Evaluation of Scheduling Algorithms for Real-Time Disk I/O

Yifeng Zhu
Department of Computer Science and Engineering
University of Nebraska - Lincoln
Lincoln, NE, 66588-0155
yzhu@cse.unl.edu

May 3, 2002

Abstract

Real-time disk I/O scheduling is extremely important to the performance improvement of the whole real-time system since the disk devices are the system's bottleneck. To meet time constraints, some conventional algorithm, such as EDF, Least Slack Time First, Rate Monotonic, can be used to schedule disk I/O. However, these algorithms ignore the relative position of requested data on the disks, hence their throughput is certainly low due to large arm positioning time. To keep a good tradeoff between optimizing throughput and meeting time constraints, several hybrid real-time scheduling algorithms were proposed. These algorithms include Priority SCAN(P-SCAN), SCAN-EDF, Feasible Deadline SCAN(FD-SCAN), Feasible Deadline EDF(FD-EDF), and Shortest Seek Earliest Deadline by Order/Value(SSEDO, SSEDV). In this project, we compared real-time algorithms, such as EDF, FD-EDF, SSEDO and SSEDV, with conventional algorithms, such as FCFS, SSTF and SCAN, in the aspects of guarantee ratio, average throughput and merit according to our simulation results.

1 Introduction

The study of real-time systems has recently become a rapidly expanding area of research. This area has many challenging issues both in theoretical research and industrial application. One of these challenges is real-time I/O scheduling. While RISC microprocessor performance has been improving 50% per year [19], disk access time, which depends on improvements of mechanical systems, has been improving only 10% per year [3]. Disks become the bottleneck of the whole system. The improvement in the I/O system will significantly improve the performance of real-time systems. This motivates us to investigate the problem of the real-time I/O scheduling. To be more specific, this project would investigate soft real-time disk I/O scheduling algorithms.

Real-time I/O scheduling has not been extensively studied. One important reason is that in many real time systems disks are not accessed under time constraints. As the real-time systems become more complex and larger, it is necessary to consider the time constrained disk access into the real-time system. For example, real-time database, video and audio player, real-time satellite imagery feature extraction, real-time visualization [24], all these soft real-time applications need a good real-time disk I/O scheduling algorithm to satisfy their time constraints.

Disk workloads are often characterized by intense bursts of activity, creating long queues of pending requests [21]. The disk scheduler is responsible for dynamically ordering the pending requests. This scheduling has been studied since 1960's. However, the conventional disk scheduling algorithms aim to optimize the disk throughput. They include First Come First Served(FCFS), Shortest Seek Time First (SSTF), SCAN, Cyclical SCAN (C-SCAN), LOOK, and Cyclical LOOK (C-LOOK) [5, 6, 8, 9, 10, 13, 23]. To meet time constraints, some conventional real-time algorithms, such as Earliest Deadline First (EDF), Least Slack Time First and Rate Monotonic (RM) [15, 17, 18], can be used to schedule disk I/O requests. However, these algorithms ignore the relative position of requested data on the disks. Due to the large seek time and rotation latency, their throughput is relatively low. To keep a good tradeoff between optimizing throughput and meeting time constraints, several hybrid real-time scheduling algorithms were proposed, such as Priority SCAN (P-SCAN) [2], SCAN-EDF [20], Feasible Deadline SCAN (FD-SCAN) [1], Shortest Seek Earliest Deadline by Order/Value (SSEDO, SSEDV) [4] and T-SCAN [7].

Ref. [4] compared nine disk scheduling algorithms in a database environment. They used transaction loss probability, i.e., the probability that a transaction does not meet its deadline, as their overall metric. Ref. [7] evaluated their T-SCAN algorithm in a multimedia system by comparing the buffer size needed by disk scheduling algorithms. Ref. [28] used fluid approximations and analytical techniques from manufacturing scheduling problems to estimate the transient and steady states of a multimedia system. The metric they used is frame loss ratio. In this paper, we evaluated disk scheduling algorithms in the aspects of guarantee ratio, merit and throughput by simulation in a multimedia environment. The guarantee ratio indicates the ratio of requests whose deadlines can be met. The merit, defined in Section 4, is used to measure the usefulness of a soft real-time scheduling algorithm.

The rest of this paper first discusses related work in Section 2, and then describes our implementation in Section 3. Section 4 describes how we evaluated our system and presents the results. Section 5 presents our conclusions and describes future work. Finally, Section 6 gives our acknowledgment.

2 Related Work

This section gives a review of conventional and real-time disk I/O scheduling algorithms.

2.1 Conventional Disk Scheduling Algorithms

The simplest form of disk scheduling algorithm is First Come First Served (FCFS). FCFS is intrinsically fair but it can incur significant seek time and rotational latency overhead. Other conventional scheduling algorithms were proposed to reduce seek times. The Shortest Seek Time First (SSTF) selects the request with the minimum seek time from the current disk arm position. SCAN algorithm, first proposed by Denning [8], serves the requests in the direction of the arm movement until no further requests in this direction and then changes its scan direction. There are some variants of SCAN, such as C-SCAN, LOOK and C-LOOK. In addition, Ref. [10] proposed a continuum of algorithms between SSTF and SCAN. It added a penalty of R times the total number of cylinders for reversing direction. It suggested that $R = 0.2$ keeps a good compromise between SSTF and SCAN. Ref. [23, 14] proposed the Shortest Access Time First algorithm(SATF), which attempts to minimize the sum of seek time and rotational delay.

