Disk Scheduling in a Multimedia I/O System

A. L. N. REDDY
Texas A&M University
JIM WYLLIE
IBM Almaden Research Center
and
K. B. R. WIJAYARATNE
Snap Appliance Inc. (Division of Adaptec Inc.)

This article provides a retrospective of our original paper by the same title in the Proceedings of the First ACM Conference on Multimedia, published in 1993. This article examines the problem of disk scheduling in a multimedia I/O system. In a multimedia server, the disk requests may have constant data rate requirements and need guaranteed service. We propose a new scheduling algorithm, SCAN-EDF, that combines the features of SCAN type of seek optimizing algorithm with an Earliest Deadline First (EDF) type of real-time scheduling algorithm. We compare SCAN-EDF with other scheduling strategies and show that SCAN-EDF combines the best features of both SCAN and EDF. We also investigate the impact of buffer space on the maximum number of video streams that can be supported. We show that by making the deadlines larger than the request periods, a larger number of streams can be supported.

We also describe how we extended the SCAN-EDF algorithm in the PRISM multimedia architecture. PRISM is an integrated multimedia server, designed to satisfy the QOS requirements of multiple classes of requests. Our experience in implementing the extended SCAN-EDF algorithm in a generic operating system is discussed and performance metrics and results are presented to illustrate how the SCAN-EDF extensions and implementation strategies have succeeded in meeting the QOS requirements of different classes of requests.

Categories and Subject Descriptors: D.4.2 [**Operating Systems**]: Storage Management—*secondary storage*; D.4.3 [**Operating Systems**]: File Systems Management

General Terms: Algorithms, Performance

Additional Key Words and Phrases: I/O systems, disk scheduling, multimedia applications, real-time, performance evaluation

1. INTRODUCTION

Future I/O systems will be required to support continuous-media such as video and audio. Continuous media put different demands on the system than data streams such as text. The real-time demands of the requests need to be taken into account in designing a system. In this article, we will use the terms *real-time*, *video*, *continuous media*, and *periodic media* interchangeably to describe

Authors' addresses: A. L. N. Reddy, Department of Electrical Engineering, 214 Zachry, College Station, TX 77843; email: reddy@ee.tamu.edu; J. Wyllie, K56-B3, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120; email: wyllie@almaden.ibm.com; K. B. R. Wijayaratne, Adaptec, Inc., 691 South Milpitas Blvd., Milpitas, CA 95035; email: ravi_wijayaratne@adaptec.com.

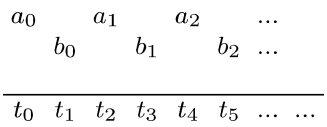

Fig. 1. Two streams of requests.

requests that have constant data rate requirements. We also use the terms *server* and *I/O system* interchangeably.

A real-time request is denoted by two parameters (c, p), where p is the period at which the real-time requests are generated and c is the service time required in each period. When c is a fixed value, it is easy to specify a real-time request with these two variables. But, the disk service time for a request depends on the random components of seek time, latency time and contention for the shared channel for data transfer. Hence, we will specify the real-time requests by specifying the required data rate in kbytes/sec.

The time at which a periodic stream is started is called the *release time* of that request. The time at which a request is to be completed is called the *deadline* for that request. Requests that do not have real-time requirements are termed *aperiodic* requests. Figure 1, illustrates the terminology used in this article. The figure shows two streams of requests a and b that are released at times t_0 and t_1 respectively. Stream a is represented by a string of requests a_0, a_1, a_2, ... and b is represented by b_0, b_1, b_2, ...

In real-time systems, when requests may have to be satisfied within deadlines, algorithms such as earliest deadline first, and least slack time first are used. The earliest deadline first (EDF) algorithm is shown to be optimal [Liu and Layland 1973] if the service times of the requests are known in advance. However, the disk service time for a request depends on the relative position of the request from the current position of the read-write head. The original EDF algorithm assumed that the tasks are preemptable with zero preemption cost and showed that tasks can be scheduled by EDF if and only if the task utilization $\sum_{i=1}^{n} c_i/p_i < 1$. Current disks are however not preemptable. Recently it has been shown that even when the tasks are nonpreemptable, EDF is an optimal policy [Jeffay et al. 1991]. However, due to the overheads of seek time, strict real-time scheduling of a disk arm may result in excessive seek time cost and poor utilization of the disk.

Traditionally, disks have used seek optimization techniques such as SCAN or shortest seek time first (SSTF) for minimizing arm movement in serving requests. These techniques reduce disk arm utilization by serving requests close to the disk arm. The request queue is ordered by the relative position of the requests on the disk surface to reduce seek overheads. Even though these techniques utilize the disk arm efficiently, they may not be suitable for real-time environments since they do not have a notion of time or deadlines in making a scheduling decision.

Another requirement is that the scheduling algorithm should be fair. For example, shortest seek time first is not a fair scheduling algorithm since requests at the edges of the disk surface may get starved. If the scheduling algorithm is not fair, an occasional request in the stream may get starved of service and hence will result in missed deadlines. Hence, we will not use SSTF type algorithms in our discussion. For the same reasons, the policy proposed in Abbott and Garcia-Molina [1990] is not suitable for real-time application.

Available buffer space has a significant impact on the performance of the system. The constant data rate of real-time requests can be provided in various ways. When the available buffer space is small, the request stream can ask for small pieces of data in each period. When the available buffer space is large, the request stream can ask for larger pieces of data with correspondingly larger periods between requests. This trade-off is significant since the efficiency of the disk service is a varying function of

the request size. We will study the impact of buffer space on the performance of various scheduling policies. We also show that by deferring deadlines, which also increases the buffer requirements, the performance of the system can be improved significantly.

To prove the correctness of the schedule, worst-case assumptions about seek and latency overheads have to be made. When worst-case overheads are assumed, random disk service times can be bounded by some constant service time. Another approach to making service times predictable is to make the request size so large that the overheads form a smaller fraction of the request service time. This approach may result in large demands on buffer space. Our approach to this problem is to reduce the overheads in service time by making more efficient use of the disk arm either by optimizing the service schedule and/or by using large requests. By reducing the random overheads, we make the service time more predictable. We utilize deadline extensions to reduce the uncertainties of meeting the deadlines.

In this article, we propose a new scheduling algorithm, SCAN-EDF. SCAN-EDF is a hybrid algorithm that incorporates the real-time aspects of EDF and seek optimization aspects of SCAN, CSCAN and other such seek optimization policies.We will show that SCAN-EDF has good characteristics for supporting multimedia requests. We also study the impact of buffer availability on the number of streams that can be supported.

The PRISM architecture was designed to provide integrated services for periodic, interactive and aperiodic request streams. The interactive requests require prompt service with little latency. Both aperiodic and interactive request streams have random arrival and load patterns and both need starvation-free service from the underlying storage system. For example, an FTP session downloading 1GB of data is an aperiodic stream whereas a flight simulation retrieving an animated media stream is an interactive stream. If there is a visible delay in fetching the data from the storage system, the user will experience undesirable glitches in the simulation. Therefore, the interactive stream requires prompt service from the underlying storage system.

We also describe how we extended SCAN-EDF algorithm to schedule Variable Bit Rate (VBR) periodic streams in the PRISM architecture. The scope of our discussion is limited to the adaptations needed for the SCAN-EDF algorithm to service periodic streams to minimize their missing deadlines in a mixed media server environment and to prevent periodic request service from starving other service classes of service. Lastly, we discuss the implementation strategies used to achieve the above two objectives and to satisfy the architectural requirements in implementing periodic stream servers in a generic operating system.

The remainder of this article is organized as follows: Section 2 describes the SCAN-EDF algorithm. We present buffer usage strategies to facilitate and optimize the SCAN-EDF algorithm in section 3. Section 4 presents simulation results of the SCAN-EDF algorithm. Section 5 analyzes the theory behind the SCAN-EDF algorithm and correlates our findings to the results presented. Section 6, presents adaptations to SCAN-EDF algorithm used in the PRISM architecture. We describe our approach in implementing the extended SCAN-EDF algorithm in section 7. Section 8 presents the effects of the implemented extended SCAN-EDF algorithm in the PRISM architecture. And lastly, sections 9 and 10 discuss related work and present our conclusions.

2. SCAN-EDF SCHEDULING ALGORITHM

The SCAN-EDF disk scheduling algorithm combines seek optimization techniques and EDF in the following way. Requests with earliest deadline are served first. But, if several requests have the same deadline, these requests are served by their track locations on the disk or by using a seek optimization scheduling algorithm for these requests. In this article, deadlines are always in multiples of request periods. This strategy combines the benefits of both real-time and seek-optimizing scheduling algorithms.

Requests with earlier deadlines are served first, but requests with the same deadline make use of seek optimization techniques to reduce the disk utilization.

SCAN-EDF applies seek optimization to only those requests having the same deadline. Its efficiency depends on how often these seek optimizations can be applied, or on the fraction of requests that have the same deadlines. The following techniques make it possible for various requests to have the same deadlines. SCAN-EDF prescribes that requests have release times that are multiples of the period p. This results in all requests having deadlines that are multiples p, which enables requests to be grouped in batches and served accordingly. When streams have different data rate requirements, SCAN-EDF can be combined with a periodic fill policy [Yee and Varaiya 1992] to let all the requests have the same deadline. Requests are served in a cycle with each request getting an amount of service time proportional to its required data rate, the length of the cycle being the sum of the service times of all the requests. All the requests in the current cycle can then be given a deadline at the end of the current cycle. These two techniques enhance the possibility of applying seek optimization in SCAN-EDF.

