A Cost Driven Disk Scheduling Algorithm for Multimedia Object Retrieval *

Shahram Ghandeharizadeh and LiGuo Huang
Computer Science Department
University of Southern California
Los Angeles, California 90089
shahram,liguo@usc.edu

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

January 4, 2002

Abstract

This paper describes a novel cost-driven disk scheduling algorithm for environments consisting of multi-priority requests. An example application is a video-on-demand system that provides high and low quality services, termed priority 2 and 1, respectively. Customers ordering a high quality (priority 2) service pay a higher fee and are assigned a higher priority by the underlying system. Our proposed algorithm minimizes costs by maintaining one-queue and managing requests intelligently in order to meet the deadline of as many priority 1 requests as possible while maximizing the number of priority 2 requests that meet their deadline. Our algorithm is general enough to accommodate an arbitrary number of priority levels. Prior schemes, collectively termed "multi-queue" schemes maintain a separate queue for each priority level in order to optimize the performance of the high priority requests only. When compared with our proposed scheme, in certain cases, our technique provides more than one order of magnitude improvement in total cost.

*This research was supported in part by NSF grant IIS-0091843.

1 Introduction

Both disk and deadline driven scheduling[1] techniques have been an active area of research for several decades. They span disciplines such as operating system, database management systems and multimedia information systems to name a few. While a typical disk scheduling strategy strives to maximize the amount of useful work performed by the disk drive, a typical deadline driven scheduling attempts to maximize the number of requests that meet their deadlines. An increasing number of data intensive applications, e.g., multimedia, require cost driven scheduling which is a variant of deadline driven disk scheduling algorithms. These applications associate a deadline, priority and cost to each request. The definition of these terms is as follows:

- **Deadline** is the absolute time by which a request must be serviced. If the scheduler fails to service a block request by its deadline, this request is considered lost and ignored from further consideration. This might cause the display of a video or audio clip to suffer from disruptions or delays, termed hiccups. Below, we expand on this to detail a taxonomy consisting of two types of requests and the consequences of discarding a block request.

- **Priority** is a conceptual ranking to show the relative importance of a request when compared with other requests. For example in our video-on-demand example (see abstract and the following paragraphs), request for high quality streams are more important, and have a higher priority than those of the low-quality streams.

- **Cost** of a request is the value lost if the scheduler fails to meet the deadline of this request. It is derived based on a number of factors, of which priority is one. We assume (a) cost is a fixed value that does not change as a function of time, and (b) there is a linear correlation between priority and cost, i.e., when compared with a low priority request, a high priority request has a higher cost.

A server employing a scheduling mechanism might loose a certain number of requests during a period of time. Their total cost is the cost incurred by the scheduling algorithm (and the server) during this period. Obviously, the objective of a scheduling algorithm is to minimize this total cost.

To minimize total cost, it might be appropriate for the scheduling algorithm to sacrifice a high priority request in favor of servicing several low priority requests. This is best demonstrated with an

[1] Also termed real-time scheduling.

example: With a video-on-demand application, a service provider may offer a range of services with an increased fee for higher quality service. For example, there might be two service levels: high-quality (priority 2) and low-quality (priority 1). Assume customers ordering a high-quality service pay 10 dollars per request while those ordering low-quality service pay 1 dollar per request. While priority 2 requests are more important than priority 1 requests, it would be more cost effective to sacrifice one priority 2 request in favor of servicing fifteen priority 1 requests. Other example applications that can benefit from the proposed paradigm are interactive applications that strive to minimize the startup latency of an application by associating a high cost and priority to the first block request.

This paper extends on our prior work to describe a cost-driven scheduler for magnetic disk drives and our insights gained by experimenting with this scheduler. We focus on two kinds of requests: aperiodic and periodic. Aperiodic requests pertain to traditional media such as text and still images. These requests arrive in a random manner and do not exhibit a regular pattern. Traditionally, the arrival time of these requests are modeled using a poison distribution. While these requests have no deadlines, they must be serviced in a timely manner that prevents starvation. A general rule of thumb is for the system to service these requests in 100 milliseconds [Den68].

Periodic requests correspond to those that produce block requests based a regular arrival pattern. They represent requests for continuous media, audio and video clips. While the client request for a clip arrives in a random manner (and is also modeled using a poison arrival rate), once the display of the clip starts, the client generates data block requests based on a regular schedule dictated by the display bandwidth requirement of the object[2]. This schedule associates a deadline with each block request. If a block request is not serviced within its deadline, it is considered lost, resulting in a possible discontinuity at a display station. Such disruptions and delays are termed hiccups.

For both request types, a display looses information when its block request misses its deadline. Hence, block loss rate is expected to be an infrequent event. With periodic requests, for example, several studies assume the loss rate to be 1 in a million requests [GK00, SMRN00]. This is accomplished by sizing the system according to the operator's peak system load expectations, and not accepting additional requests once the peak system load is reached (using admission control policies).

[2]Assuming a pull paradigm. With a push paradigm, the server might generate block requests internally. Both cases have the same affect; block requests are generated based on a schedule dictated by the display rate of the clip.

We propose an algorithm that maintains only one queue for all requests. It schedules requests based on priority, cost and deadline while maximizing the useful utilization of the disk drive. We compare this algorithm with a multi-queue algorithm that (a) maintains a different queue for each priority level, and (b) services requests in a high priority queue first. The results demonstrate the superiority of the one queue algorithm because it can both service a large number of low priority requests and satisfy the deadline of high priority requests. In addition, the one-queue algorithm increases useful utilization of the disk drive. With high system loads, our algorithm behaves almost identical to the multi-queue algorithm because it is best to service high priority requests only.

The main intuition here is that our one queue algorithm minimizes cost by scheduling requests based on a global view of the pending requests. This view enables it to service as many low-priority requests as possible while meeting the deadline of high-priority requests. A multi-queue scheduling algorithm services requests based on a partial view because, by definition, it is forced to service requests from one priority queue at a time. With certain system loads, our one queue algorithm provides 25 time savings when compared with the multi-queue algorithm.