These algorithms aim to provide a high I/O bandwidth by minimizing the total seek time and rotational latency. However, no time constraint has been considered in there algorithms. Therefore, they can not be used in the real time system environment, where each request has a deadline and must have been served before its deadline.

2.2 Disk Scheduling Algorithms for Real-time I/O

In real-time systems, when requests have to be satisfied within deadlines, algorithms such as Earliest Deadline First (EDF) and Least Slack time First (LSF) are used. Ref. [18] shows that the EDF algorithm is optimal if preemption is allowed and jobs do not contend for resources. However, current disks are non-preemptive. Jeffay [15] examined the feasibility of scheduling a set of periodic or sporadic tasks on a uniprocessor without preemption and concluded that EDF algorithm is optimal for sets of sporadic and periodic tasks and for sets of concrete sporadic tasks, except for concrete periodic tasks. Accordingly EDF is a natural choice for the real-time disk I/O scheduling. However, strictly EDF has a large overhead of seek time and rotational latency and thus results in a poor utilization of the disk. Several hybrid algorithms, which combine EDF with conventional disk scheduling algorithms, are proposed in recent years. The following discusses five of them: P-Priority, FD-EDF, SCAN-EDF, SSEDO, SSEDV.

2.2.1 Priority SCAN

The Priority SCAN (P-SCAN) [2] divides all I/O requests into several priority levels according to their deadlines and SCAN algorithm is used within each level. The scheduler maintains several queues, each of which has a different priority. All the I/O requests in the same queue have the same priority. The requests are assigned a priority according to their deadlines and then buffered into the corresponding queue. Upon the completion of a disk request, the scheduler checks whether a non-empty queue of a higher priority is waiting for services. If such a queue is found, the scheduler switches to that queue and serves it using SCAN algorithm.

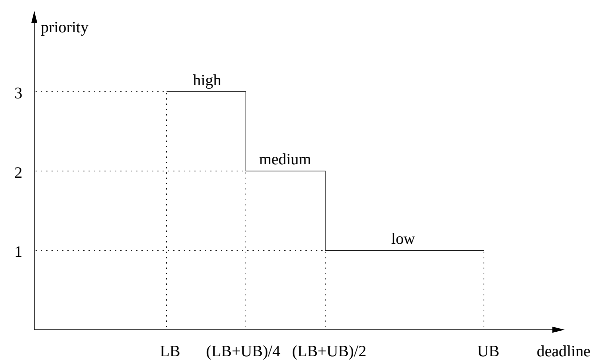

Figure 1: Priority assignment according to the relative deadline.

Ref. [2] discovers that the average response time can worsen as the number of priority levels increases. Therefore, a small number of priorities is preferred to provide reasonable I/O performance. Ref. [4] discusses how to assign priorities into three levels: high, medium, and low. The priority assignment is shown in Fig. 1, where UB and LB are the upper bound and lower bound of the relative deadlines respectively. It concludes that this assignment is better than the one-third strategy, which evenly divides the deadline range into 3 intervals and maps each interval into a priority level.

2.2.2 FD-EDF

Abbott [1] suggested a real-time disk scheduling algorithms, called Feasible Deadline SCAN (FD-SCAN). A deadline is called feasible if it is estimated that it can be met. FD-SCAN dynamically adapts the scan direction towards the request with the earliest feasible deadline. This motivates the development of another algorithm, Feasible Deadline EDF (FD-EDF)[4]. If there are requests with feasible deadlines waiting the queue, the disk controller serves these requests using the EDF algorithm. Otherwise, it serves the queue using FCFS. A request which has the earliest deadline but whose deadline is not feasible, may be served only when there is no request with feasible deadline in the waiting queue.

2.2.3 SCAN-EDF

SCAN-EDF is a hybrid algorithm proposed in Ref. [20]. It serves the requests with earlier deadlines first and requests with the same deadline using SCAN algorithm. This would efficiently make use of the disk bandwidth in addition to maintaining the time constraints. A precise description of this algorithm is as follows:

Step 1. Let T = sets of tasks with earliest deadlines.

Step 2. If $|T| = 1$, serve that request, else let t_1 be the first task in T in the scan direction, serve t_1, and then go to **Step 1**.

The efficiency of SCAN-EDF depends on the fraction of the requests that have the same deadlines. Two techniques [20] can be used to increase the fraction. The first one is to let the deadline be equal to the period, and let the release time be a multiple of the period. The second one is used when the requests have different data rate requirements. It uses a periodic fill policy, proposed in Ref. [27], to let all the requests have the same deadline.

2.2.4 Shortest Seek and Earliest Deadline by Ordering

Shortest Seek and Earliest Deadline by Ordering (SSEDO) [4] sorts all the requests in a queue according to their absolute deadline. A window of size m is defined as the first m requests in the queue. The scheduler first assigns a weight $w_i = \beta^{i-1}$, $(\beta \geq 1,\ i = 1, 2, \ldots, m)$, to each request r_i in the window at each scheduling instance. Note that β is an adjustable scheduling parameter. Then it serves the request with the minimum value of $w_i d_i$, where d_i is the distance between the current disk arm position and the desired arm position of request r_i.

The basic idea of $SSEDO$ is to give chances to a request that has a large deadline but is "very" close to the current arm position. This algorithm approximates SSTF when $\beta = 1$ and EDF when $m = 1$.

