I/O issues in a multimedia system

A. L. Narasimha Reddy

Jim Wyllie

IBM Almaden Research Center

650 Harry Road

San Jose, CA 95120.

reddy,wyllie@almaden.ibm.com

1 Abstract

In this paper, we look at the various I/O issues in a multimedia system. In a multimedia server, the disk requests may have constant data rate requirements and need guaranteed service. We study the impact of the real-time nature of the I/O requests on the various components of the I/O system. We study the impact of disk scheduling algorithms on the performance of a multimedia system. We 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 show how deadline extension helps in utilizing multiple disks on a single SCSI bus.

2 Introduction

Current computer systems put emphasis on processor performance with little attention paid to the I/O system. There have been several recent studies at improving the I/O system performance [1, 2, 3, 4]. These studies concentrate on improving the I/O throughput. 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 paper, we explore some of the I/O issues in designing such a multimedia system.

Playback of video and audio streams, teleconferencing are some of the examples of real-time applications. Each application places different demands on the I/O system. For example, teleconferencing requires very small latencies of delivery of an I/O stream and playback applications require guaranteed I/O throughput. In this paper, we will primarily consider the playback applications. We will specify the real-time requests by specifying the required data rate in kbytes/sec. The time at which a periodic request is started is called the **release time** of that request. The time at which the request is to be completed is called the **deadline** for that request. Requests that do not have real-time requirements are termed **aperiodic** requests.

In a multimedia server, some of the different service phases of a real-time request are disk scheduling, SCSI

bus scheduling, and processor scheduling. Additional service may be needed if the request has to be satisfied across a local area network (LAN). We will restrict ourselves to the support that needs to be provided at the server with special emphasis on the two service phases, disk scheduling and SCSI bus contention.

In traditional real-time systems, when requests have to be satisfied within deadlines, algorithms such as earliest deadline first (EDF), and least slack time first are used. EDF assumes that tasks are preemptable and that their service times are known in advance. However, the disk service time for a request is a random variable that depends on the relative position of the request from the current position of the read-write head. Current disks are not preemptable. Moreover, due to the overheads of seek time, strict real-time scheduling of 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 the arm movement in serving requests. These techniques reduce the 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 the 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. There is a need for devising an appropriate disk scheduling policy for serving real-time requests.

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 tradeoff 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 the 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. In Section 3, we will describe how buffers can be used to improve the throughput of the I/O system.

Continuous-media I/O systems may have to serve aperiodic requests also. In such environments, it is necessary to ensure that periodic requests do not miss their deadlines while giving reasonable response times for aperiodic requests. A similar problem is studied in [5]. Anderson *et al* [6] present a comprehensive treatment of the disk scheduling problem. A work-ahead scheduling algorithm based on least slack time first is presented. A periodic fill policy is proposed in [7] for scheduling multi-media requests at the disk. A file system for handling audio/video data is presented in [8]. IBM's Ultimedia server, Starlight's video server are some of the commercial efforts in supporting audio and video data.

2 Disk Scheduling Algorithms

The scheduling algorithm should be fair, i.e., should avoid starvation of service to requests. Hence, we will not consider shortest seek time first (SSTF) kind of algorithms in our discussion. The scheduling policy has to meet the deadlines of real-time requests and at the same time provide low response time to aperiodic requests. Aperiodic I/O requests are known to be bursty. 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. This model of service, used for this study, is shown in Fig. 1. In this model, if the aperiodic requests are generated faster than they are being served, they are queued in a separate queue. In the following sections, a number of scheduling policies and their application in a multi-media system are described.

2.1 EDF

Earliest deadline first (EDF) algorithm is shown to be optimal [9] 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 is less than 1. Current disks are however not preemptable. Due to the overheads of seek time, strict real-time scheduling of disk arm may result in excessive seek time cost and poor utilization of the disk. In this scheduling policy, each real-time request is given a deadline. In a later section, we explain how deadlines are set for real-time requests. Requests are served strictly by their order of deadlines.

