94	593	19	1	0	37591	0	0	0
96	847	33	2	0	53577	0	0	0

Table 3. Loss Rate for Four Priority Levels (P_i= priority i; 0 is the lowest)

Table 3 shows the number of data block losses for the case of four priority levels. As in the previous case, the One-queue scheme achieves about two order of magnitude improvement over the Multi-queue scheme in the number of losses of priority 0 requests. Also, the One-priority queue encounters few losses in the higher priority level.

In reality, in applications like those described in Section 1, there is a cost function that governs the relation between the different priority levels. For example, losing a request of priority i costs 5 times as much as losing a request of priority $i-1$ (where i is a higher priority than $i-1$). In this case, the One-queue scheme offers a significant improvement over the Multi-queue scheme [2].

One of the advantages of using the One-queue

[2] However, if losing a request of priority i costs 10,000 times (or infinite for critical application) as much as losing a request of priority $i-1$ then the Multi-queue scheme is more appropriate.

scheme is the increased throughput achieved due to longer SCAN cycles. This also has an impact on the loss rate, and response time. Table 4 shows the improvement achieved in the average time spent on disk seeks per request.

Another important metric is the response time of a single request – i.e., how long does it take on the average, for the file server to service a read request? Tables 5 and 6 show the average response time for the two priority levels case and for the four priority levels case respectively. As it can be seen, a strict enforcement of priorities (as in the Multi-queue scheme) leads to a higher response time for the lower priority requests, and hence a higher cumulative average response time. It is interesting to notice that the One-queue scheme results in comparable response time for all priority levels. Also the One-queue scheme provides lower average response time than the Multi-queue scheme.

Users/MSFS	One-queue	Multi-queue
82	5.30	6.11
84	5.20	6.04
86	5.15	5.98
88	5.06	5.91
90	4.97	5.84
92	4.92	5.79
94	4.84	5.72
96	4.76	5.65

Table 4. Average Seek Time per Request for Four Priority Levels in milliseconds (P_i= priority i; 0 is the lowest)

6 Conclusions

In this paper we propose a new disk scheduling algorithm in support of requests with multiple priorities. Traditional Multi-queue schemes maintains separate queue for each priority level and service low priority requests only when higher priority queues are empty. They strive to satisfy the deadline of high priority requests primarily due to their infinitely high price. Our proposed algorithm is based on one queue and services low priority requests whenever the deadlines of higher priority requests permit. Our method improves the system output bandwidth and increases the number of serviced low priority requests one- two order of magnitude when compared with the Multi-queue scheme. Under some conditions the proposed algorithm might violate the deadline of a few high priority requests (less than 5 in a million).

For those applications, e.g., video-on-demand and digital libraries, that associate a bounded price to high priority requests, our proposed algorithm is a clear winner.

References

[AGM90] R. K. Abbot and H. Gracia-Molina. Scheduling I/O Requests with Deadlines: A Performance Evaluation. In *IEEE Real Time Systems Symposium*, 1990.

[CGM97] E. Chang and H. Garcia-Molina. BubbleUp: Low Latency Fast-Scan for Meida Servers. *Proc. of the 5th ACM Conference on Multimedia*, June 1997.

[CKY93] M. Chen, D. Kandlur, and P. Yu. Optimization of the Grouped Sweeping Scheduling (GSS) with Heterogeneous Multimedia Streams. In *First ACM International Conference on Multimedia*, August 1993.

[CRL89] M. Carey, R.Jauhari, and M. Livny. Priority in DBMS Resourse Scheduling. In *International Conference on Very Large Data Base*, Amsterdam, Neatherland, 1989.

[ea91] S. Chen et al. Performance Evaluation of Two New Disk Scheduling Algorithms. *Real time systems*, (3):307–336, 1991.

[GC92] J. Gemmell and S. Christodoulakis. Principles of Delay-Sensitive Multimedia Storage and Retrieval. *ACM Transaction of Information Systems*, (10):51–90, 1992.