A more precise description of the algorithm is given below.

SCAN-EDF algorithm
Step 1: Let T = set of requests with the earliest deadline
Step 2: if $|T| = 1$, (there is only a single request in T), service that request.
else let t_1 be the first task in T in scan direction, service t_1.
go to **Step 1**.

The scan direction can be chosen in several ways. In Step 2, if the tasks are ordered with the track numbers of tasks such that $N_1 <= N_2 <= \cdots <= N_l$, then we obtain a CSCAN type of scheduling where the scan takes place only from smallest track number to the largest track number. If tasks are ordered such that $N_1 >= N_2 >= \cdots >= N_l$, then we obtain a CSCAN type of scheduling where the scan takes place only from largest track number to the smallest track number. If the tasks can be ordered in either of the above forms depending on the relative position of the disk arm, we get (elevator) SCAN type of algorithm.

SCAN-EDF can be implemented with a slight modification to EDF. Let D_i be the deadlines of the tasks and N_i be their track positions. Then, the deadlines can be modified to be $D_i + f(N_i)$, where $f()$ is a function that converts track numbers of the tasks into small perturbations to the deadlines. The perturbations have to be small enough such that $D_i + f(N_i) > D_j + f(N_j)$, if $D_i > D_j$ and requests i and j are ordered in the SCAN order when $D_i = D_j$. We can choose $f()$ in various ways. Some of the choices are $f(N_i) = N_i/N_{max}$ or $f(N_i) = N_i/N_{max} - 1$, where N_{max} is the maximum track number on the disk or some other suitably large constant. For example, let tasks A, B, C and D have deadlines 500, 500, 500, and 600 respectively and ask for data from tracks 347, 113, 851, and 256 respectively. If $N_{max} = 1000$, the modified deadlines of A, B, C and D become 499.347, 499.113, 499.851 and 599.256 respectively when we use $f(N_i) = N_i/N_{max} - 1$. When these requests are served by their modified deadlines, requests A, B and C are served in the SCAN order of B, A and C and request D is served later.

3. BUFFER SPACE TRADE-OFF

Real-time requests typically need some kind of response before the next request is issued. Hence, the deadlines for the requests are made equal to the periods of the requests. The multimedia I/O system needs to provide a constant data rate for each request stream. This constant data rate can be provided in various ways. When the available buffer space is small, the request stream can ask for small pieces of data in each period. When the available buffer space is large, the request stream can ask for larger pieces of data with correspondingly larger periods between requests. This trade-off is significant since

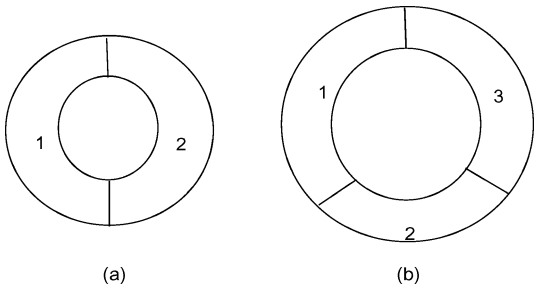

Fig. 2. Arrangement of buffers for a single real-time stream.

the efficiency of the disk service is a varying function of the request size. The disk arm is more efficiently used when request sizes are large; hence, it may be possible to support more multimedia streams on a single disk. A stream of requests described by (c,p) supports the same data rate as a stream of requests of (2c,2p) if larger buffers are provided.

Each request stream requires a buffer for the consuming process and one buffer for the producing process (disk). If we decide to issue requests at the size of S, then the buffer space requirement for each stream is $2S$. If the I/O system supports n streams, the total buffer space requirement is $2nS$. Figure 2(a) shows the usage of the buffers for this process. In Figure 2(a), while buffer 1 is being consumed, an outstanding request tries to fill buffer 2 and when buffer 2 is being consumed, an outstanding request tries to fill buffer 1.

There is another trade-off that is possible. The deadlines of requests need not be chosen equal to the periods of the requests. For example, we can defer the deadlines of the requests by a period and make the deadlines of the requests equal to $2p$. This gives more time for the disk arm to serve a given request and may allow more seek optimizations than is possible when the deadlines are equal to the period p. This results in a scenario where the consuming process is consuming buffer 1, the producing process (disk) is reading data into buffer 3 and buffer 2 is filled earlier by the producer and waiting consumption. Hence, this raises the buffer requirements to $3S$ for each request stream. This arrangement is shown in Figure 2(b) where three buffers 1, 2 and 3 are used circularly to satisfy the requests of a single stream. In general, when the requests are allowed to have deadlines that are mp, the buffer requirements for each stream are $(m+1)S$, where S is the size of the request in each period. We will show in the next section that this strategy has significant benefits. The extra time available for serving a given request allows seek optimization techniques to be applied more often to the request queue at the disk. This results in more efficient use of the disk arm and as a result, a larger number of request streams can be supported at a single disk. A similar technique called work-ahead is utilized by Anderson et al. [1991].

It is shown that when all the deadlines are extended by a multiple of the periods, monotonic scheduling achieves higher useful utilization of the resource [Lehoczky 1990]. It is shown that if the periods of all requests are extended by the largest period, a modified rate monotonic scheduling algorithm is optimal [Shih et al. 1992]. Both these studies assume that tasks are preemptable.

Both techniques, that is larger requests with larger periods and delayed deadlines, increase the start up latency of real-time streams. When deadlines are delayed, the multimedia data stream cannot be consumed until two buffers are filled as opposed to waiting for one filled buffer when deadlines are equal to periods. When larger requests are employed, similarly longer time is taken before a buffer is filled and hence a longer time before the multimedia stream can be started. Larger requests increase the response time for aperiodic requests as well since the aperiodic requests will have to wait longer

Table I. Disk Parameters

Time for one rotation	11.1 ms
Avg. seek	9.4 ms
sectors/track	84
sector size	512 bytes
tracks/cylinder	15
cylinders/disk	2577
seek cost function	nonlinear
Min. seek time s_0	1.0 ms

behind the current real-time request that is being served. The improved efficiency of these techniques needs to be weighed against higher buffer requirements and higher start up latency.

We discussed two techniques, deferring deadlines and employing larger requests, in which increased buffer space can improve the efficiency of disk scheduling. Quantitative evaluation and comparison of these techniques will be provided in a later section.

4. PERFORMANCE EVALUATION

In this section, we present a simulation model and results to evaluate the algorithms.

4.1 Simulation Model

A disk with the parameters shown in Table I is modeled. It is assumed that the disk uses split-access operations or zero latency reads. In split-access operation, the request is satisfied by two smaller requests if the read-write head happens to be in the middle of the requested data at the end of the seek operation. The disk starts servicing the request as soon as any of the requested blocks comes under the read-write head. For example, if a request asks for reading blocks numbered 1,2,3,4 from a track of eight blocks 1,2,. . .,8, and the read-write head happens to get to block number 3 first, then blocks 3 and 4 are read, blocks 5,6,7,8 are skipped over and then blocks 1 and 2 are read. In such operation, a disk read/write of a single track will not take more than one single revolution. Split access operation of the disk has been shown to improve the response time of the disk considerably in Reddy [1992]. Split-access operation, besides reducing the service time of a request, also helps in reducing the variability in service time.

Each real-time request stream is assumed to require a constant data rate of 150 KB/sec. This roughly corresponds to the data rate requirements for a 1x CD ROM data stream. Each request stream is modeled by an independent request generator. The release times of the requests are dependent on the scheduling policy employed as described below. The number of streams is a parameter to the simulator.

Aperiodic requests are modeled by a single aperiodic request generator. Aperiodic requests are assumed to arrive with an exponential distribution. The mean time between arrivals is varied from 25 ms to 200 ms. If we allow unlimited service for the aperiodic requests, a burst of aperiodic requests can disturb the service of real-time requests considerably. It is necessary to limit the number of aperiodic requests that may be served in a given period of time. A separate queue could be maintained for these requests and these requests can be released at a rate that is bounded by a known rate. A multimedia server will have to be built in this fashion to guarantee meeting the real-time schedules. Hence, we decided to model the arrival of aperiodic requests by a single request generator. In our model, if the aperiodic requests are generated faster than they are being served, they are queued in a separate queue. This model of service is shown in Figure 3.

The service policy for aperiodic requests depended on the scheduling policy employed. In EDF, SCAN-EDF and STAG EDF, they are served using the immediate server approach [Lin and Tarng 1991] where the aperiodic requests are given higher priority over the periodic real-time requests. The service schedule for these three policies allows a certain number of aperiodic requests each period and when sufficient

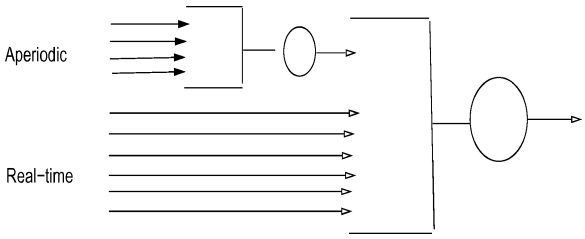

Fig. 3. Service model for serving real-time and aperiodic requests.

number of aperiodic requests are not present, the real-time requests make use of the remaining service period. This policy of serving aperiodic requests is employed so as to provide reasonable response times for both aperiodic and periodic requests. This policy is in contrast to earlier approaches where the emphasis has been only on providing real-time performance guarantees. In CSCAN, aperiodic requests are served in SCAN order and in PCSCAN, aperiodic requests are served as explained earlier.