When the deadline of a block is a function of its size, our simulation results demonstrate that the loss rate of the system is dependent on the service time of each request. With disks, the block size dictates the service time of a request. Given a fixed system load, the loss rate is minimized with certain block sizes. (This applies to both the one-queue and multi-queue algorithms.) A secondary contribution of this paper is its use of the one-server birth-death model M/M/1/K [GH98] to analytically model this minimum.

The rest of this paper is organized as follows. Section 2 introduces our new one-queue algorithm. Our simulation model and obtained results are contained in Section 3. An analytical model that captures the essence of our obtained simulation results is also presented in this section. Section 4 surveys prior studies and how they relate to our proposed algorithm. Brief conclusions and future research directions are detailed in Section 5.

2 The New Algorithm: One-queue Scheme

The proposed disk scheduling algorithm services requests with the objective to minimize the total incurred costs. In our one-queue algorithm, the application assigns a value to each user request according to its priority as detailed in the previous section. We assume a linear relationship between

cost and priority. Our algorithm attempts to service low priority requests with tight deadlines while meeting the deadline of high priority requests. Upon the arrival of a request whose deadline cannot be met, the algorithm drops one or more low priority requests. Those requests are selected carefully in a way to minimize the total cost incurred by the server. To maximize useful disk utilization, the disk queue is maintained in a SCAN order whenever possible. SCAN minimizes the seek time by sorting requests based on their position on the disk platters, increasing the amount of time a disk spends transferring data. The new scheduling algorithm can be conceptualized to provide an on-line solution to an optimization problem with four dimensions: 1) priority, 2) seek time, 3) deadlines, and 4) cost. We assume that priority and cost are linearly correlated. Hence, one may reduce the number of dimensions to three by replacing either priority or cost.

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 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. Two adjacent SCAN cycles are sorted in reverse order.

We define the state of the scheduling queue to be *valid* if and only if all requests in the queue are serviced with no deadline violations. Initially, the queue is assumed to be in a valid state (all the deadlines are satisfied). 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. If insertion of R_{new} results in an invalid state (at least one request misses its deadline and R_{new} itself might be this request), the algorithm performs 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 total cost incurred by the server as a result of dropped requests should be minimized.
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). But the potential insertion of R_{new} in SCAN order may lead to deadline violation of one or more requests. There are two possibilities: either (1) the deadline of R_{new} itself is violated, or (2) the deadline of some other requests following R_{new} are violated.

The disk queue is assumed to be in the valid state before applying our algorithm. The outline of the one-queue algorithm is as follow:

1 Attempt to insert the new request R_{new} into the last scan cycle in the disk queue according to the scan order of that cycle;
2 Compute the service time (transfer time + seek time + rotational latency) of all the requests from R_{new} to R_n; check deadline violations from R_{new} to R_n;
3 while (there exists a request R_j whose deadline is violated): {
4 Find the request with the lowest priority, R_{low}, among all the requests in front of R_j (including R_j); if several candidates exist then choose the one with the most relaxed deadline;
5 Move it to the tail of the queue and reschedule R_{low} in a new scan cycle whose scan order is the reverse of the prior cycle;
6 Recompute the service time of all the requests following R_{low}, including R_{low};
7 if the deadline of R_{low} is violated, then: {
Remove R_{low} from the queue;
Declare R_{low} lost;
total cost += cost(R_{low});
}
8 if (total cost >= cost of loosing (R_{new})) then: {
Remove R_{new} from the queue and restore the queue to its initial state before R_{new} was inserted;
Declare R_{new} lost;
break;
}
9 Check if there is a request with a violated deadline in the disk queue;
10 } – end of while;
11 algorithm terminates.

In order to restore the validity of the disk queue, the requests in the queue are examined one at a time, lowest priority first. Requests with identical priority are examined in the order of deadlines, most relaxed first. The lowest priority requests are moved to the tail of the queue, one by one, until the schedule becomes valid. It is understood that those requests that are pushed back might lose

their deadlines. If the total cost incurred by the server to restore the validity of the queue exceeds or equals the value of the new request R_{new}, then it will abort inserting R_{new} in order to minimize total cost. Thus, our one-queue algorithm attempts to intelligently rearrange the queue in order to avoid possible deadline violations and try to minimize the server total cost and maximize overall system performance.

3 A Comparison

This section contains an experimental evaluation of the one-queue algorithm for aperiodic and periodic requests. As a comparison yard stick, we include the evaluation of a multi-queue scheme. The rest of the section is organized as follows. We start with a description of the multi-queue algorithm. Section 3.2 describes our experimental model and Section 3.3 contains our obtained results and observations.

3.1 Multi-queue Scheme

A multi-queue scheme maintains each priority group in a separate queue. In addition to SCAN, see Section 2, the system may employ alternative policies to service requests that constitute each priority queue.

- First In First Out, FIFO: Service the requests in each queue based on their arrival time.
- Earliest Deadline First, EDF: Service requests according to their deadlines (the time by which the requested data must be serviced).
- CSCAN: With this algorithm, the disk head scans for requests in one direction from the outermost track to the innermost track or vice versa, serving requests in the order of its scan direction. If it is scanning inward, after it services the innermost requests the head moves to the outermost pending request. This policy does seek optimization while guaranteeing that no request starves [RW94].
- SCAN-EDF: Service requests according to both their position on the disk platters and their deadline.

In this paper we utilize SCAN-EDF because Reddy and Wyllie evaluated and compared this algorithm with CSCAN and EDF to demonstrate its superiority [RW94]. Similar to this study, they investigated two types of requests: periodic and aperiodic. A periodic request can be denoted by two parameters (c, p), where p is the period at which the periodic requests are generated and c is the service time required in each period. All periodic requests have the deadlines that are multiples of the period p and all requests in one periodic request stream arrive with constant data rate. Aperiodic requests are assumed to arrive with an exponential distribution. They demonstrated that CSCAN supports periodic requests well but not aperiodic requests. EDF does not support periodic requests well but provides good response time for aperiodic requests. SCAN-EDF performs well for both request types. It supports almost as many periodic streams as CSCAN and at the same time offers the best response times for aperiodic requests.