2.2.5 Shortest Seek and Earliest Deadline by Value

Shortest Seek and Earliest Deadline by Value (SSEDV) [4] is very similar to SSEDO except that the weights of the requests in the window are assigned based on the value of the deadlines, instead of the order of the deadlines. It associates $w_i = \alpha d_i + (1 - \alpha) l_i$, where α is an adjustable scheduling parameter, d_i is the distance between the current arm position and the desired arm position, and l_i is the slack time, defined as the length between the current time and the absolute deadline of request r_i. It is interesting to notice that SSEDV approximates SSTF when $\alpha = 1$ and LSF when $\alpha = 0$. Ref. [4] also did experiments in an actual real-time transaction system benchmark, called RT-CARAT, and concluded that SSEDV outperforms SSEDO.

sector size	512 bytes
cylinders	1962
tracks per cylinder	19
data sector per track	72
number of zones	1
tract skew	8 sectors
cylinder skew	18 sectors
revolution speed	4002 RPM
tract switch time	1.6 ms
read fence size	64 KB
disk buffer cache size	128 KB

Table 1: Disk parameters of HP 97560.

3 Implementation

In this section, we introduce some characteristics of disk devices. After that, the disk model, including bus interface, mechanical positioning, data layout and caching, is discussed. The workload, used in the simulation, is also described here. At the end of this section, we discuss how to implement the simulation.

3.1 Characteristics of Disk Drives

A modern disk consists of a set of rapidly rotating platters coated on both sides with magnetic media. Data is read and written in circular **tracks** on each **surface** through disk heads. Each track is further broken down into smaller pieces called **sectors**. A **cylinder** is a set of tracks equidistant from the center of the disk.

Each disk has a mechanical positioning component, which ensures that the appropriate head gets to the desired track as quickly as possible, fine tunes the head position and keeps it on that track. The response time of this procedure includes **seek time**, **rotational latency** and **head switching time**. Seek time is the amount of time needed to move a head to the correct tract. Rotational latency is the amount of time needed for the desired sector to rotate under the disk head. Head switching time is the amount of time for the controller to switch its data channel from one surface to the next in the same cylinder due to the physical alignment error between the tracks on the different surface.

3.2 Disk Model

Ref. [22, 25] give a precise and detailed description of disk modeling. Ref. [26] gives a detailed information about extracting disk drive parameters via the ANSI-standard interface. Since the parameters of HP 97560, as shown in Table 1, is clearly given in Ref. [22], we basically take this disk as our disk model in this project. To precisely measure the performance of disk scheduling algorithms, we incorporate mechanical positioning, data layout, and request queuing into our simulation. Admission control currently is not considered in this project. Jeffay [15] provides an efficient admission control algorithm on non-preemptive periodic, sporadic and concrete sporadic tasks. We may incorporate this into our model in the future.

3.2.1 Bus Interface

The most important aspects of a disk driver's host channel are its topology, transfer rate, and overhead. Currently, the Small Computer System Interface (SCSI) is a typical bus. Most disk drives use the SCSI bus operation's synchronous mode, which can run at the maximum bus speed. Standard SCSI transmits data at 5 MB/Sec and its fastest type transmits at approximately 40 MB/Sec. As the number of devices on the bus increases, contention for the bus can occur. On SCSI, the cost of acquiring control of the bus is on the order of a few microseconds if the bus is idle. The SCSI protocol also allows a disk drive to disconnect from the bus and reconnect later once it has data to transfer. This cycle may take $200\mu s$ [22]. In addition, most SCSI drives take data off the media more slowly than they can send it over the bus, hence the drive fills partially its buffer before attempting to commence the bus data transfer [22]. A read fence size, defined as the amount of data which is read into the buffer before the SCSI drive initiates the transfer, is used to measure how fast the SCSI drives can response. Write requests can be asynchronous since it can transfer the data to the disk's buffer as long as the buffer has free space.

3.2.2 Mechanical Positioning

The Mechanical positioning time includes disk seek time and rotational latency. By keeping track of the rotational position of the disk, we can explicitly calculate the rotational latency, assuming that the disk is spinning at exactly its nominally rated speed. The seek time $T_S(d)$ can be expressed in milliseconds as follows [22]:

$$T_S(d) = \begin{cases} 0 & \text{if} \quad d = 0; \\ 3.24 + 0.40\sqrt{d} & \text{if} \quad d \leq 383; \\ 8.20 + 0.0075d & \text{if} \quad d > 383. \end{cases}$$

where d is the seek distance in cylinders. This model can be conceptually explained in the following way. During a long distance seek, most modern disk arms first accelerate up to their maximum speed, then keep that speed for some time, next decelerate back to zero, and finally settle onto the desired cylinder. The whole process is dominated by running with the maximum speed. Therefore, the seek time increases linearly with the seek distance. During short seeks, the arms accelerate up in the first half distance, decelerate and then settle in the second half distance. The whole process is dominated by accelerating and decelerating. Therefore, the seek time increases with the square root of the seek distance.

3.2.3 Data Layout

Most disks need to map the logical block number to the physical block number. The complexity of mapping is increased due to the fact that the disk must hide bad sectors and do some low-level optimization, such as zoning, track/cylinder skew, and sparing [25]. To precisely calculate the disk seek time, we incorporate the complex mapping information into our disk model.