[GM98] S. Ghandeharizadeh and R. Muntz. Design and Implementation of Scalable Continuous Media Servers. *Parallel Computing*, 24:91–122, June 1998.

[IT94] Y. Ito and T. Tanaka. A video server using atm switching technology. In *The 5th International Workshop on Multimedia Communication*, pages 341–346, May 1994.

[JMA95] S. Johnson, V. Mani, and R. Alonso. Video server simulation: Users' manual. PINTL Technical Report #138, Panasonic Information and Networking Technology Laboratory, July 1995. Ph.D. Dissertation.

Users /MSFS	One-queue scheme			Multi-queue scheme		
	P0	P1	Avg	P0	P1	Avg
82	72.5	72.6	72.5	89.5	37.6	85.3
84	74.8	75.1	74.8	92.7	37.6	88.3
86	76.1	76.4	76.1	94.9	37.7	90.3
88	78.3	78.3	78.3	98.4	37.2	93.3
90	80.7	81.0	80.7	101.8	37.3	96.4
92	82.0	82.5	82.1	104.0	37.7	98.6
94	84.3	84.6	84.4	107.6	37.9	101.8
96	86.7	86.5	86.7	111.2	37.8	105.1

Table 5. Average Response Time for Two Priority Levels in milliseconds (P_i= priority i; 0 is the lowest)

Users /MSFS	One-queue scheme					Multi-queue scheme				
	P0	P1	P2	P3	Avg	P0	P1	P2	P3	Avg
82	72.5	72.5	72.4	72.8	72.5	90.4	37.3	35.7	35.7	85.6
84	74.7	74.7	74.9	74.3	74.7	93.8	37.4	35.8	35.8	88.7
86	76.1	76.1	76.0	76.5	76.1	96.0	37.5	35.8	35.9	90.7
88	78.4	78.3	78.2	78.8	78.3	99.3	37.6	36.0	35.7	93.7
90	80.7	80.7	80.6	80.6	80.7	102.6	38.2	36.4	36.3	96.9
92	82.1	82.1	81.9	81.7	82.1	105.4	37.8	36.0	35.8	99.2
94	84.4	84.4	84.3	85.4	84.4	109.0	37.9	36.1	35.9	102.4
96	86.8	86.8	86.5	85.8	86.8	112.6	38.0	36.1	36.0	105.7

Table 6. Average Response Time for Four Priority Levels in milliseconds (P_i= priority i; 0 is the lowest)

[KI96] I. Kamel and Y. Ito. A proposal on disk bandwidth definition for video servers. In *The 1996 Society Conference of IEICE*, September 1996. Also available as Tech. Report Matsushita Information Tech Lab TR-153-96.

[KS97] D. Kenchammana and J. Srivastava. I/O Scheduling for Digital Continous Media. *ACM Multimedia Systems*, 5:213–237, 1997.

[LS94] A. Y. Levy and A. Silberschatz. Challenges for Global Information System. *Proc. of International Conf. on VLDB*, 1994.

[RS97] K. Ramamritham and N. Soparkar. Report on DART '96: Databases: Active and Real-Time (Concepts meet Practice). *ACM SIGMOD Record*, pages 51–53, 1997.

[RW94] A.L. Narasimaha Reddy and Jim Wyllie. Disk Scheduling in a Multimedia I/O System. In *Proc. of the First International Conf on Multimedia*, pages 225–233, August 1994.

[SM98] J. R. Snatos and R. Muntz. Performance Analysis of the RIO Multimedia Storage System with Heterogeneous Disk Configurations. *Proc. of 6th ACM International Multimedia Conference*, 1998.

[SP89] A. Silberschatz and J. Peterson. *Operating System Concepts*. Addison Wesley, 1989.

[SSH99] J. Stankovic, S. H. Son, and J. Hansson. Misconceptions About Real-Time Databases. *IEEE Computer*, 32(6):29–36, June 1999.

[WR99] R. Wijayaratne and N. Reddy. Integrated QOS Management for Disk I/O. *Proc. of IEEE Multimedia Systems*, pages 487–492, June 1999.