The multi-queue scheme services high priority requests prior to low priority requests. It considers neither (a) the deadlines of requests at a low priority level nor (b) the impact of seeks across different priority queues. It services low priority requests only when higher priority queues are empty. In general, this scheme results in the best response time and loss rate for high priority requests. However, this is at the expense of low priority requests which suffer from either long delays or potential starvation. In many cases, the multi-queue scheme loses too many low priority requests in order to satisfy the deadline of one high priority request. It gives strict precedence for high priority requests without considering the total cost incurred by dropped requests. Experimental results in Section 3.3 demonstrate the superiority of the one-queue algorithm.

3.2 Experimental Model

Our experimental study is based on an open simulation model where user requests are modeled by employing the Poisson distribution with the mean arrival rate λ (requests/sec). The interarrival time T between successive user requests is modeled using the following function:

$T = \lfloor (1/\lambda) \times ln(P) \times 1000 \rfloor$

T: interarrival time (msecs);

λ: mean arrival rate (requests/sec);

P: a random number uniformly distributed between 0.0 and 1.0.

The interarrival time T follows the exponential distribution. We manipulate λ to simulate alternate system loads. The block size (KB) varies from 32KB, 64KB, 128KB, to 256KB. The database consists of a single media type with a 4.0Mbps bandwidth requirement. A request not serviced prior to its deadline is considered lost.

Our simulation model is flexible enough to support n (numbered $1, 2, ..., n$) priority levels. Requests of priority K are more important than priority $k-1$, where $k \geq 2$. The one-queue algorithm schedules disk requests based on an aggregate cost model. Its details are as following: let $loss_i$ denotes the total number of priority i requests whose deadlines are violated. Assuming that C_i denotes the cost of each priority i request, then the total cost (\$) is as follows:

$$Total\ Cost(\$) = \sum_{i=1}^{n} C_i \times loss_i$$

In our simulation model, the server consists of only one disk. The disk is modeled after a year 2000 IBM Ultrastar 36ZX high capacity 3.5-inch disk drive. Its parameters are shown in Table 2.

Disk Parameters	Values
Type	IBM Ultrastar 36ZX
Disk Size	36 Gbytes
No. of Cylinders	11748
Tracks/Cylinder	10
No. of Zones	15 (0–14)
Sector Size	512 Bytes
Rotational Speed	10000 RPM
Single Cylinder Seek Time	0.4 msecs (Read)
Average Seek Time (Weighted)	5.4 mescs (Read)
Max Seek Time	11.4 msecs (Read)
Seek Cost Function	$0.35 + 0.000446381 \times d + 0.0516916 \times \sqrt{d}$
Data Transfer Rate	23.27–44.31 Mbytes/sec

Table 1: Disk Model

The disk seek time is a function of the distance traveled by its arm. Its analytical model of seek time is a nonlinear function:

$$Seek(d) = S1 + S2 \times d + S3 \times \sqrt{d}, S1 = 0.35, S2 = 0.00046381, S3 = 0.0516916$$

$Seek(d)$ denotes the time required for the disk arm to travel d cylinders to reposition itself from cylinder i to cylinder i + d or i - d.

An average rotational latency is modeled as one half the time required for one disk rotation. In our studies, the analytical model of rotational latency is a function of disk rotational speed (Rotations Per Minute, RPM): $Rotational\ Latency = 0.5 \times RPM \times 60 \times 1000$.

3.3 Experimental Results

We conducted many experiments for aperiodic and periodic requests respectively. And those experiments were performed with multiple priority levels, different mix of requests, and alternative costs for the different priority levels. Based on those results, we made several key observations. Next, we identified the simplest experimental study that embodied all our observations for the purpose of this presentation. The following details our study and all our observations.

The experiments are performed with 2 (numbered 1 and 2) priority levels. Loss of priority 2 requests cost twice that of priority 1 requests, i.e., if each priority 1 request costs 1 dollar then each priority 2 request costs 2 dollars. In our mix of requests, priority 1 requests occur ten times more frequently than priority 2 requests. The disk block size is 64KB. Our experiments are performed for both aperiodic and periodic requests. For periodic requests, each request references a movie clip consisting of one hundred 64KB blocks with a constant consumption rate of 4.0 megabits per second.

We considered workloads consisting of different mixes of periodic and aperiodic requests. The observed trends are identical in all cases. In this study, we present results form the two ends of the spectrum: a) 100% periodic and 0% aperiodic, and b) 0% periodic and 100% aperiodic. Even in those two cases, the final observations are identical. Thus, for the rest of the presentation, we report on the results for workload (a). Appendix A contains results obtained for workload (b).

3.3.1 Impact of System Load for Aperiodic Requests

We used both the average system response time and the total incurred cost to compare our proposed one-queue algorithm with the multi-queue algorithm. The cost incurred by each algorithm is presented in Figures 1.a and 1.b. The x-axis in both figures corresponds to the system load, i.e., the mean arrival rate of requests, λ. In Figure 1.a, the y-axis corresponds to the total cost of each algorithm. The y-axis in Figure 1.b shows the percentage difference between these two algorithm, computed as follows:

$$\text{Percentage Difference of Total Cost (\%)} \quad = \quad \frac{Total\ Cost_{mQ} - Total\ Cost_{1Q}}{Total\ Cost_{1Q}}$$

A positive (negative) value in Figure 1.b demonstrates the superiority (inferiority) of the one-queue algorithm. The y-axis of this figure is log scale. The mean arrival rate of Figure 1.b starts with 80 because the loss rate of the one-queue algorithm is zero for low system loads, causing the percentage difference to become infinite. With a low system load ($\lambda \leq 40$), there is not much difference between the two algorithms because there are no requests in the disk queue. With a moderate to a high system load ($50 \leq \lambda \leq 140$), a queue starts to form with both algorithms. In this study, the one queue algorithm is clearly the superior strategy because it manages the disk queue intelligently and tries to service as many priority 1 requests with tight deadlines as possible without violating the deadline of priority 2 requests. The multi queue algorithm does not perform well because it sacrifices many priority 1 requests in favor of servicing priority 2 requests. With system loads that utilize 100% of system resources, both algorithms provide comparable performance because it is most cost effective to service priority 2 requests first. (Figure 1a does not show total cost of $\lambda > 180$ because it would un-necessarily increase the scale of the y-axis, making it difficult to see the important parts of this figure.)