3.2.4 Caching

Cache is used to match the speed disparity between the disk drive and the CPU. SCSI supports read caching, also called read-ahead, which means actively retrieving and caching data that the disk expects the host to request momentarily. As for write caching, two techniques are usually applied. One is immediate reporting, the other is battery-backed RAM. In addition, SCSI also

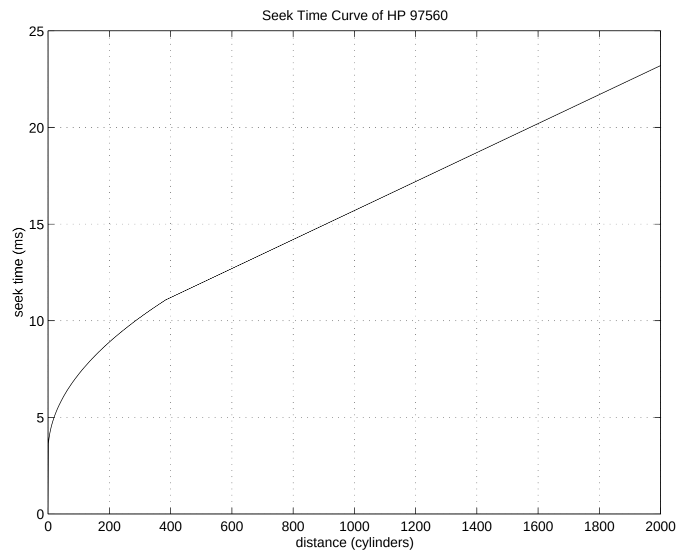

Figure 2: Seek time versus seek distance in cylinders for the HP 97560.

supports command queuing, which allows the host to give the disk controller several requests and let the controller determine the best execution order.

3.3 Workload Model

Multimedia, such as video and audio, is a soft real-time workload. It is also often referred to constant rate workload because they consist of a sequence of media quanta (such as video frames or audio samples), which convey meaning only when presented continuously in time. While the maximum throughput of modern disks is 5-8 MBytes/s, the throughput requirement of an MPEG-2 encoded video stream is 0.42 MBytes/s, and the requirement of an uncompressed CD-quality stereo audio is about 0.18 MBytes/s [11]. In this project, we assume that

- All the workloads are MPEG-2 video streams, or all the workloads are CD-quality stereo audio.
- The first request of each stream arrives in an exponential distribution. Let $T_i^{(1)}$ is the first request of the video stream i. The probability density of $T_i^{(1)}$ is given as follows,

$$f(T_i^{(1)}) = \lambda e^{-\lambda T_i^{(1)}} \tag{1}$$

  where λ is the arrival rate of video streams.
- The following requests of each stream are issued exactly at the time instant of the absolute deadline of the previous request.
- The size of each request is uniformly distributed from 1 to 50 tracks.
- The relative deadline of each request is the ratio between the request data size and the throughput requirement (0.42 MBytes/s or 0.18 MBytes/s).
- The data is uniformly distributed in the whole disk.

3.4 Event Driven Simulation

Ref. [16] implemented a detailed model of the HP 97560 disk drive in ANSI C, which replicated the model developed by Ruemmler and Wilkes [22]. Their design can simulate one or more disk drives attached to one or more SCSI buses. They validated their model using seven-day traces obtained from Hewlett-Packard. In Ref. [22], Ruemmler and Wilkes evaluated their disk model by calculating "demerit". They plotted the time distribution curves for the real drive and model output and defined the root mean square of the horizontal distance between these two curves as "demerit". Using the same "demerit" measure as Ruemmler and Wilkers, they obtained a demerit percentage of 3.9%, indicating that their model was extremely accurate.

However, they did not consider the disk request queue in the disk controller and only FCFS scheduling algorithm was implemented. Based on their model, we incorporated a disk request queue into the disk controller. Ref. [21] discovered that disk I/O is very bursty in UNIX and the maximum number of entries in the disk request queue can be as large as 1000. In our disk model, we assume that the disk is able to store 1000 requests. When the queue is full, CPU has to wait until the queue has empty space for this request.

We implemented some real-time disk scheduling algorithms, such as EDF, FD-EDF, SSEDV and SSEDO. We built a scheduling class, which encapsulated different disk scheduling algorithms. The scheduling class is driven by the events of requests, such as the release of a new request and the completion of an old request. Disk request preemptions lead to low throughput because preemptions cause more large disk arm moving. Hence, most modern disks are non-preemptable. Therefore, the scheduler only dispatches the request after the event of the completion of an old request. The scheduler can be briefly described as follows:

1. Sort all the requests according to their release times and put them into a queue named **ToBeServed**.
2. Current time t = the release time of the first request in **ToBeServed**.
3. Initialize a queue, named **Released**, to be empty.
4. while **ToBeServed** is not empty, do
   (a) Move all requests in **ToBeServed** with release time less than t into **Released**.
   (b) Select an request r from **Released** according to the disk scheduling algorithm.
   (c) Calculate the disk service time t_{serve} of r according to the disk model
   (d) Delete r from **Released**.
   (e) Current time $t = t + t_{serve}$
5. end of while

Due to time limitation, other real-time algorithms, such as P-SCAN and FD-SCAN are not implemented. To compare their performance with conventional disk scheduling algorithms, FCFS, SSTF and SCAN have also been implemented in our simulation.