Each aperiodic request is assumed to ask for a track of data. With split-access operations, the service time for a request asking for less than or equal to a track of data is bounded by the service time for a track. Hence, this assumption is equivalent to assuming that no aperiodic request asks for more than a track of data at a time. The request size for the real-time requests is varied among 1, 2, 5, or 15 tracks. The effect of request size on the number of supportable streams is investigated. The period between two requests of a real-time request stream is varied depending on the request size to support a constant data rate of 150 KB/sec. The requests are assumed to be uniformly distributed over the disk surface.

Five scheduling policies are modeled. EDF is the strict earliest deadline first scheduling policy. SCAN-EDF is the policy proposed earlier in Section 2. We used $f(N_i) = N_i/N_{max} - 1$ to modify the deadlines. CSCAN is the policy where the disk arm moves from the outermost request to the innermost request. After serving the innermost request, the disk arm jumps back to the outermost request waiting to be served. PCSCAN is a slight modification of CSCAN where aperiodic requests that are less than half the number of tracks behind the disk arm are served immediately that is, out of the CSCAN arm movement order. STAGEDF is an EDF policy with staggered release times. For this policy, the release times of request streams are equally placed between 0 and the period p. For all the other policies, the release times are at time zero. The relative merits of the various scheduling policies are studied.

Two systems, one with deadlines equal to the request periods and the second with deadlines equal to twice the request periods are modeled. A comparison of these two systems gives insight into how performance can be improved by deferring the deadlines.

Two measures of performance are studied. The number of real-time streams that can be supported by each scheduling policy is taken as the primary measure of performance. We also look at the response time for aperiodic requests. The response time for aperiodic requests cannot be unduly large. A good policy will offer good response times for aperiodic requests while supporting a large number of real-time streams.

Each experiment involved running 50,000 requests of each stream. The maximum number of supportable streams n is obtained by increasing the number of streams incrementally until deadlines cannot be met. Twenty experiments were conducted, with different seeds for random number generation, for each point in the figures. The minimum among these values is chosen as the maximum number of streams that can be supported. Each point in the figures is obtained in this way. The minimum is chosen (instead of the average) in order to guarantee the real-time performance. We confirm the simulations later through analysis.

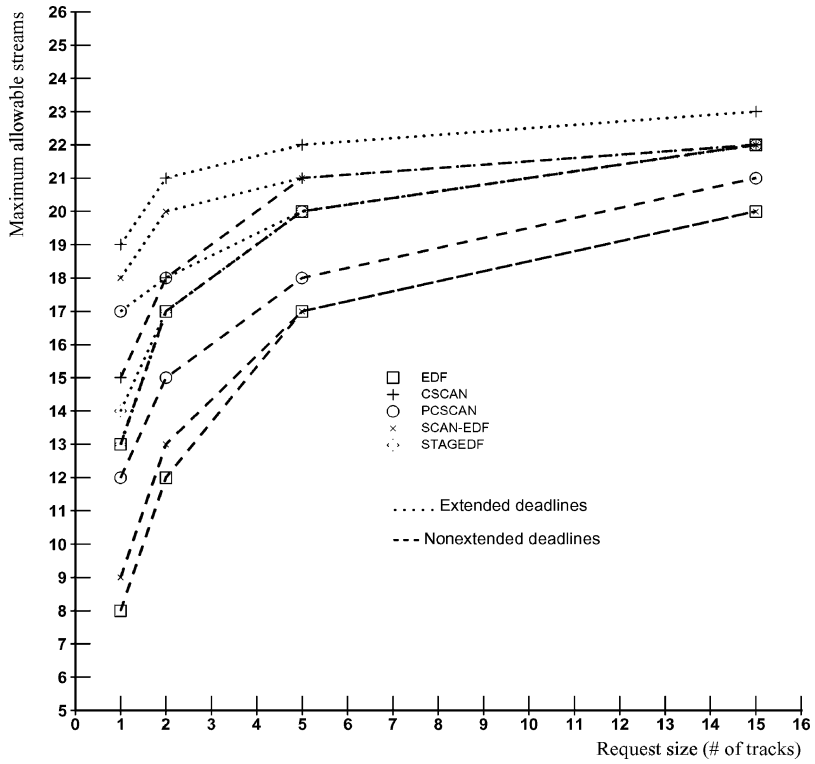

Fig. 4. Performance of different scheduling policies.

4.2 Results

4.2.1 *Maximum Number of Streams.* Figure 4 shows the results from simulations. The dark lines correspond to a system with normal deadlines (=p) and the other lines are for the system where deadlines are extended.

It is observed that deferring deadlines improves the number of supportable streams significantly for all the scheduling policies. The performance improvement ranges from 4 streams for CSCAN to 9 streams for SCAN-EDF at a request size of 1 track.

When deadlines are deferred, CSCAN has the best performance. SCAN-EDF has performance very close to CSCAN. EDF has the worst performance. EDF scheduling results in random disk arm movement and this explains the poor performance of this policy. Figure 4 clearly shows the advantage of utilizing seek optimization techniques.

Figure 4 also presents the improvements that are possible by increasing the request size. As the request size is increased from 1 track to 15 tracks, the number of supportable streams keeps increasing. The knee of the curve seems to be around 5 tracks or 200 kbytes. At larger request sizes, the different scheduling policies make relatively less difference in performance. At larger request sizes, the transfer time dominates the service time. When seek time overhead is a smaller fraction of service time, the different scheduling policies have less scope for optimizing the schedule. Hence, the performance of all the scheduling policies does not differ significantly at larger request sizes. At a request size of 5 tracks, that is, 200 kbytes/buffer, minimum of 2 buffers/stream corresponds to 400 kbytes of buffer space per

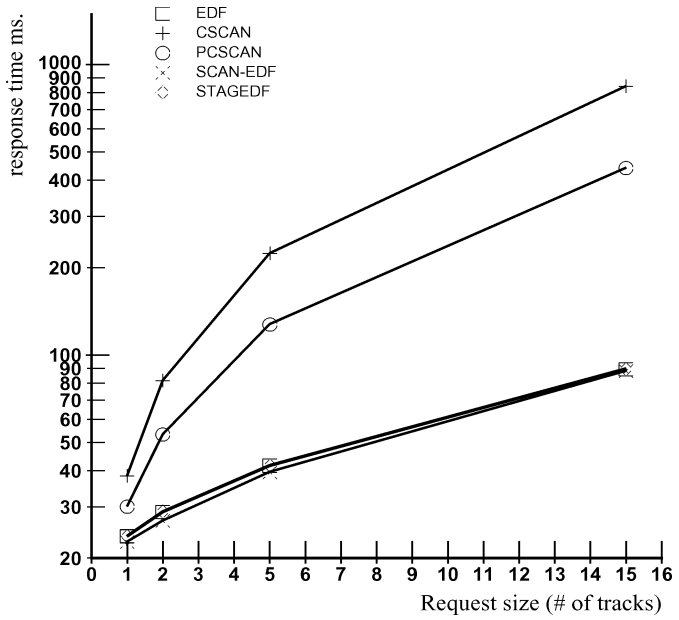

Fig. 5. Aperiodic response time with different scheduling policies.

stream. This results in a demand of 400 kbytes * 20 = 8Mbytes of buffer space at the I/O system for supporting 20 streams. If deadlines are deferred, this corresponds to a requirement of 12 Mbytes. When that buffer space is unavailable, smaller request sizes need to be considered.

At smaller request sizes, deferring the deadlines has a better impact on performance than increasing the buffer size. For example, at a request size of 1 track and deferred deadlines (with buffer requirements of 3 tracks) EDF supports 13 streams. When deadlines are not deferred, at a larger request size of 2 tracks and buffer requirements of 4 tracks, EDF supports only 12 streams. A similar trend is observed with other policies as well. A similar observation also seems to hold when request sizes of 2 and 5 tracks are compared.

Figure 4 also shows that staggering the deadlines of the requests had an impact on the number of supportable streams when the deadlines are not extended. When deadlines are not extended, STAGEDF has considerably better performance than EDF. But, when deadlines are extended, the two policies have almost no difference in performance. In general, staggering deadlines is not exactly feasible. When all the requests arrive at once, it is possible to make the release times uniformly distributed between 0 and p. In our simulations, we assumed this scenario. But when requests are incrementally allowed to arrive, this uniform distribution will not be possible.