With a high system load, the slightly lower performance (recall the log scale for y-axis) of the one queue algorithm can be explained in three possible ways: First, it is small enough, less than 10%, to be experimental error. Second, it can be attributed to rare adverse scenarios such as the following: Consider a queue with 2 requests, one of each priority level. The deadline of priority 2 request is relaxed enough to motivate the one queue algorithm to start servicing the priority 1 request. Once this happens, the workload generator issues many priority 2 requests in a manner that prevents the disk drive from meeting their deadlines. In these cases, the priority 1 request cannot be preempted because the disk drive has started to service this request. This causes a priority 2 request to loose its deadline. Of course, such scenarios are rare, explaining the small negative percentage difference.

A combination of these two reasons might be a third possible explanations.

With a moderate system load, a side effect of servicing more priority one requests with the one-queue algorithm is a higher average wait[3] time, see Figure 2.a. This is because this algorithm queues up a larger number of requests in a manner that meets their deadlines. With longer queues, see Figure 2.b, the average wait time increases. The multi-queue algorithm provides a lower wait time because it rejects a larger number of priority 1 requests, reducing the overall average delay incurred by the pending requests. The longer wait-time is acceptable because it does not exceed the request deadlines.

3.3.2 Impact of Block Size on Loss Rate

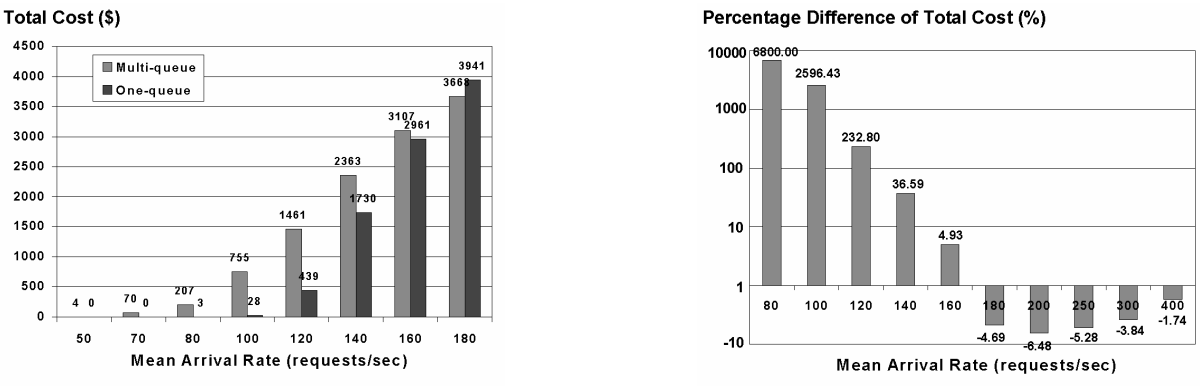
With a fixed system load, we observed that the block size impacts the percentage of lost requests when the deadline of each block request is dictated by a specific consumption rate. This is because a larger block size increases both the disk service time and the deadline of the request[4]. The latter is dictated by the consumption rate of the block while the former is governed by the physical disk characteristics. Thus, there is always a block size that incurs the lowest loss rate for a scheduling algorithm: both multi-queue and one-queue. To illustrate, Figure 3 shows the total number of data block losses as a function of different block sizes with two priority levels. As we increase the block size from 32KB to 64KB, the total number of lost requests drops with the one-queue algorithm. It starts to increase when we increase the block size from 128KB to 256KB.

We can derive a general analytical model to estimate the block size that minimizes the loss rate. This model is based on the one-server birth-death model M/M/1/K [GH98] which limits the number of requests in the system to K. The intuition behind this model is as follows: We derive the probability of K requests in the system when a new request is submitted to the system as a function of the block size, termed $P_k(b)$. (Note that a maximum of K requests can be in the system because this is the peak system load.) By choosing a block size that minimizes this probability, we can minimize the total cost. This is accomplished by setting the first derivative of $P_k(b)$ to zero and solving for the block size that minimizes the number of lost requests.

The following analytical models capture neither the different priority levels nor the cost associated with each priority. Hence, they cannot capture the total cost of a queuing algorithm. Their only

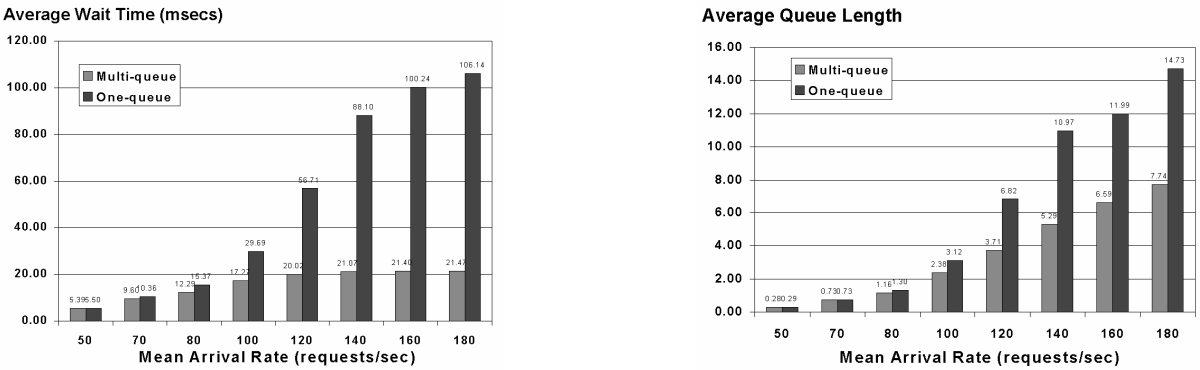
[3]The delay incurred from when a request is issued until it is serviced.