Aperiodic requests are served using the immediate server approach [5] where the aperiodic requests are given higher priority over the real-time requests i.e., a waiting aperiodic request will be served immediately after serving the current real-time request. The service schedule allows a certain number of aperiodic requests during each round of service and when sufficient number of aperiodic requests are not present, the real-time requests make use of the remaining service time during that period. This policy of serving aperiodic requests is employed so as to provide reasonable response times for aperiodic requests while guaranteeing the deadlines for real-time requests. This is in contrast to earlier approaches where the emphasis has been only on providing real-time performance guarantees.

2.2 CSCAN

CSCAN is one of the members of the SCAN type of disk scheduling algorithms. In this algorithm, the disk read-write head scans for requests in one direction, either from outermost to the innermost track or vice versa. Requests are served in the order of the read-write head scan direction and when the inner most request is served, the read-write head is moved to the outermost pending request to be served. This policy does seek optimization while guaranteeing that no request gets starved of service. Since this policy has no notion of time or deadlines, real-time requests are strictly served in the order of their location on the disk surface. Aperiodic requests are also served in scan order. As a result, aperiodic requests may have to wait behind a number of real-time requests, resulting in large response times.

2.3 SCAN-EDF

SCAN-EDF is a hybrid scheduling algorithm proposed in [10]. 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. This strategy combines the benefits of both real-time and seek-optimizing scheduling algorithms.

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. The following techniques make it possible for various requests to have the same deadlines. SCAN-EDF prescribes that the requests have release times that are multiples of the period p. This results in all requests having deadlines that are multiples of the period p. This enables the requests to be grouped in batches and served accordingly. When the requests have different data rate requirements, SCAN-EDF can be combined with a periodic fill policy [7] 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 tasks 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**.

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 the 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, and 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.

Aperiodic requests are served using the immediate server approach as described under the the EDF scheduling policy.

3 Buffer space tradeoff

Real-time requests typically need some kind of response before the next request is issued. Hence, the deadline for a request is set equal to the release time plus the period of the request. 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 tradeoff is significant since the efficiency of the disk service is a varying function of the request size. The disk arm is more efficiently used when the request sizes are large and hence it may be possible to support larger number of multimedia streams at a single disk. This flexibility in changing the deadlines gives a powerful handle on controlling the system performance.

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 $2nS$. There is another tradeoff that is possible. The deadline of a request need not be set equal to release time plus the period of the request. For example, we can defer the deadline of a request by a period and make the deadline of the request equal to release time + $2p$. In this case, the consumer of real-time data works $2p$ time behind the producer of the real-time data. This gives more time for the disk arm to serve a given request and may allow more seek optimizations than that is possible when the deadlines are equal to release time + 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 (it is possible that when the consumer is consuming buffer 1, both buffers 2 and 3 are waiting for service at the disk, buffer 2 request having an earlier deadline). Hence, this raises the buffer requirements to $3S$ for each request stream. In general, when the requests are allowed to set deadlines equal to $r + mp$, r being the request release time, 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 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 frequently to the request queue at the disk. This results in more efficient use of disk arm and as a result, larger number of request streams can be supported at a single disk. A similar technique called work-ahead is utilized by Anderson *et al* [6].

In [11], 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. In [12], it is shown that if the periods of all the requests are extended by the largest period, a modified rate monotone scheduling algorithm is optimal. Both these studies assume that tasks are preemptable.

Both the above techniques, larger requests with larger periods and delayed deadlines, increase the latency of service for a real-time stream. When the deadlines are delayed, the consumption of multimedia data stream

can only be started after (release time + 2p) as opposed to (release time + p) when deadlines are not delayed. When larger requests are employed, 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 for a longer time behind the current real-time request that is being served. The improved efficiency of these techniques needs to be weighed against the higher buffer requirements and the higher latency for starting a stream.

4 Performance Evaluation

4.1 System model

IBM's Allicat disk is used as a model, with the parameters shown in Table 1. 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 CDROM data stream. Aperiodic requests have poisson arrivals. The mean time between arrivals is varied from 25 ms to 200 ms. Each aperiodic request is assumed to ask for a track of data. The request size for the real-time requests is varied among 1, 2, 5, or 15 tracks. 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.