4.2.2 *Aperiodic Response Time.* Figure 5 shows the response time for aperiodic requests. The figure shows the aperiodic response time when 8, 12, 15, and 18 real-time streams are being supported in the system at request sizes of 1, 2, 5, and 15 tracks respectively. It is observed that CSCAN has the worst performance and SCAN-EDF has the best performance. With CSCAN, on an average, an aperiodic request has to wait for half a sweep for service. This may result in waiting behind half the number of real-time requests. In SCAN-EDF, EDF, and STAGEDF, aperiodic requests are given higher priorities by giving them shorter deadlines (e.g., 100 ms from the issuing time). In these strategies, requests with shorter deadlines get higher priority. As a result, aperiodic requests typically wait behind only the

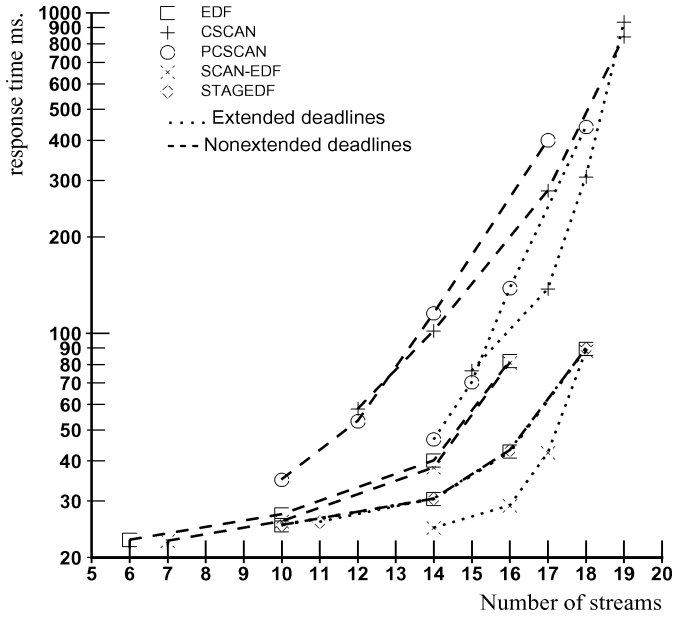

Fig. 6. Aperiodic response time as a function of number of streams.

current request that is being served. As a result, aperiodic requests have to wait, on an average, for about half the service time of a single real-time request. Among these three policies, the slightly better performance of SCAN-EDF is due to the lower arm utilizations.

From Figures 4 and 5, it is seen that SCAN-EDF performs well under both measures of performance. CSCAN performs well in supporting real-time requests but does not have very good performance in serving the aperiodic requests. EDF does not perform very well in supporting real-time requests but offers good response times for aperiodic requests. SCAN-EDF supports almost as many real-time streams as CSCAN and at the same time offers the best response times for aperiodic requests. When both the performance measures are considered, SCAN-EDF has better characteristics.

When deadlines are deferred, smaller request sizes can be used to support the same number of real-time streams at the disk. This improves the aperiodic request response time besides reducing the demand on the buffer space needed at the I/O system. When deadlines are not deferred, the aperiodic requests have to wait behind larger requests when supporting the same number of real-time streams. Hence, the aperiodic response time suffers. This effect is shown in Figure 6, which shows the aperiodic response time with various scheduling policies at 80% of the maximum number of supportable streams. Figure 6 shows that, it is better to defer deadlines than to use a larger request size since better aperiodic response times are obtained.

Figure 7 shows the effect of aperiodic request arrival rate on the number of real-time streams that can be supported. It is observed that the aperiodic request arrival rate has a significant impact on all the policies. Except for CSCAN, all other policies support less than 5 streams at an interarrival time of 25 ms. Figure 7 shows that the interarrival time of aperiodic requests should not be below 50 ms if more than 10 real-time streams need to be supported at the disk. CSCAN treats all requests equally and hence a higher aperiodic request arrival time only reduces the time available for the real-time request streams and does not alter the schedule of service. In other policies, since aperiodic requests are given higher priorities, a higher aperiodic request arrival rate results in less efficient arm utilization due to

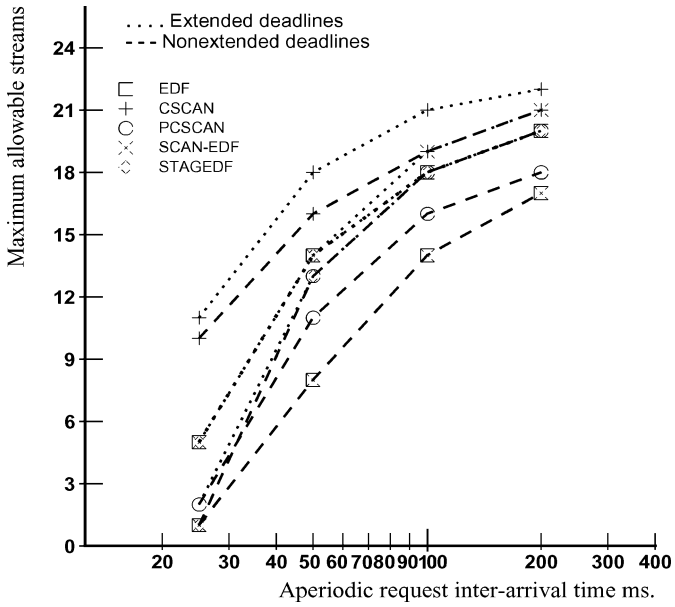

Fig. 7. Effect of aperiodic request arrival rate on the number of streams.

more random arm movement. Hence, other policies see more impact on performance due to the higher aperiodic request arrival rate.

4.2.3 *Performance with an Array.* Figure 8 shows the performance of various policies, at an aperiodic request inter-arrival time of 200 ms, when an 8-disk array is employed in a RAID3 configuration without parity protection (i.e., all 8 disk arms are tied together with a transfer rate of 8 times that of a normal disk). This configuration is known to offer good performance for sequential transfers [Patterson et al. 1988; Kim 1986; Salem and Garcia-Molina 1986; Reddy and Banerjee 1989]. Performance of all the policies is improved by nearly 8-fold. It is observed that the performance differences between SCAN-EDF and EDF are higher with an array than with a single disk. This improvement is primarily due to the fact that seek time is a bigger fraction of the service time in the array.

4.2.4 *Multiple Data Rates.* Figure 9 shows the performance of various scheduling policies when requests with different data rates are served. The simulations modeled an equal number of three different data rates of 150 kB/sec, 8 kB/sec and 176 kB/sec with aperiodic requests with an inter-arrival time of 200 ms. Even with multiple data rate requirements, SCAN-EDF supports almost as many streams as CSCAN and more than EDF.

5. ANALYSIS OF SCAN-EDF

In this section, we will present an analysis of the SCAN-EDF policy and show how request service can be guaranteed. We assume that the disk seek time can be modeled by the following equation s(m) = $s_0 + m * s_1$, where $s(m)$ is the seek time for m tracks, and s_0 is the minimum seek time. This equation assumes that the seek time is a linear function of the number of tracks. This simplifying assumption makes analysis easy (in simulations earlier, we used the actual measured seek function of an IBM disk). The value of s_1 can be chosen such that the seek time function $s(m)$ gives an upper bound on the actual seek time. Let M denote the number of tracks on the disk and T the track capacity. We will denote the

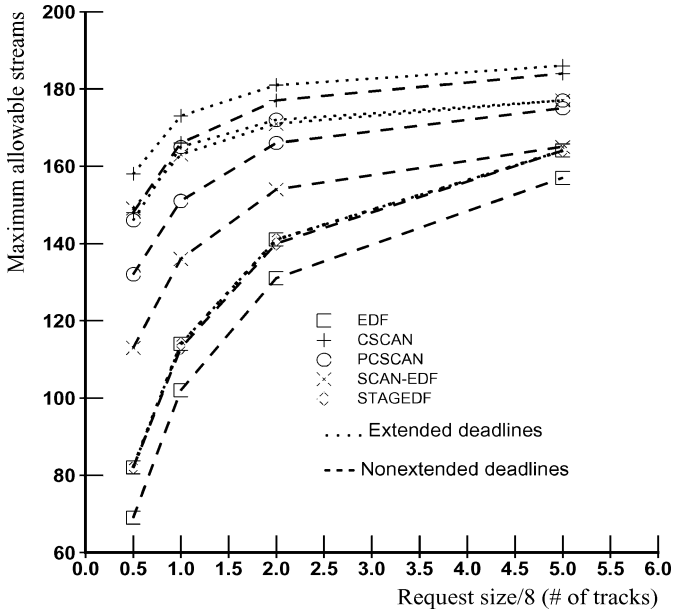

Fig. 8. Performance of various policies with a disk array.

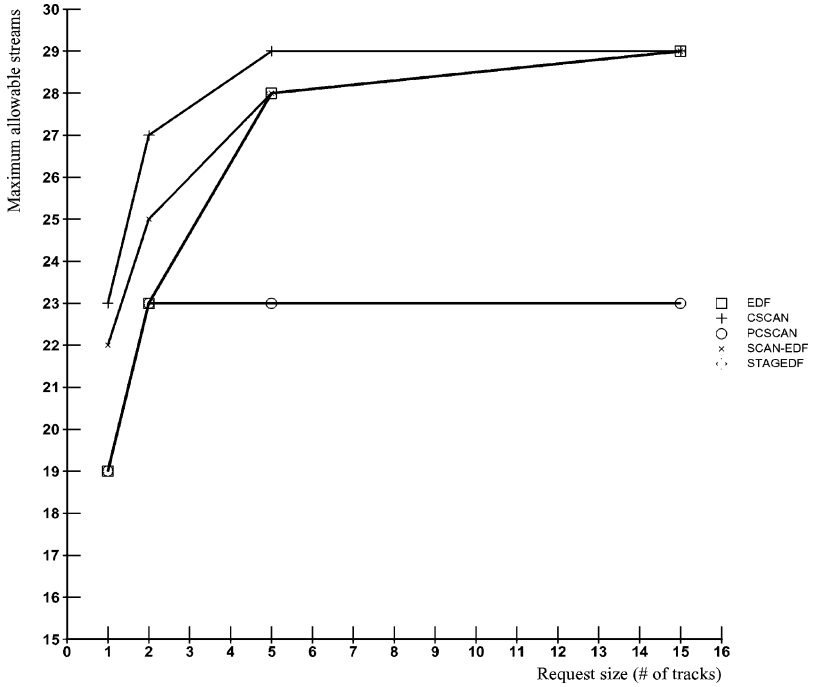

Fig. 9. Performance of various policies with multiple data rates.

required data rate for each stream by C. We also assume that the disk requests are issued at a varying rate, but always in multiples of track capacity. Let kT be the request size. Since C is the required data rate for each stream, the period for a request stream $p = kT/C$. If r denotes the data rate of the disk in bytes/sec, $r = T/(rotation\ time)$. We assume the disk employs a split-access operation and hence no latency penalty. This analysis also assumes there are no aperiodic requests. These assumptions are made so that we can establish an upper bound on performance.