[4]This is specially true for the periodic requests where a block has a pre-specified consumption rate.



(a) Total cost **(b) Percentage difference**

Figure 1: Total cost and percentage difference of a function of system load



(a) Average wait time **(b) Average queue length**

Figure 2: Average wait time and average queue length as a function of the mean arrival rate

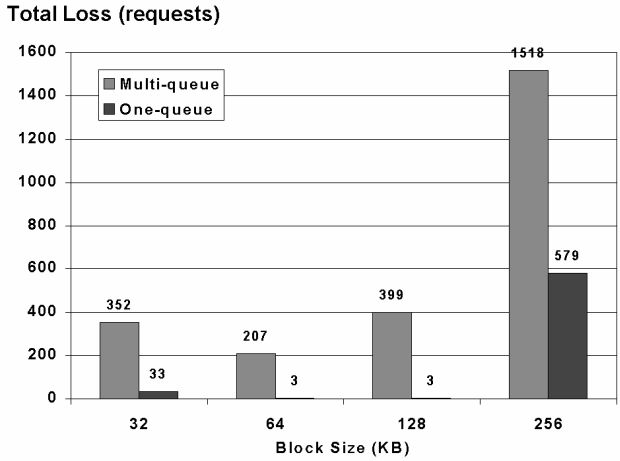


Figure 3: Number of lost aperiodic requests as a function of block size

purpose is to approximate a block size that minimizes the loss rate of a disk scheduling algorithm. As such, they apply to both the one-queue and the multi-queue algorithms.

Using the one-server birth-death model M/M/1/K queuing model, the probability that there are K requests in the system queue when a new request arrives as a function of block size b is:

$$p_K(b) = \frac{\frac{1}{(0.003+s+\frac{b}{tr})\lambda} - 1}{(\frac{1}{(0.003+s+\frac{b}{tr})\lambda})^{1+\frac{2b}{(0.003+s+\frac{b}{tr})\lambda}} - 1} \quad (1)$$

b : block size (MB);

λ : mean arrival rate (requests/sec);

s : seek time (secs);

tr : average transfer rate (MB/s);

The experimental results of Figure 3 assume an arrival rate (λ) of 80 requests per second, a transfer rate (tr) of 35.68 MB/second, and average seek time (s) of 0.0043secs:

$$p_K(b) = \frac{\frac{12.5}{7.3+28.0269b} - 1}{(\frac{12.5}{7.3+28.0269b})^{1+\frac{2000b}{7.3+28.0269b}} - 1} \quad (2)$$

The derivative of $p_K(b)$ is:

$$\begin{aligned} p'_K(b) = & -\frac{350.336}{(7.3+28.0269b)^2((\frac{12.5}{7.3+28.0269b})^{1+\frac{2000b}{7.3+28.0269b}} - 1)} - \\ & ((\frac{12.5}{7.3+28.0269b} - 1)((\frac{12.5}{7.3+28.0269b})^{1+\frac{2000b}{7.3+28.0269b}} \\ & (\frac{2000}{7.3+28.0269b} - \frac{56053.8b}{(7.3+28.0269b)^2})\log(12.5) + \\ & (\frac{12.5}{7.3+28.0269b})^{1+\frac{2000b}{7.3+28.0269b}}((\frac{2000}{7.3+28.0269b} - \frac{56053.8b}{(7.3+28.0269b)^2})\log(\frac{1}{7.3+28.0269b}) - \\ & \frac{28.0269(1+\frac{2000b}{7.3+28.0269b})}{7.3+28.0269b})))/((\frac{12.5}{7.3+28.0269b})^{1+\frac{2000b}{7.3+28.0269b}} - 1)^2 \end{aligned} \quad (3)$$

By setting $p'_K(b)$ to zero, we can solve for the block size that minimizes the loss rate, $b = 0.084MB$.

Equation 1 is derived from the following six equations. First, the mean service time of the disk drive (T) is a function of the average transfer time (t), seek (s) and rotational latency of the disk drive. The transfer time is a function of the block size (b MB) and the transfer rate of the disk drive

(tr MB/second), hence:

$$T = \frac{b}{tr} + s + \text{rotational latency} \quad (4)$$

The deadline of a block (d) is a function of its size (b) and consumption rate of its media type (r MB/second):

$$d = \frac{b}{r} \quad (5)$$

The mean service rate (μ) of the disk is a function of its mean service time:

$$\mu = \frac{1}{T} \quad (6)$$

And, the maximum number of requests that the disk can service is:

$$K = \frac{d}{T} \quad (7)$$

Let ρ denote the traffic density and p_n denote the probability that there are n requests in the system queue:

$$\rho = \frac{\lambda}{\mu} \quad (8)$$

$$p_n = \begin{cases} \frac{(1-\rho)\rho^n}{1-\rho^{K+1}} & (\rho \neq 1) \\ \frac{1}{K+1} & (\rho = 1) \end{cases} \quad (9)$$

We can substitute the appropriate definition of K, ρ and n in the right hand side of p_n and solve to obtain Equation 1.

This discussion applies when the deadline of a block is a function of its size. If one assumes a fixed system load with video clips consisting of n blocks (fixed display time), then the total loss as a function of block size would be different, see Figure 4. This is because a video clip with a fixed display time consists of a fixed number of blocks. When we increase its block size, the number of blocks that constitute the clip decreases. This also reduces the percentage of wasted disk bandwidth, see Figure 5. By increasing the useful utilization of a disk drive, the number of lost requests decreases. If one maintains a fixed number of blocks for a video clip while increasing its block size, then the trends observed in Figure 3, including the analytical models, apply.