4 Evaluation

In this section, we evaluate these disk scheduling from three perspectives: guarantee ratio, average throughput and merit. Guarantee ratio is defined as the percentage of requests which meet their time constraints. In the following, we describe our merit function. After that, we present our simulation results.

4.1 Merit Definition

To indicate the usefulness of scheduling algorithms, we developed a merit function that is associated with the response time of requests. Let the total request set be $\{T_1, T_2, \ldots, T_n\}$. We define the rated response time R_i^* as follows:

$$R_i^* = \frac{R_i}{D_i} \tag{2}$$

where R_i and D_i are the response time and relative deadline of request T_i. The merit of serving an individual request T_i by these scheduling algorithms is defined as a function of its rated response time, given as follows:

$$\varphi(R_i^*) = \begin{cases} 1 & \text{if} \quad R_i^* \leq 1; \\ e^{-2(R_i^*-1)} & \text{if} \quad R_i^* > 1. \end{cases}$$

This function is graphically shown in Fig. 3. The merit of some disk scheduling algorithm can be defined as the average merit, i.e.,

$$merit = \sum_{i=1}^{n} \varphi(R_i^*)/n \tag{3}$$

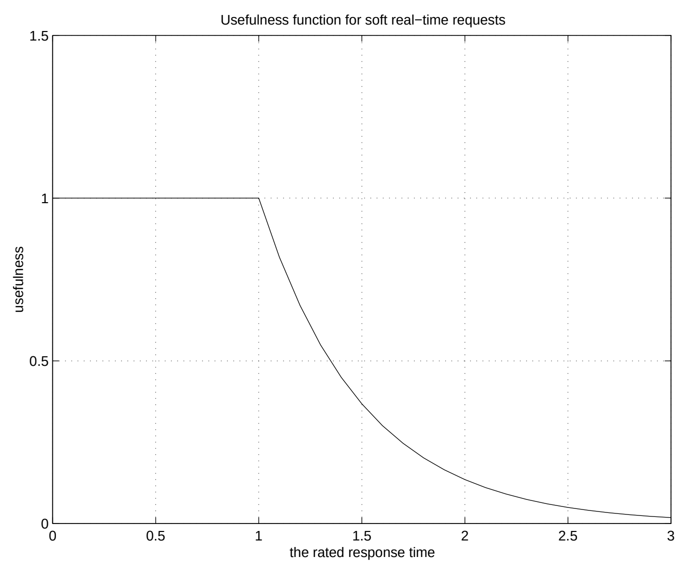

Figure 3: Usefulness function for soft real-time requests.

As we can see from the definition, the value of each merit is between 0 and 1. In this project, we will calculate the merit of each disk scheduling algorithm from the simulation results and evaluate these algorithms based on their merit values. Validation of our merit function is really a challenge for us. Usefulness functions have been used in evaluating soft real-time system [12]. However, validation of usefulness functions is always tricky. In this project, we developed this merit function by art. The definition shows that the higher the average merit value is, the more possibly the deadlines are met. Hence, a higher average merit value implies a better scheduling algorithm.

4.2 Simulation Results

The throughput requirements of a MPEG-2 encoded video stream and an uncompressed CD-quality stereo audio are 0.42 and 0.18 MBytes/s respectively [11]. Each MPEG-2 or CD audio load is a constant rate stream. In this project, we evaluate the performances of FCFS, SSTF, SCAN, EDF, FD-EDF, SSEDO, SSEDV algorithms under different number of MPEG-2 and CD audio streams. The following describes their performances from three aspects: guarantee ratio, average throughput, and average merit.

4.2.1 Guarantee Ratio

Guarantee ratio is defined as the percentage of requests that finish before their deadlines. We simulate 1000 MPEG-2 requests and 1000 CD audio requests. The requests are generated in the way described in Section 3.3. Fig. 4 and Fig. 5 give their guarantee ratio curves when the total stream number increases from 10 to 60. These curves show that the guarantee ratio decreases when the work load increases. This is reasonable since larger ratio of requests will miss their deadlines under heavier workloads. Fig. 6 shows the average guarantee ratio of different algorithms under the MPEG-2 and CD audio loads. While F-EDF is the best and has an average guarantee ratio of 0.2492, the guarantee ratio of SSEDV is pretty close to that of F-EDF. Among them, EDF and SCAN have the lowest guarantee ratios.

The simulation results also show that EDF is much worse than SSTF in the sense of guarantee ratio. This can be explained in two aspects. Firstly, this EDF is non-preemptive. Secondly, EDF schedules the requests only according to their deadlines and does not consider the distance between the current disk arm position and the desired arm position. Thus, the disk arm often moves a long distance to serve the earliest deadline request. These long distance moves further delay the serving to the rest requests waiting in the queue and thus greatly deteriorate the overall performance of EDF. Although SSTF does not consider the deadlines, it serves the request nearest to the current disk arm position and greedily serves every request as soon as possible.

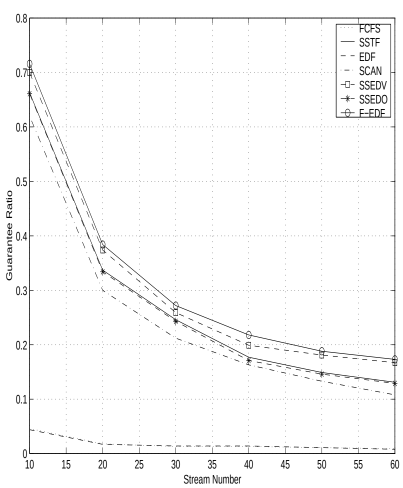

Figure 4: Guarantee ratio (CD audio loads).