Two systems, one with deadlines equal to release time + p and the second with deadlines equal to release time + $2p$ are modeled. 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 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 $n + 1$ where the deadlines

Table 1. 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

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.

4.2 Maximum number of streams

Fig. 2 shows the maximum number of real-time streams supported by each scheduling policy when the average aperiodic request arrival period is 200 ms. The dotted lines correspond to a system with extended deadlines and the dashed lines are for the system where deadlines are not delayed.

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 is the reason for poor performance of this policy. Fig. 2 clearly shows the advantage of utilizing seek optimization techniques.

At larger request sizes, the different scheduling policies have 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 seek optimization techniques do not result in significant improvements in performance. Hence, the performance does not differ significantly across the scheduling policies at larger request sizes. At a request size of 5 tracks, i.e., at roughly 200 kbytes/buffer, minimum of 2 buffers/stream corresponds to 400 kbytes of buffer space per 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 such amount of buffer space is not available, smaller request sizes need to be considered.

At smaller request sizes, deferring the deadlines has a better impact on performance than increasing the request size. For example, at a request size of 1 track and deferred deadlines (with buffer requirements of 3 tracks per stream) 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.

The results presented in Fig. 2 may be counter-intuitive. CSCAN which is a non-realtime scheduling policy supports most streams and EDF, a strict real-time scheduling policy supports the least number of streams. Reasons for this behavior are: (1) efficient utilization of disk arm in CSCAN results in reduced service times and reduced variability in serving a bunch of real-time requests in a single scan. This resulting reduction in variability of service makes meeting deadlines more predictable and (2) giving higher priority to aperiodic requests in EDF results in reducing the number of real-time requests supported by EDF.

4.3 Aperiodic response time

Fig. 3 shows the response time for aperiodic requests. The figure shows the aperiodic response time at request

sizes of 1, 2, 5 and 15 tracks. Since different scheduling policies support different number of maximum number of real-time streams, to make a fair comparison, equal number of 8, 12, 15, 18 real-time streams are utilized at the four request sizes of 1, 2, 5 and 15 tracks respectively. CSCAN has the worst performance and SCAN-EDF has the best aperiodic request performance. With CSCAN, on an average, an aperiodic request has to wait for half a sweep for service. As a result, an aperiodic request may have to wait behind a large number of real-time requests. In SCAN-EDF and EDF, aperiodic requests are given higher priorities by giving them shorter deadlines (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 current request that is being served. Among these three policies, the slightly better performance of SCAN-EDF is due to the lower arm utilizations.

From figures 2 and 3, 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 aperiodic requests. EDF, does not perform very well in supporting real-time requests due to inefficient utilization of disk arm but offers good response times for aperiodic requests (by giving them higher priority). 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. STAGEDF, a variation of EDF where the release times of requests are staggered and PCSCAN, a variation of CSCAN where aperiodic requests are given priority were also considered. It was observed that these two schemes do not perform as well as SCAN-EDF [10].

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 (for supporting the same number of real-time streams) and hence observe larger

response times. Impact of other factors such as aperiodic request arrival, multiple data stream rates and the use of disk arrays have been considered [10]. Similar trends in performance were observed.

4.4 Effect of SCSI bus contention

In today's systems, disks are connected to the rest of the system through a peripheral device bus such as SCSI or IPI. To amortize the costs of SCSI controllers, multiple disks may be connected to the system on a single bus. SCSI bus, for example can support 10 MB/sec (also 20 MB/sec with wider buses). Since most disks have raw data rates in the range of 3-5 MB/sec, two to three disks can be attached to a single SCSI bus without affecting the total throughput of the disks. However, even when the raw data rate of the SCSI bus may be fast enough to support two to three disks, in a real-time environment, this shared bus could add delays to individual transfers and may result in missed deadlines. To study the affect of the SCSI bus contention on the throughput of the real-time streams in a system, we simulated 3 Allicat disks attached to a single SCSI bus. The raw data rate of these disks is 3.8 MB/sec. This implies that the total throughput of these disks slightly exceeds the rated bandwidth of the SCSI bus at 10 MB/sec. However, due to seek and latency penalties paid for each access, the disks do not sustain the 3.8 MB/sec for long periods of time.