SCAN-EDF serves requests in batches. Each batch is served in a scan order for meeting a particular deadline. We assume that the batch of n requests are uniformly placed over the disk surface. Hence the seek time cost for a complete sweep of n requests can be given by $s_1 * M + n * s_0$. This assumes that the disk arm sweeps across all M tracks in serving the n requests. The read time cost for n requests is given by $n * kr$. The total time for one sweep is the time taken to serve the n requests plus the time taken to move the disk arm back from the innermost track to the outermost track. This innermost track to outermost track seek takes $s_0 + M * s_1$ time. Hence, the total time for serving one batch of requests is given by $Q = (n * s_0 + M * s_1 + n * kr) + s_0 + M * s_1 = n * (s_0 + kr) + 2M * s_1 + s_0$. The worst-case for a single stream results when its request is the first request to be served in a batch and is the last request to be served in the next batch of requests. This results in roughly 2Q time between serving two requests of a stream. This implies the number of streams n is obtained when $p = 2Q$ or $n = (kT/C - 4M * s_1 - 2 * s_0)/2 * (s_0 + kr)$. However, this bound can be improved if we allow deadline extension. If we allow the deadlines to be extended by one period, the maximum number of streams n is obtained when $n = (kT/C - 2M * s_1 - s_0)/(s_0 + kr)$.

The time taken to serve a batch of requests through a sweep, using SCAN-EDF, has little variance. The possible variances of individual seek times could add up to a possible large variance if served by a strict EDF policy. SCAN-EDF reduces this variance by serving all the requests in a single sweep across the disk surface. SCAN-EDF, by reducing the variance, reduces the time taken for serving a batch of requests and hence supports a larger number of streams. This reduction in the variance of service time for a batch of requests has a significant impact on improving service time guarantees. Larger request sizes, and split-access operation of the disk arm also reduce the variance in service time by limiting the variable portions of the service time to a smaller fraction.

Figure 10 compares the predictions of analysis with results obtained from simulations for extended deadlines. For this experiment, aperiodic requests were not considered and hence the small difference in the number of streams supportable by SCAN-EDF from Figure 4. It is observed that the analysis is very close to the simulation results. The error is within one stream for all request sizes examined.

6. SCAN-EDF IN PRISM ARCHITECTURE

In PRISM, we extended SCAN-EDF to handle a third class of requests, namely interactive requests that require short latencies. The conceptual diagram of the PRISM architecture is illustrated in Figure 11, which closely resembles the original SCAN-EDF approach shown in Figure 3. The periodic, aperiodic and interactive requests go through an admission controller which admits streams based on a static bandwidth allocation to each class [Wijayaratne and Reddy 2000]. The disk I/O scheduler schedules these requests to achieve 4 main objectives.

—Meet soft real-time requirements of VBR periodic streams.

—Provide prompt response for interactive streams.

—Provide best effort starvation free service for aperiodic streams

—Maximize storage system throughput and fairness among service classes.

A detailed description and analysis of the PRISM disk scheduling algorithm to achieve the above goals is described in Wijayaratne [2001]. What is relevant to this discussion is how we adapted SCAN-EDF

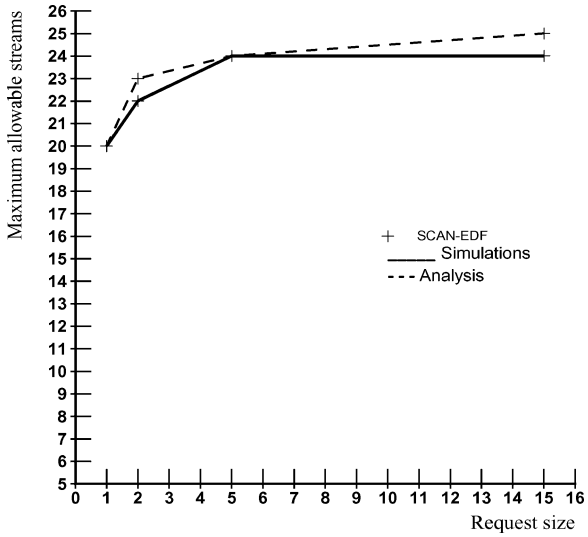

Fig. 10. Comparison of analysis with simulation results.

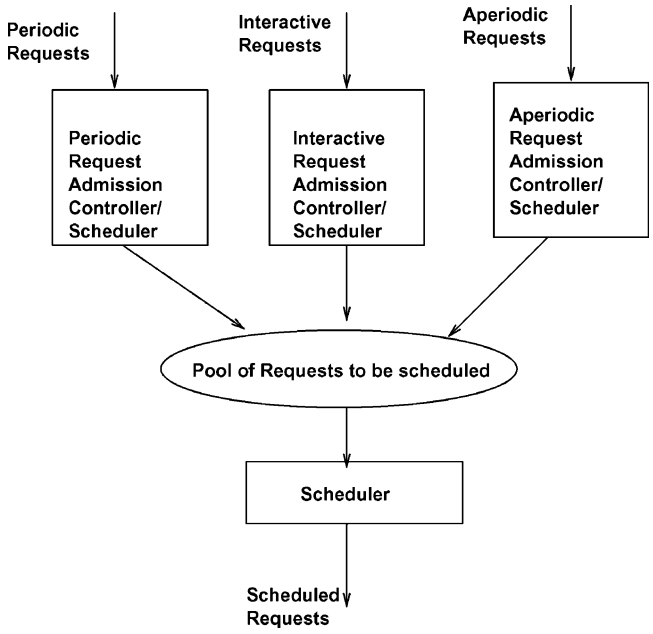

Fig. 11. Basic model for supporting multiple QOS levels.

to meet the additional requirements of interactive requests and how we dealt with practical issues in implementing SCAN-EDF. While periodic requests needed at time $t + 1$ can be assumed to arrive at the scheduler at time $t - 1$ as described in Section 3, aperiodic and interactive request arrival cannot be predetermined and is asynchronous to the scheduling intervals.

We first schedule the periodic requests using the SCAN-EDF algorithm as described in Section 2. Before scheduling any other type of request we calculate the slack time available in the current scheduling period. If the slack time is less than zero, non-periodic requests are not scheduled. When SCAN-EDF order needs to be disturbed for scheduling the next request, we charge it a worst-case disk seek time in the process of calculating the slack time. These adaptations to SCAN-EDF would potentially starve aperiodic and interactive requests if periodic requests are admitted without any policing. However, the admission controller restricts the periodic streams to its static system-wide bandwidth based on conservative service time estimates. Therefore, the possibility of aperiodic and interactive requests getting no service for an extended period is greatly minimized.

SCAN-EDF assumes periodic streams require CBR service. However, later work on video compression has shown the usefulness of using VBR streams for improving quality [Chang and Zakhor 1994]. VBR streams have variable data rate requirements over time. While VBR streams can be treated as CBR streams for scheduling purposes by considering the worst-case data rate requirements, considerable performance degradation due to overcommitment of resources can result with such an approach [Wijayaratne and Reddy 1999b; Shenoy et al. 1999]. PRISM noted that the aggregate load of periodic streams at the device scheduler is what is important for scheduling, rather than individual stream requirements. Hence, periodic admission controller keeps track of aggregate system load of periodic streams and combines this with the load of an incoming request to decide if a periodic stream can be admitted within specified latency bounds without violating the aggregate bandwidth allocated to periodic requests. This approach smooths out demands across multiple streams which provides considerable improvement in performance [Wijayaratne and Reddy 2000].

In PRISM, both periodic and aperiodic requests are scheduled based on the SCAN-EDF algorithm. Aperiodic requests are serviced in SCAN order as long as slack time to service periodic requests exists. The moment the slack time becomes less than zero, all nonperiodic requests are overlooked. Scheduling instances occur in shorter durations than the main period—the period of all periodic requests. On each scheduling instance an interactive request is scheduled. All interactive requests are scheduled in FIFO order to reduce the waiting time of requests, subject to the availability of sufficient slack time to service all scheduled periodic requests. If the SCAN order has to be disturbed to schedule the next interactive request, we select a set of candidate requests clustered around the scheduled interactive request and schedule that cluster in SCAN order.