4 Related Work

Several studies have investigated disk request scheduling for both real time systems and multimedia systems [AGM92, CG96, CSKT91, HCL91, LT91, RS94, TH99]. We use four dimensions to categorize

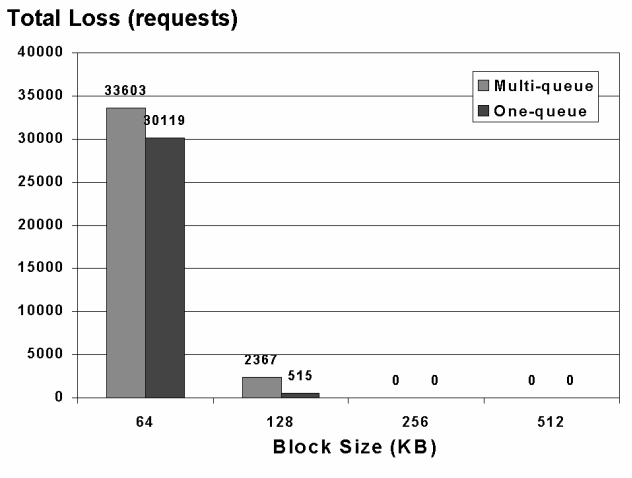


Figure 4: Total loss of periodic requests as a function of the block size, mean arrival rate = 90 requests/min. Clip display time is fixed at 1,000 seconds.

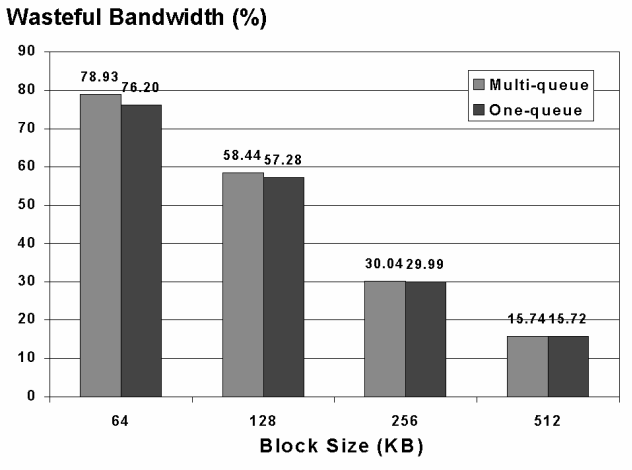


Figure 5: Wasteful bandwidth as a function of the block size for periodic requests, mean arrival rate = 90 requests/min. Clip display time is fixed at 1,000 seconds.

these studies: 1) priority, 2) disk scheduling, 3) deadline, and 4) cost, see Table 1. A subset of these, namely [CG96, CSKT91, TH99, AGM92, LT91, RS94], differ because they do not investigate either priority or cost of requests.

Related Work	Priority	Disk Scheduling	Deadline	Cost
[ABZ96]		√		
[AGM92]	√		√	
[CG96]		√		
[CGM97]		√		
[CJL89]	√			
[CKY93]		√		
[CLea88](The HiPAC Project)	√		√	
[CSKT91]		√	√	
[GC92]			√	
[HCL91]	√		√	√
[KNG00]	√	√	√	
[LT91]	√		√	
[RS94]	√		√	
[RW94]		√	√	
[TH99]			√	
[WR99]	√	√	√	

Table 2: Categorization of previous related work on scheduling

Several studies investigated priorities in transaction processing systems (TPS) [AGM92, HCL91, LT91, RS94]. These studies considered the role of deadlines and priorities in system components in support of concurrency control. They assumed advance knowledge of neither the transaction execution time nor its expected resource consumption. Hence, they considered scenarios where a transaction consumes a certain amount of system resources and aborts because its deadline expired. These environments are different than our assumed environment because our scheduler estimates the service time of a block with high accuracy. It does not schedule a request unless its deadline can be satisfied. It may reject a queued request, however, it does not utilize resources on behalf of such a request. One limitation of our environment is that requests are not pre-emptible. Once the disk starts to service a request, this request is processed to its completion. This assumption is in accord

with the current physical disk characteristics. Similarly, block size is important in our environment due to our focus on a magnetic disk drive (it is not a focus of afro-mentioned TPS studies).

A few studies investigated scheduling issues in the Database Management System (DBMS) such as the HiPAC project [CLea88], Carey and Jauhari and Livny [CJL89]. The prior addressed how to support timing constraint in databases but did not consider the priority, cost and disk scheduling issues. The later addressed the priority in DBMS resource scheduling but did not consider disk scheduling and cost.

Other studies have investigated disk scheduling in the context of multimedia servers and applications. There are two classes of algorithms in this area. The first focuses on the placement of data on the disk and the disk head movement, e.g., Grouped Sweeping Scheduling (GSS) [CKY93]. Cycle-based algorithms partition requests into groups and sort requests in each group according to the placement of the data on the disk [GM98]. Next, they break a cycle into slots with one slot supporting a single display. Andrews and Bender presented a 3/2-approximation algorithm to schedule the disk head so that it services all the requests in the shortest time possible [ABZ96]. Typically, they do not associate a deadline with the request. The second one is called deadline-driven algorithm. It associates a deadline with each request but doesn't guarantee to meet all the requests' deadlines. Examples are Earliest Deadline First (EDF), SCAN-EDF [RW94]. Gemmell and Christodoulakis [GC92] established principles for the retrieval and storage of delay-sensitive multimedia data and identified that retrieval of those data from secondary storage must satisfy certain time constraint in order to be acceptable. These studies did not consider the role of different priority or cost levels associated with different requests.

Interactive multimedia data resident on the disk motivated their own disk scheduling algorithms. For instance, Chang and Garcia-Molina proposed Bubbleup [CGM97] to minimize initial latency for interactive requests. While all periodic requests are served in the SCAN order, when the first request of a new stream arrives, it is served at the head of the queue for immediate service. Wiljayaratne and Reddy [WR99] presented a similar algorithm that provides different performance level guarantees for interactive and periodic requests. The algorithm differentiated between two types of applications: audio/video applications and interactive applications. Audio and video applications produce periodic requests that arrive periodically to the disk with pre-specified deadlines. Interactive applications issue requests in an unpredictable way which are termed aperiodic requests. The algorithm maintains the request queue in SCAN order. It divides the periodic requests into groups and schedules the aperiodic

requests at the head of each group. Therefore, this algorithm can be regarded as a specialization of our proposed one-queue algorithm with two priority levels. In addition, our algorithm strives to minimize the total cost by potentially violating the SCAN order when scheduling requests in a queue.