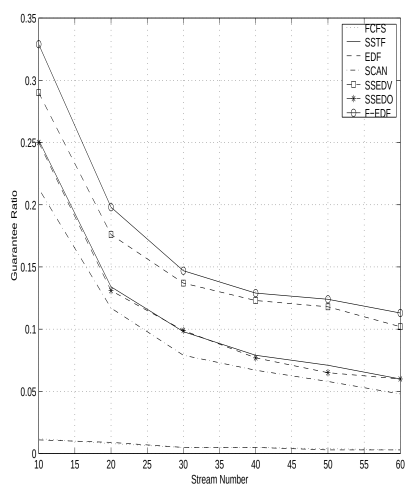

Figure 5: Guarantee ratio (MPEG-2 loads).

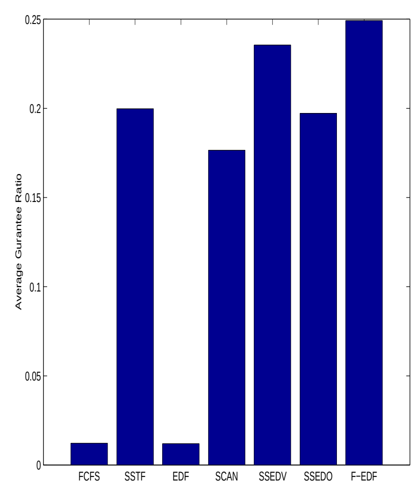

Figure 6: Average guarantee ratio.

4.2.2 Average Merit

The relationship between the average merit and the stream number is given in Fig. 7 and Fig. 8. The figures show that, under both MPEG-2 and CD audio loads, the average merit decreases when the stream number increases. Fig. 9 shows the average merits of these scheduling algorithms. These average merits seem to be too low to be practical for real multimedia applications. However, most MPEG or CD audio applications employ play-back technologies, i.e., applications buffer some amount of data before beginning to play. Fig. 10 shows that the merit value increases with the total buffer size when the loads are 40 CD audio streams. Under our workload, when the total buffer size is 8MBytes, (0.2MBytes for each stream), the average merit can reach as high as 0.85. Fig. 10 also indicates that the merit of real-time algorithms may be inferior to that of non-real-time algorithms when play-back technologies are used. However, when we change the work loads, we can not always observe this characteristics. This characteristics may be explained by the following. We notice that real-time algorithms usually have larger response time variance than non-real-time algorithms. Therefore playback technologies can reduce the variance more significantly in non-real-time algorithms and thus improve their merit possibly. To quantitively analysis the influence of playback techniques is an interesting topic. Currently in this project, we did not consider the play-back techniques.

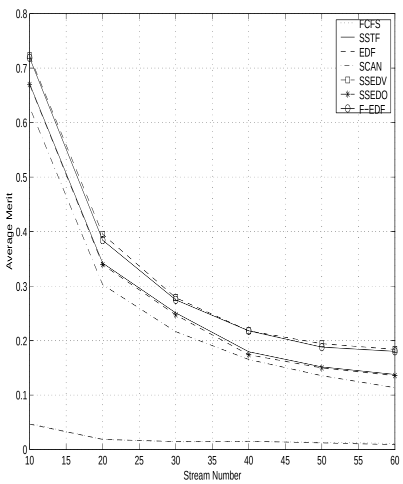

Figure 7: Average Merit (CD audio loads).

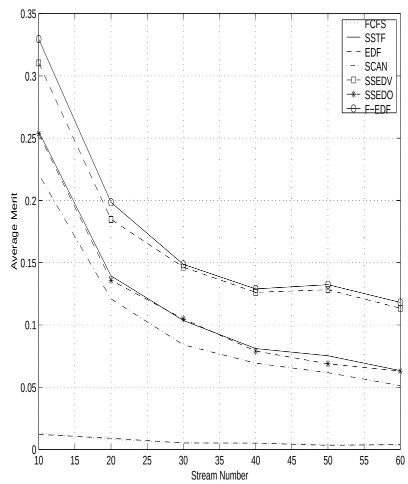

Figure 8: Average Merit (MPEG-2 loads).

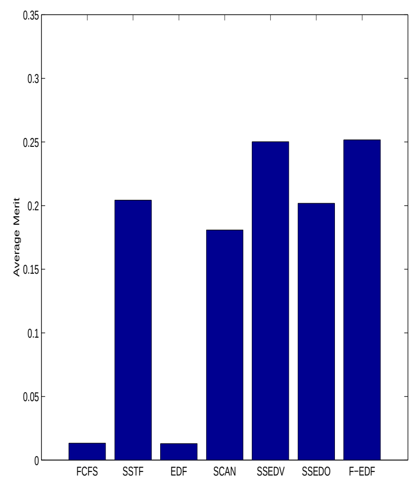

Figure 9: Average Merit.

As seen in Fig. 9, SSEDV and F-EDF have a higher merit than all conventional non-real-time scheduling algorithms. Although SSEDO performs 1.3% worse than SSTF, it is much better than other conventional non-real-time algorithms. In addition, the average merit of SSEDO is 15% lower than that of SSEDV because SSEDV uses more knowledge concerning the time constraints and this knowledge is obtained without adding any overhead. Ref. [4] also showed that SSEDV performs better than SSEDO in a real-time database transaction system.