7. SCAN-EDF IMPLEMENTATION

The PRISM architecture was implemented in the Linux operating system version 2.2. The SCAN-EDF algorithm was used to schedule periodic requests to the disk I/O subsystem. The following architectural support was needed to implement SCAN-EDF algorithm in a server which provides integrated services.

—A QOS context management module.

—File system and kernel support to accommodate prioritizing requests and maintaining QOS parameters of various streams.

—An architecture for layered drivers.

A disk I/O scheduler needs to distinguish to which service class each I/O request belongs in order to prioritize and scheduled requests. Therefore, an association of binding the service class to the disk I/O request is needed inside the operating system. There is a broad selection of techniques in which this association can be made. A file can be labeled as a periodic file by storing its service class in the file's inode or in an extended attribute. However, in such case, all instances of the opened file would receive the labeled service class which is not desirable as a file may be used for different purposes. The process that accesses the stream may be labeled with its service class. The disadvantage of this

approach is that other files accessed by the labeled process will receive the service class of the process that is not desirable. Furthermore, the process context is lost in the disk I/O scheduler level of the operating system, which mostly is executed in an interrupt context. Therefore, the more appropriate strategy is to associate the runtime instance of the opened file structure with the its service class. The QOS context management module provides the system call interface and the association of the runtime opened file structures to their service classes.

A disk I/O request will percolate through the system call interface, the virtual file system switch, the native file system, the buffer caches, the block device abstraction layers and the disk device drivers. The disk I/O request will mutate into allocated pages to buffers to disk I/O request structures. The operating system support is needed to carry the service class all the way from the system call interface to the disk device driver to make the association of the service class to the disk I/O request. Furthermore, at the block device interface level, the service class context of a disk I/O request may be lost due to coalescing of requests. Therefore, attention must be given not to lose the service class context of disk I/O requests at the block device interface layer.

In most file systems, the file metadata is stored in inodes and several indirection blocks. The metadata needs to be read to determine where the data is stored in the disk. Therefore, failure to read metadata, if scheduled in a lower priority could cause periodic streams to miss deadlines. In the PRISM architecture for the SCAN-EDF scheduler, we had a separate thread to prefetch metadata to ensure such failures did not occur. Furthermore, critical system-generated disk I/O such as page swapping needs higher priority than other request streams so that system housekeeping is not disrupted due to prioritizing of the disk I/O requests.

Many operating systems allocate file system buffers to maximize memory utilization and to make sure that all processes receive a fair share of the memory. Therefore, a chunk of memory is allocated and the process is made to wait until that chunk of data is fetched from the disk before another chunk is allocated. This strategy conflicts with our assumption that periodic requests need to be at the scheduler before the main period is started in which the requests are scheduled. Therefore, in PRISM for periodic requests, we allocate buffers for the full request and submit that request to the disk subsystem.

For the SCAN-EDF algorithm, the scheduler has to have the capability of manipulating all disk bound I/O requests. The disk I/O traffic is generated by several entities in a typical operating system such as explicit disk I/O requests (e.g., `read()` or `write()` system calls) or operating system events (e.g., page faults) or house keeping procedures (e.g., buffer cache synchronization processes) [Card et al. 1998]. All these requests go through the operating system block device support layer. Therefore, the best location to place the SCAN-EDF scheduler in PRISM is between the operating system block device support routines and the disk device driver. Most current operating systems provide such a facility even though in PRISM we needed to implement a driver layering facility with call backs to signal scheduling events in Linux 2.2.

A kernel timer facility is necessary for the PRISM disk I/O scheduler [Wijayaratne and Reddy 2000] to function. Most contemporary kernels provide such timer facilities. The scheduler routines that get invoked when a scheduling instance occurs need to be reentrant as these routines would typically be scheduled in the interrupt context. The layered driver call back facility can be used to create a scheduling instance event when the device driver request queues become empty. The scheduler implements its own queues which holds the I/O requests until they are scheduled to the device. A dispatcher invoked at scheduling instance can schedule requests from the internal scheduler queues to the device queues. There needs to be an instance of the scheduler per disk in a multidisk array. If the scheduler is implemented using the layered driver architecture, it can be a self-contained module. Therefore, it is possible to detach the scheduler from the PRISM server and place it at a remote storage device away from the file server. This flexibility will be advantageous if PRISM is deployed in a network-attached storage device or a storage area network environment.

Table II. Testbed Parameters

CPU	PII 233MHz	Average latency	5.5ms
Disk capacity	4.3 Gb	Spindle speed	90 rev/s
Maximum heads	15	Head switch time	890μs
Maximum cylinders	8896	Sectors per track	63
Track to track seek time	3ms	Sector size	512
Average seek time	11ms	Minimum media to buffer bandwidth	68 Mbits/s
Maximum seek time	21ms	Maximum media to buffer bandwidth	135.5 Mbits/s
		Total memory	196 MB

SCAN-EDF and PRISM are based on a server-push notion of service where the server fetches data for periodic clients at regular intervals. However, most file systems, including NFS used in PRISM, are based on a client-pull architecture where data is only fetched when requested by the client. To bridge this gap, a prefetch thread is created for each active periodic stream. The prefetch thread conforms to the server-push scheduling order and when the client requests arrive at the server, they will see buffer hits, obviating the need for disk accesses on request arrival.

File systems typically require metadata accesses for accessing file data. These accesses need to be accounted for in keeping track of resource usage and availability. Metadata accesses are initiated by the kernel and cannot be directly associated with a service class without extensive modifications to the kernel. However, metadata accesses need at least the same level of service as the data file. PRISM solves this problem by allowing metadata accesses as interactive requests and preallocating system resources for these accesses. Our experiments showed that because of high buffer hits to metadata, about 5–10% of system resources are sufficient for serving metadata requests.

8. EXTENDED SCAN-EDF PERFORMANCE

The PRISM implementation was run on a system with a configuration listed in Table II. In our testbed, periodic applications read 50 Mb MPEG files. The access pattern was derived after modifying the MPEG frame traces obtained from Rose [1995]. To read the 50 Mb file according to the access trace, the experiment had to be run between 2.5 to 5 minutes. The block size for periodic streams was set to 64 Kb. The aperiodic and interactive access streams were simulated by bursty traffic sources. The off time for these sources was randomly selected with an average of 70 ms. The number of data bytes requested on each interval was randomly distributed between 4 and 12 Kb. These random reads simulate typical file system accesses. The performance data, unless otherwise stated, was measured at the application. The admission controllers were disabled when required.

Where relevant, we compared the performance of the PRISM architecture disk I/O scheduler with Linux operating system version 2.2 running on the same hardware platform.

We briefly present results from real experiments on PRISM. The details of the configurations and the tests can be found elsewhere [Wijayaratne 2001]. Figure 12 illustrates the influence of the aperiodic traffic on the periodic streams. There were 4 periodic streams and 10 interactive streams in the system. The fraction of deadlines missed for both I/O requests and bytes requested was less than 2%. For Linux, the percentage of data and I/O requests missing deadlines increased up to 25.5% for I/O requests and 34.9% for bytes requested. Similar performance figures were experienced with the variation of the number of interactive threads. These results clearly demonstrate that in `adapted` SCAN-EDF the periodic streams are well insulated from aperiodic and interactive traffic. Increases in traffic of either service class cannot cause periodic streams to miss deadlines.

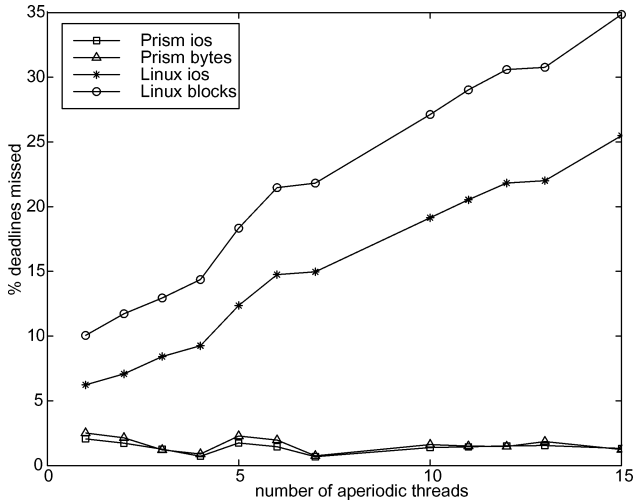

Fig. 12. Influence of aperiodic traffic.

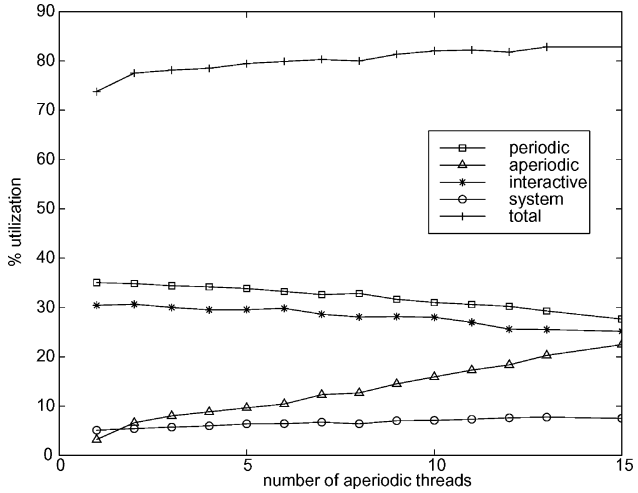

Fig. 13. Influence of aperiodic traffic on disk I/O bandwidth utilization.