We proposed the one-queue scheduling algorithm in [KNG00]. This study is an improvement of [KNG00] based on insights gained by presenting and discussing the original study. These improvements are as follows. First, we have fixed several details of the one-queue algorithm to make it cost-driven. Second, the evaluation section of this study is more comprehensive consisting of additional insights[5]. We used a multi-disk architecture for evaluation purposes of [KNG00]. This made the results somewhat ambiguous because, at times, one node would become the bottleneck for the entire disk subsystem and determine its overall processing capability. The formation of bottlenecks is orthogonal to a study that focuses on a disk scheduling algorithm. A focus on a single disk has enabled us to efficiently quantify and compare the one-queue algorithm with the multi-queue algorithm.

5 Conclusion and Future Research Directions

This paper proposes a new cost driven disk scheduling algorithm in support of environments consisting of requests with different priorities and costs. We compared this algorithm with a multi-queue algorithm using a simulation study. The obtained results demonstrate the superiority of the one-queue algorithm with moderate to high system loads (disk utilization of 70%-90%). We observed that the block size has an impact on the performance of both periodic and aperiodic requests when the deadline of a block is a monotonic function of its size. We derived analytical models to capture the essence of this observations.

The one-queue algorithm minimizes the impact of seeks by maintaining requests in scan cycles. While this has negligible impact on the overall system performance, it is worthwhile to note that the current technological trends in the area of magnetic disks is one of 40% annual increase in transfer rate and 5% annual improvement for the seek time [Gro00]. These trends re-enforce the practicality of our proposed algorithm.

A future research direction of this activity is to implement it in a real video-on-demand system for further analysis. In particular, we intend to adapt this algorithm to those environments that

[5]For example, the discussion on loss rate as a function of block size did not appear in [KNG00].

might change priority of requests in a dynamic manner. An example application is a system that both reads and writes data. It can assign a lower priority to write requests while there is sufficient memory to buffer the data to be written to the magnetic storage [AKG00]. The priority of these requests increase as resources become scarce.

References

[ABZ96] M. Andrews, M. A. Bender, and L. Zhang. New algorithms for the Disk Scheduling Problem. In *Proceedings of 37th Annual Symposium on Foundations of Computer Science*, pages 550–559, October 1996.

[AGM92] R. Abbott and H. Garcia-Molina. Scheduling Real-Time Transactions: A Performance Evaluation. *ACM Transaction on Database Systems*, 17(3):513–560, September 1992.

[AKG00] W. G. Aref, I. Kamel, and S. Ghandeharizadeh. Disk Scheduling in Video Editing Systems. To appear on *IEEE Transactions on Knowledge and Data Engineering*, 2000.

[CG96] R. M. K. Cheng and D. W. Gillies. Disk Management for a Hard Real-Time File System. In *Proceedings of the Eighth Euromicro Workshop on Real-Time Systems*, 1996.

[CGM97] E. Chang and H. Garcia-Molina. BubbleUp: Low Latency Fast-Scan for Media Servers. In *Proceedings of the fifth ACM International Conference on Multimedia*, pages 87–98, November 1997.

[CJL89] M. J. Carey, R. Jauhari, and M. Livny. Priority in DBMS Resource Scheduling. In Peter M. G. Apers and Gio Wiederhold, editors, *Proceedings of Fifteenth International Conference on Very Large Data Bases*, pages 397–410. Morgan Kaufmann, August 1989.

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

[CLea88] M. J. Carey, M. Livny, and R. Jauhari et al. The HiPAC Project: Combining Active Databases and Timing Constrains. *ACM SIGMOD RECORD*, 17(1):51–71, March 1988.

[CSKT91] S. Chen, J. A. Stankovic, J. F. Kurose, and D. Towsley. Performance Evaluation of Two New Disk Scheduling Algorithms. *Real Time Systems, (3)*, pages 307–336, 1991.

[Den68] P. J. Denning. The Working Set Model for Program Behavior. In *Communications of the ACM (CACM)*, 11(5):323–333, May 1968.

[GC92] J. Gemmell and S. Christodoulakis. Principles of Delay-Sensitive Multimedia Data Storage and Retrieval. In *ACM Transactions on Information Systems*, 10(1):51–90, January 1992.

[GH98] D. Gross and C. M. Harris. *Fundamentals of Queue Theory*, chapter 2. A Wiley-Interscience Publication, 3rd edition, 1998.

[GK00] S. Ghandeharizadeh and S. H. Kim. Design of Multi-User Editing Servers in Support of Continuous Media. In *Kluwer Multimedia Tools and Applications*, May 2000.

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

[Gro00] E. Grochowski. Magnetic Disk Trends. IBM Almaden Research Center, 2000.

[HCL91] J. R. Haritsa, M. J. Carey, and M. Livny. Value-Based Scheduling in Real-Time Database Systems. In *VLDB Journal*, 2(2):117–152, 1991.

[KNG00] I. Kamel, T. Niranjan, and S. Ghandeharizedah. A Novel Deadline Driven Disk Scheduling Algorithm for Multi-Priority Multimedia Objects. In *Proceedings of IEEE Data Engineering Conference*, 2000.

[LT91] T.H. Lin and W. Tarng. Scheduling Periodic and Aperiodic Tasks in Hard Real-Time Computing Systems. In *Proceedings of the 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 31–48, May 1991.

[RS94] K. Ramamritham and J. A. Stankovic. Scheduling Algorithms and Operating Systems Support for Real-Time Systems. In *Proceedings of the IEEE*, 82(1):55–67, January 1994.

[RW94] A. L. N. Reddy and J. C. Wyllie. Disk Scheduling in a Multimedia I/O System. In *IEEE Computer*, 27(3):69–82, March 1994.