4.2.3 Average Throughput

Fig. 11 and Fig. 12 give the average throughput curves of each algorithm. The throughput of each algorithm becomes worse when the load becomes heavier. This is exactly what we anticipated since,

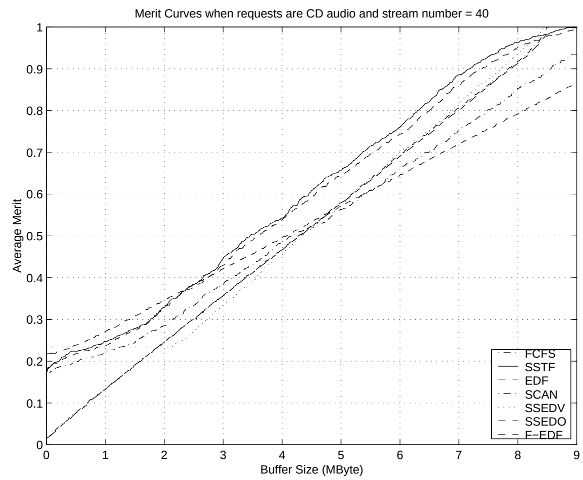

Figure 10: The influence of buffer size on merit value.

when the load becomes heavier, more requests will be waiting in the queue and thus the average response time increases.

SSTF, the best non-real-time algorithm, has the highest throughput of around 4.5MByte/s. All real-time disk scheduling algorithms have a lower throughput than SSTF. As shown in Fig. 13, the throughputs of F-EDF and SSEDV are 17% and 12% lower than that of SSTF respectively. However, their average merits are 23% and 22% higher, and their average throughputs are 25% and 19% higher respectively. This proves that F-EDF and SSEDV algorithms really keep a good tradeoff between satisfying time constraits and optimizing the throughput.

Fig. 13 also illustrates that FCFS, EDF, SCAN and F-EDF have almost the same throughput under our workloads. We anticipates that they may differ from each other in other workloads.

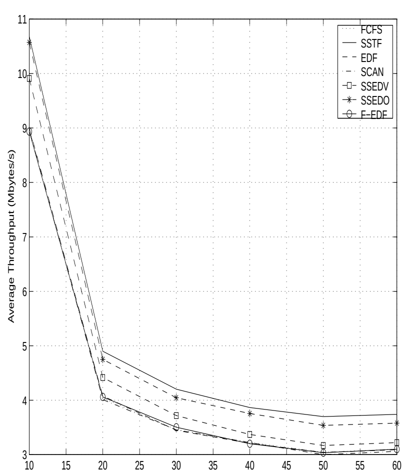

Figure 11: Average Throughput (CD audio loads).

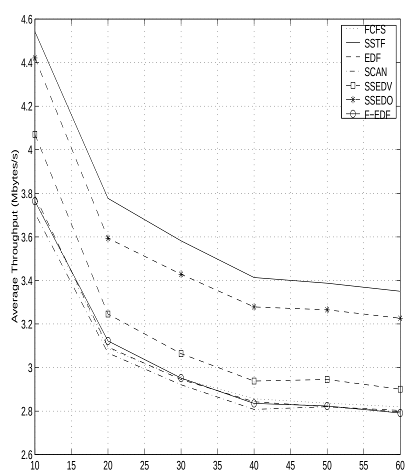

Figure 12: Average Throughput (MPEG-2 loads).

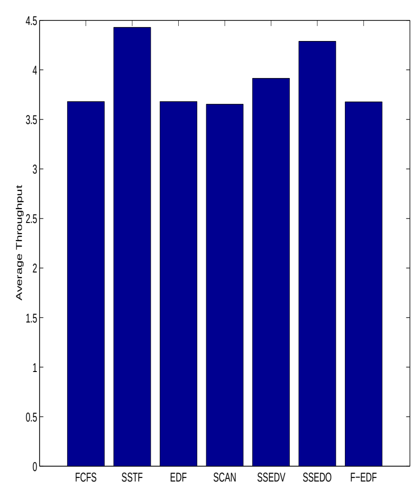

Figure 13: Average Throughput.

5 Conclusion and Future Work

In this project, we examined the performance of various disk scheduling algorithms in a multimedia system. The performance of each scheduling algorithm highly depends on the workloads. One algorithm performs best in one workload while it may perform worst in another workload. Under MPEG-2 encoded video (0.42MBytes/s) and CD-quality stereo audio (0.18MBytes/s) workloads, we simulate real-time disk scheduling algorithms EDF, SSEDV, SSEDO and F-EDF, as well as the conventional algorithms, FCFS, SCAN and SSTF.

The main performance results are as follows:

- SSEDV and F-EDF can significantly improve the guarantee ratio and merit more than other algorithms.
- SSEDV has a higher merit and throughput than SSEDO because SSEDV uses more information about the deadlines than SSEDO. This proves the conclusion of Ref. [4].
- Not all real-time algorithms perform better than non-real-time algorithms. For example, non-preemptive EDF has the worst guarantee ratio, merit and throughput.
- Playback techniques in multimedia applications significantly improve the performance of all disk algorithms.
- All real-time disk scheduling algorithms have a lower throughput than SSTF, the best non-real-time algorithm.
- SSEDV and F-EDF have a good tradeoff between optimizing throughput and meeting time constraints. While their throughputs are $12-17\%$ lower than that of the SSTF, their average merits are around 23% higher, and their average guarantee ratio are $19-25\%$ higher.