Figure 13 illustrate how various service classes share the disk I/O bandwidth. In this experiment, periodic streams were limited by an admission controller to utilize only 50% of the I/O bandwidth. The interactive streams were allocated 25% of the bandwidth by a rate controller. The amount of aperiodic traffic cannot considerably influence the periodic or interactive disk utilization. However, aperiodic disk utilization increases with the aperiodic load demonstrating the work conserving nature of the PRISM disk scheduler. Although periodic streams have an allocated share of 50% of the bandwidth, the disk utilization for periodic streams is about 40%. This underutilization is primarily due to the VBR nature of the streams and the use of worst-case service time estimates in the admission controllers. Because interactive requests get preferred best-effort service, the interactive requests receive more than the allocated share of bandwidth. In Section 6, we stated that the SCAN-EDF schedule will disturb the schedule for interactive requests. A full surface seek time is charged to interactive request in such

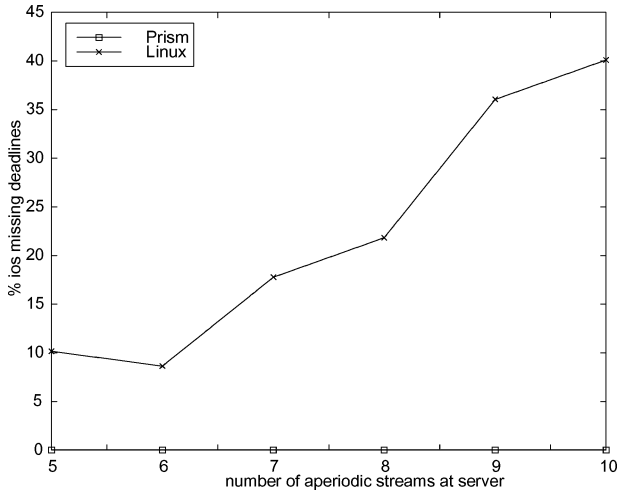

Fig. 14. SCAN-EDF performance on NFS.

cases to be conservative in the slack time calculation. The additional seek time is charged to interactive requests though a cluster of requests around the interactive request benefit from the additional seek. Therefore, interactive requests seem to utilize more than its allocated share of bandwidth.

Figure 14 illustrates the periodic stream performance for NFS on PRISM. We had one periodic stream active at the NFS client. The periodic block size was 64 Kb. Up to 2.5 seconds worth of data was buffered at the client before the periodic stream started to counter the network delay variation. The experiment was conducted for PRISM and Linux version 2.2. For PRISM, no blocks missed deadlines, in other words, all the data points are on the x-axis in the figure. However, for Linux the fraction of data blocks missing deadlines increased from 10.15% to 40.10%.

9. DISCUSSION AND RELATED WORK

Some of the parameters considered in our original paper seem dated now, within a time of little more than 10 years. For example, the considerations based on 1.5 Gb disks, and the availability of 12 Mb buffer space seem unimportant with currently available disks and memory. At the same time, our original paper considered supporting 1.5 Mbps CBR streams. DVD quality streams require 3–10 Mbps and HDTV streams require even higher data rates (7-22 Mbps). As the hardware characteristics improve, our quality requirements tend to improve and the SCAN-EDF algorithm seems even more relavent today. The realization of SCAN-EDF in practice in PRISM required a number of innovations beyond the basic algorithm, highlighting the difficulties in bringing theory into practice.

Since the publication of the SCAN-EDF paper [Reddy and Wyllie 1993], disk I/O scheduling in multimedia servers has received significant attention. This body of work can be broadly categorized under: (a) algorithms for scheduling video streams, (b) algorithms for scheduling multiple classes of requests including video streams, and (c) architectural support for QOS aware scheduling.

Typically, systems that need to schedule only video streams are video-on-demand servers. In addition to meeting deadlines of requests, they may have added requirement of providing premium video services such as rewind, fast forward, pause etc. [Reddy and Haskin 1996; Yu et al. 1993; Chen et al. 1994, 1995; Jayanta et al. 1994]. In the Tiger-Shark video server from IBM [Haskin 1998], the disk requests are scheduled based on the EDF algorithm. Many algorithms extended the EDF algorithm by making scheduling decisions based on read/write head seek optimization strategies [Chen et al. 1991; Abbott

and Garcia-Molina 1990; Chang et al. 2000]. Most of these algorithms balance the demands for fairness of scheduling, meeting request deadlines, buffer requirements and seek optimization. In the Tiger video server from Microsoft, the requests are scheduled cyclically among several distributed storage nodes [Bolosky et al. 1997]. Scheduling considerations of VBR video streams are evaluated in Makaroff et al. [1997], Wijayaratne and Reddy [1999b], and Shenoy et al. [1999].

Mixed media schedulers deal with the general problem of scheduling the periodic video request streams along with the demands of other service classes. These schedulers may employ multiple levels of control/scheduling. One level employs class-specific controllers/schedulers while another level may coordinate across multiple classes of requests [Shenoy and Vin 1998; Wijayaratne and Reddy 1999a; Lund and Goebel 2003]. In PRISM, the class specific scheduler is combined with the admission controllers. In storage system architectures presented in Martin et al. [1996], Rompogiannakis et al. [1998], and Bosch et al. [1999], single layer disk schedulers reschedule disk I/O requests based on specific needs of the service classes. The schedulers employ dynamic request priorities based on available slack time for requests, deadlines, consumed quota of allocated bandwidth, and the proximity to the disk read/write head [Shenoy and Vin 1998; Wijayaratne and Reddy 1999a; Lund and Goebel 2003; Bruno et al. 1999; Anderson et al. 1992; Rompogiannakis et al. 1998]. A theoretical background in making disk I/O scheduling decisions based on multiple criteria is presented in Mokbel et al. [2004]. Some mixed media schedulers give higher priorities to aperiodic requests over periodic requests based on the immediate server approach used in the SCAN-EDF algorithm [Martin et al. 1996] whereas others are more oriented toward servicing deadline sensitive data and service aperiodic streams only based on the available slack times [Anderson et al. 1992].

Multimedia storage architectures differ in how they deviate from standard storage architectures to provide QOS sensitive service to data request streams. A summary of such special architectural considerations is presented in Plagemann et al. [2000]. For special treatment of data access streams, their QOS context needs to be established and the binding to the access streams maintained as long as the data session is active. In video servers presented in Freedman and DeWitt [1995], Chiueh et al. [1996], Haskin [1998], and Bolosky et al. [1997], the QOS context is predetermined as the servers only service video streams. In the RT-Mach mixed media server, the QOS context is bound to the thread that is spawned to service the request stream [Molano et al. 1997] whereas in the PRISM, Fellini and CMFS mixed media architectures, the QOS context is bound to the opened instance of the file containing the data [Wijayaratne and Reddy 2001; Martin et al. 1996; Anderson et al. 1992]. In the Eclipse architecture, the QOS context is established and maintained by using the file system name space [Blanquer et al. 1999].

Real-time data access in file systems is contingent upon the real-time availability of file system metadata blocks. Some systems propose special data block placement policies to minimize the need for real-time metadata [Shenoy et al. 1998]. Some architectures propose special metadata prefetch threads [Wijayaratne and Reddy 2001] or special control paths to access file system metadata [Lougher and Shepherd 1993]. Data block striping among a set of storage devices is used to optimize data access and improve fault tolerance. Most multimedia architectures utilize the special characteristics of continuous media to determine stripe sizes and adjacency of data blocks [Shenoy et al. 1998] to facilitate efficient disk I/O scheduling.

10. CONCLUSION

This article presented a new disk scheduling algorithm, SCAN-EDF. SCAN-EDF is a hybrid scheduling policy that combines the features of real-time scheduling policies and traditional seek optimization policies. SCAN-EDF with deferred deadlines is shown to perform well in multimedia environments. SCAN-EDF employs an immediate server approach to provide reasonable performance to aperiodic

requests at the same time supporting a large number of real-time streams. Larger requests and deferred deadlines are shown to improve performance significantly. We have done comparative evaluation of these techniques to show that, for a given amount of buffer space, deferring deadlines is a better trade-off than using larger requests in a multimedia I/O system.

We also described an extended SCAN-EDF algorithm implemented in the PRISM architecture. We presented the architectural pre-requisites for implementing SCAN-EDF and discussed our implementation strategy. We established that in a mixed media environment admission controllers are necessary to keep service classes from monopolizing the disk I/O bandwidth and special consideration is needed to ensure that periodic streams do not miss deadlines. We presented the architectural support provided in PRISM to maintain the QOS context and to integrate the scheduler into the system. Our results show that PRISM improves class-specific service compared to native Linux system.

REFERENCES

ABBOTT, R. AND GARCIA-MOLINA, H. 1990. Scheduling I/O requests with deadlines: A performance evaluation. In *Proceedings of the IEEE Real-Time Systems Symposium*. IEEE Computer Society Press, Los Alamitos, Calif. 113–124.

ANDERSON, D. P., OSAWA, Y., AND GOVINDAN, R. 1992. A file system for continuous media. *ACM Trans. Comput. Syst. 10*, 4 (Nov.), 311–337.

ANDERSON, D. P., OSAWA, Y., AND GOVINDAN, R. 1991. Real-time disk storage and retrieval of digital audio/video data. Tech. rep. UCB/CSD 91/646, University of California, Berkeley, Calif..