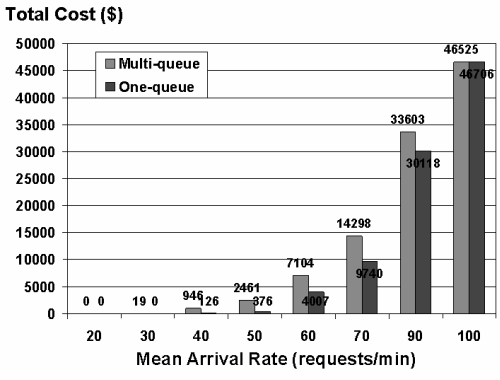
[SMRN00] J. R. Santos, R. R. Muntz, and B. Ribeiro-Neto. Comparing Random Data Allocation and Data Striping in Multimedia Servers. In *Proceedings of ACM Sigmetrics*, pages 44–56, June 2000.

[TH99] T. J. To and B Hamidzdeh. Dynamic Real-Time Scheduling Stratigies for Interactive Continous Media Servers. In *ACM Multimedia Systems*, 7(2):91–106, March 1999.

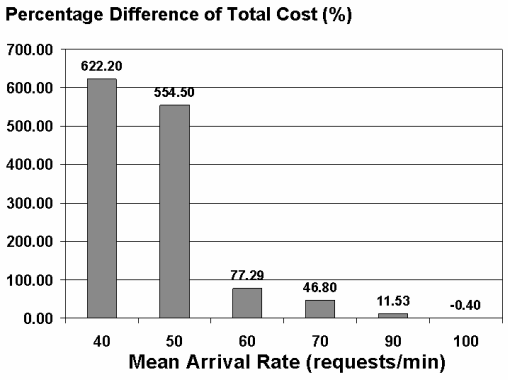
[WR99] R. Wijayaratne and A. L. N. Reddy. Integrated QOS management for disk I/O. In *Proc. of IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS'99)*, June 1999.

Appendix A: Experimental Results for Periodic Requests

This appendix presents the obtained results for the aperiodic requests. The database consisted of X video clips, each with a consumption rate of four megabits per second. Each clip consisted of one hundred 64KB blocks. The general observations and trends are in accord with those presented in Section 3.3. Hence, we will not repeat this discussion. Figures 6.a and b show the total cost of each algorithm. With moderate system loads, the one-queue algorithm provides significant savings. This is because this algorithm can queue and process a larger number of priority one requests while satisfying the deadline of priority two requests. This is demonstrated in Figures 7.a and b showing a longer average wait time for the one queue algorithm. Finally, Figures 8.a and b show the queue length with each algorithm.

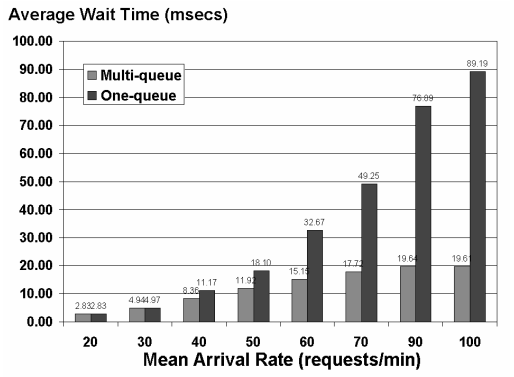


(a) Total cost

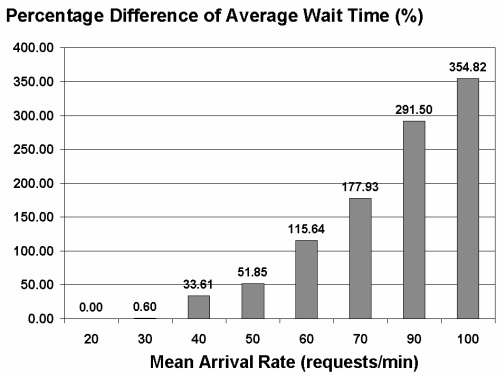


(b) Percentage difference

Figure 6: Total cost and percentage difference incurred by the lost periodic requests during 1,000 second display time as a function of the mean arrival rate, block size = 64KB, priority 1 : priority 2 = 10 : 1, priority 1 cost : priority 2 cost = 1 : 10.

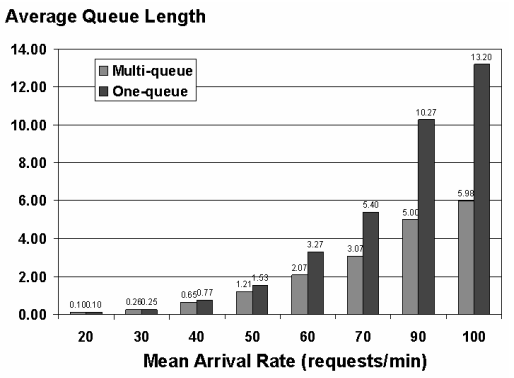


(a) Average wait time

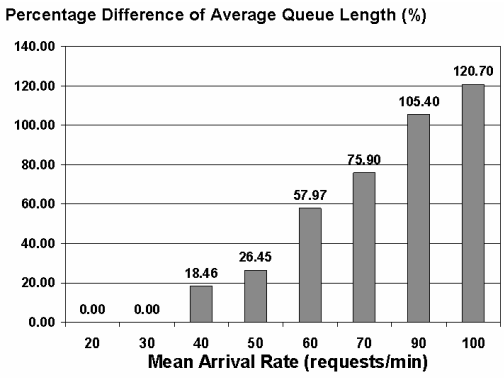


(b) Percentage difference

Figure 7: Average wait time and percentage difference of periodic requests as a function of the mean arrival rate, block size = 64KB, priority 1 : priority 2 = 10 : 1, priority 1 cost : priority 2 cost = 1 : 10.



(a) Average queue length



(b) Percentage difference

Figure 8: Average queue length and percentage difference of periodic requests as a function of the mean arrival rate, block size = 64KB, priority 1 : priority 2 = 10 : 1, priority 1 cost : priority 2 cost = 1 : 10.