In the future, we can add admission control to the disk controller to improve their performance. The influence of playback techniques on the performance of disk scheduling algorithms is another interesting research issue. Also we have examined only the disk read operations in this project, a choice justified by the fact that multimedia systems rarely require real-time write. However, the effects of real-time disk write operations should be examined. In addition, we may simulate these disk scheduling algorithms in RAID and incorporate the real-time communication protocol to implement real-time storage in Storage Area Networks (SAN) and Network Attached Storage (NAS).

6 Acknowledgment

The author wish to thank S. Goddard for his constructive comments on the proposals. Thanks are also due to X. Qin and H. Jiang for their invaluable help with disk modeling.

References

[1] R. Abbott and H. Garcia-Molina. Scheduling I/O requests with deadlines: a performance evaluation. *Proceedings of the Real-time Systems Symposium*, pages 113–124, 12, 1990.

[2] M. J. Carey, R. Jauhari, and M. Livny. Priority in DBMS resource scheduling. *Proceedings of the Fifteenth International Conference on Very Large Data Bases*, pages 397–410, 1989.

[3] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, 1994.

[4] S. Chen, J. A. Stankovic, J. F. Kurose, and D. Towsley. Performance evaluation of two new disk scheduling algorithms for real-time systems. Technical Report UM-CS-1990-077, University of Massachusetts, Amherst, 1991.

[5] E G. Coffman and M. Hofri. On the expected performance of scanning disks. *SIAM Journal of Computing*, 10(1):60–70, 1982.

[6] E G. Coffman, L A. Klimko, and B. Ryan. Analysis of scanning policies for reducing disk seek times. *SIAM Journal of Computing*, 1(3):269–279, 1972.

[7] S. Daigle and J. Strosnider. Disk scheduling for multimedia data streams, 1994.

[8] P.J. Denning. Effects of scheduling on file memory operations. *AFIPS Spring Joint Computer Conference*, 4, 1967.

[9] R. Geist, R. Reynolds, and E. Pittard. Disk scheduling in system V. *Communications of the ACM*, 30:59–68, 1987.

[10] R. M. Geist and S. Daniel. A continuum of disk scheduling algorithms. *ACM Transactions on Computer Systems*, 5(1):77–92, 2, 1987.

[11] Jim Gemmell, Harrick M. Vin, Dilip D. Kandlur, P. Venkat Rangan, and Lawrence A. Rowe. Multimedia storage servers: A tutorial. *IEEE Computer*, 28(5):40–49, 1995.

[12] S. Goddard. CSE990 real-time systems. *Lecture notes, University of Nebraska - Lincoln*, 2002.

[13] M. Hofri. Disk scheduling: FCFS vs. SSTF revisited. *Communications of the ACM*, 23(11):645–653, 1980.

[14] D. M. Jacobson and John Wilkes. Disk scheduling algorithms based on rotational position. Technical Report HPL-CSP-917rev1, HP Laboratories Technical Report, 2, 1991.

[15] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. *Proceeding of the Twelfth IEEE Real-Time Systems Symposium*, pages 129–139, 12, 1991.

[16] David Kotz, Song Bac Toh, and Sriram Radhakrishnan. A detailed simulation model of the HP 97560 disk drive. Technical Report TR94-220, Dartmouth College, 1994.

[17] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. *IEEE Real Time Systems symposium*, 1989.

[18] C. L. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *J. ACM*, 17(1), 1973.

[19] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design: the hardware/software interface*. Morgan kaufmann, San Mateo, Calif., 1994.

[20] A. N. Reddy and J. Wyllie. Disk scheduling in multimedia I/O system. *Proceedings of ACM Multimedia'93, Anaheim, CA*, pages 225–234, 8, 1993.

[21] C. Ruemmler and J. Wilkes. Unix disk access patterns. *Winter USENIX Conference*, pages 405–420, 1, 1993.

[22] Chris Ruemmler and John Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–28, 1994.

[23] M. Seltzer, P. Chen, and J. Ousterhout. Disk scheduling revisited. In *Proceedings of the USENIX Winter 1990 Technical Conference*, pages 313–324, Berkeley, CA, 1990. USENIX Association.

[24] Prashant J. Shenoy and Harrick M. Vin. Cello: A disk scheduling framework for next generation operating systems. Technical Report CS-TR-97-27, University of Texas at Austin, 1, 1998.

[25] B. L. Worthington, G. R. Ganger, and Y. N. Patt. Scheduling algorithms for modern disk drives. In *Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 241–251, Nashville, TN, USA, 16–20 1994.

[26] Bruce L. Worthington, Gregory R. Ganger, Yale N. Patt, and John Wilkes. On-line extraction of SCSI disk drive parameters. Technical Report CSE-TR-323-96, University of Michigan, Ann Arbor, 12 1996.

[27] J. Yee and P. V. Rangan. Disk scheduling policies for real-time multimedia applications. Technical report, University of California, Bekeley, 8, 1992.

[28] James Yee and Pravin Varaiya. Models and performance of real-time disk access policies. *Computer Communications*, 18(10):725–741, 1995.