BLANQUER, J., BRUNO, J., GABBER, E., MCSHEA, M., OZDEN, B., AND SILBERSCHATZ, A. 1999. Resource management for QOS in Eclipse/BSD. *FreeBSD Conference*. Available HTTP:`http://freebsdcon.org/1999/exhibitors/`; accessed April 2000.

BOLOSKY, W. J., FITZGERALD, R. P., AND DOUCER, J. R. 1997. Distributed schedule management in the tiger video file server. In *Proceedings of the ACM Symposium on Operating Systems Principles*. ACM, New York, 212–223.

BOSCH, P., MULLENDER, S. J., AND JANSEN, P. G. 1999. Clockwise: A mixed-media file system. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems 2*. IEEE Computer Society Press, Los Alamitos, Calif., 277–281.

BRUNO, J., BRUSTOLINI, J., GABBER, E., OZDEN, B., AND SILBERSHATZ, A. 1999. Disk scheduling with quality of service guarantees. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems 2*. IEEE Computer Society Press, Los Alamitos, Calif., 400–405.

CARD, R., DUMAS, E., AND MEVEL, F. 1998. *The Linux Kernel Book*. Wiley, New York, Chap. 3–16.

CHANG, E. AND ZAKHOR, A. 1994. Scalable video data placement on parallel disk arrays. In *Proceedings of the SPIE Symposium on Electronic Imaging Science and Technology.* 208–221.

CHANG, R. I., SHIH, W. K., AND CHANG, R. C. 2000. Deadline modification scan with maximum scannable groups for multimedia real-time disk scheduling. In *Proceedings of the Real-Time Systems Symposium 19*, 149–168.

CHEN, H. J., KRISHNAMURTHY, A., LITTLE, T. D., AND VENKATESH, D. 1995. A scalable video-on-demand service for the provision of VCR-like functions. In *Proceedings of the of IEEE Conference on Multimedia Computing and Systems*. IEEE Computer Society Press, Los Alamitos, Calif., 65–72.

CHEN, M. S., KANDLUR, D., AND YU, P. S. 1994. Support for fully interactive playout in a disk-array-based video server. In *Proceedings of the ACM Multimedia Conference*. ACM, New York, 391–398.

CHEN, S., STANKOVIC, J. A., KUROSE, J. F., AND TOWSLEY, D. 1991. Performance evaluation of two new disk scheduling algorithms for real-time systems. *J. Real-Time Syst. 3*, 307–306.

CHIUEH, T., VENKATRAMANI, C., AND VERNICK, M. 1996. Design of the stony brook video server. In *Proceedings of the SPIE First International Symposium on Technologies and Systems for Voice, Video and Data Communications 2604*, 133–145.

FREEDMAN, C. AND DEWITT, D. 1995. The SPIFFI scalable video-on-demand system. In *Proceedings of the ACM SIGMOD Conference*. ACM, New York, 352–363.

HASKIN, R. L. 1998. Tiger Shark—A scalable file system for multimedia. *IBM J. Res. Develop. 42*, 2 (Mar), 185–197.

JAYANTA, K. D., SALEHI, J. D., KUROSE, J. F., AND TOWSLEY, D. 1994. Providing vcr capabilities in large-scale video server. In *Proceedings of the ACM Multimedia Conference*. ACM, New York, 25–32.

JEFFAY, K., STANAT, D. F., AND MARTEL, C. U. 1991. On non-preemptive scheduling of periodic and sporadic tasks. In *Proceedings of the of Real-time Systems Symposium*. 129–139.

KIM, M. Y. 1986. Synchronized disk interleaving. *IEEE Trans. Comput. C-35*, *11*, 978–988.

LEHOCZKY, J. 1990. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of Real-time Systems Symposium*. 201–212.

LIN, T. H. AND TARNG, W. 1991. Scheduling periodic and aperiodic tasks in hard real-time computing systems. In *Proceedings of ACM SIGMETRICS*. ACM, New York, 31–38.

LIU, C. L. AND LAYLAND, J. W. 1973. Scheduling algorithms for multiprogramming in a hard real-time environment. *J. ACM*, 46–61.

LOUGHER, P. AND SHEPHERD, D. 1993. The design of a storage server for continuous media. *The Comput. J. 36*, 1 (Feb.), 32–42.

LUND, K. AND GOEBEL, V. 2003. Adaptive disk scheduling in a multimedia DBMS. In *Proceedings of the ACM Multimedia Conference*. ACM, New York, 65–74.

MAKAROFF, D., NEUFELD, G., AND HUTCHINSON, N. 1997. An evaluation of vbr disk admission algorithms for continuous media file servers. In *Proceedings of the ACM Multimedia Conference*. ACM, New York, 143–154.

MARTIN, C., NARAYAN, P. S., OZDEN, B., RASTOGI, R., AND SILBERSCHATZ, A. 1996. *The Fellini Multimedia Storage System*. Kluwer Academic Publications, Chap. 5.

MOKBEL, M. F., AREF, W. G., ELBASSIONI, K., AND KAMEL, I. 2004. Scalable multimedia disk scheduling. In *Proceedings of the International Conference of Data Engineering*. 498–509.

MOLANO, A., JUVVA, K., AND RAJKUMAR, R. 1997. Real-time file systems: Guaranteeing timing constraints for disk accesses in rt_mach. In *Proceedings of the IEEE Real-Time Systems Symposium*. IEEE Computer Society Press, Los Alamitos, Calif., 155–165.

PATTERSON, D. A., GIBSON, G., AND KATZ, R. H. 1988. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the ACM SIGMOD Conference*. ACM, New York, 109–116.

PLAGEMANN, T., GOEBEL, V., HALVORSEN, P., AND ANSHUS, O. 2000. Operating system support for multimedia systems. *The Comput. Commun. J. 23*, 3 (Feb.), 267–289.

REDDY, A. L. N. 1992. A study of I/O system organizations. In *Proceedings of the International Symposium on Computer Architecture*. 308–317.

REDDY, A. L. N. AND BANERJEE, P. 1989. An evaluation of multiple-disk I/O systems. *IEEE Trans. Comput. C-38,* 12 (Dec.), 1680–1690.

REDDY, A. L. N. AND HASKIN, R. 1996. *The Communications Handbook*. CRC Press, Chapt. 106.

REDDY, A. L. N. AND WYLLIE, J. 1993. Disk scheduling in a multimedia I/O system. In *Proceedings of the ACM Multimedia Conference*. ACM, New York, 225–233.

ROMPOGIANNAKIS, T., NERJES, G., MUTH, P., PATERAKIS, M., TRIANTAFILLOU, P., AND WEIKUM, G. 1998. Disk scheduling for mixed-media workloads in a multimedia server. In *Proceedings of the ACM Multimedia Conference*. ACM, New York, 297–302.

ROSE, O. 1995. Statistical Properties of MPEG video traffic and their impact on traffic modeling in ATM systems. Tech. Rep. 101, Institute of Computer Science, University of Wurzburg.

SALEM, K. AND GARCIA-MOLINA, H. 1986. Disk striping. In *Proceedings of the International Conference on Data Engineering*. 336–342.

SHENOY, P., GOYAL, P., AND VIN, H. M. 1999. Architectural considerations for next generation file systems. In *Proceedings of the ACM Multimedia Conference*. ACM, New York, 457–467.

SHENOY, P. J., GOYAL, P., RAO, S., AND VIN, H. M. 1998. Symphony: An integrated multimedia file system. In *Proceedings of the ACM/SPIE Multimedia Computing and Networking*. ACM, New York, 124–138.

SHENOY, P. J. AND VIN, H. M. 1998. Cello: A disk scheduling framework for next generation operating systems. *Proc. ACM SIGMETRICS 26*, 1 (June), 44–55.

SHIH, W. K., LIU, J. W., AND LIU, C. L. 1992. Modified rate monotone algorithm for scheduling periodic jobs with deferred deadlines. Tech. Rep. University of Illinois, Urbana-Champaign.

WIJAYARATNE, K. B. R. 2001. PRISM: A file server architecture for providing integrated services. Ph.D. dissertation, Department of Computer Science, Texas A&M University, Texas.

WIJAYARATNE, R. AND REDDY, A. L. N. 1999a. Integrated QOS management for disk I/O. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems 1*. IEEE Computer Society Press, Los Alamitos, Calif., 487–492.

WIJAYARATNE, R. AND REDDY, A. L. N. 1999b. Techniques for improving the throughput of VBR streams. *ACM/SPIE Multimed. Comput. Netw. 3654*, 216–227.

WIJAYARATNE, R. AND REDDY, A. L. N. 2000. Providing QOS guarantees for disk I/O. *Multimed. Syst 8*, 1 (January), 57–68.

WIJAYARATNE, R. AND REDDY, A. L. N. 2001. System support for providing integrated services from networked multimedia storage servers. In *Proceedings of the ACM Multimedia Conference*. ACM, New York, 270–279.

YEE, J. AND VARAIYA, P. 1992. Disk scheduling policies for real-time multimedia applications. Tech. rep. University of California, Berkeley, Berkeley, Calif..

YU, P. S., CHEN, M. S., AND KANDLUR, D. D. 1993. Grouped sweeping scheduling for dasd-based multimedia storage management. *IEEE Multimed. Syst. 1*, 99–109.

Received November 2004; accepted December 2004