SCSI bus is a priority arbitrated bus. If more than one disk tries to transfer data on the bus, the disk with a higher priority always gets the bus. Hence, it is possible that real-time streams being supported by the lower priority disks may get starved if the disk with higher priority continues to transmit data. If the arbitration is driven by other policies such as round-robin or round-robin with a time slice, better performance may be obtained. For multimedia applications, other channels such as the proposed SSA by IBM, which operates as a time division multiplexed channel, are more suitable and would be less complex to guarantee deadlines.

Fig. 4 shows the impact of SCSI bus contention on the number of streams that can be supported. The number of streams supported is less than three times that of the individual disk real-time request capacity.

This is mainly due to the contention on the bus. At a request size of 5 tracks, the ratio of the number of streams supported in a three disk configuration to that of a single disk configuration varies from 2.1 in the system with extended deadlines to 1.8 in the system without extended deadlines. This again shows that deadline extension increases the chances of meeting deadlines, in this case smoothing over the bus contention delays. Figure 4 assumes that the number of streams on the three disks differ at most by one. If the higher priority disk is allowed to support more real-time streams, the total throughput of real-time streams out of the three disks would be lower. We observed a sharp reduction in the number of streams supported at the second and third disks when the number of streams supported at the first disk is increased even by one. For example, at a request size of 5 tracks and extended deadlines, SCAN-EDF supported 15, 14 and 14 streams at the three disks but only supported 7 streams at the second and the third disks when the number is raised to 16 at the first disk.

The optimal request size is mainly related to the relative transfer speeds of the SCSI bus and the raw disk. When a larger block size is used, disk transfers are more efficient, but disks with lower priority see larger delays and hence are more likely to miss deadlines. When a shorter block is used, disk transfers are less efficient, but the latency to get access to SCSI bus is shorter. This tradeoff determines the optimal block size.

Most of the modern disks have a small buffer on the disk arm for storing the data currently being read by the disk. Normally, the data is filled into this buffer by the disk arm at the media transfer rate (in our case, at 3.8 MB/sec) and transferred out of this buffer at the SCSI bus rate (in our case, at 10 MB/sec). If this arm buffer is not present, the effective data rate of SCSI bus will be reduced to the media transfer rate or lower. When the disk arm buffers are present, SCSI transfers can be initiated by the individual disks in an intelligent fashion such that the SCSI data rate can be maintained high while providing that the individual transfers are completed across the SCSI bus as they are being completed at the disk surface. IBM's Allicat

drive utilizes this policy for transferring in and out of its 512 kbyte arm buffer and this is what is modeled in our simulations. Without this arm buffer, when multiple disks are configured on a single SCSI bus, the real-time performance will be significantly lower.

5 Admission control

To guarantee the required data rate for a given I/O stream, some sort of admission control needs to be applied. If the requests are admitted without any limit, the load on the system may be so large that none of the streams will be able to meet their deadlines. Traditionally, real-time systems consider worst-case scenarios into account to obtain lower bounds on achievable utilizations without missing deadlines.

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 to 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. Deadline extensions are utilized to further reduce the uncertainties of meeting the deadlines.

An analysis has been carried out in [10] for guaranteeing the deadlines in SCAN-EDF policy. However, since a multimedia stream requires so many pieces of service, disk scheduling, SCSI bus scheduling, processor scheduling etc., this kind of analysis could quickly become cumbersome when applied to all the different pieces of service. One simple approach could be to estimate the achievable utilizations through either measurement or analysis and underutilize the system to decrease the probability of deadline misses to an acceptable level.

6 Impact on Computer System Design

We observed in this paper that buffering is a very effective tool for minimizing the variations in service to be

provided. Typically, current systems utilize part of the main memory as an I/O cache. Individual request streams do not have a good control over the cache space and policies such as LRU are unlikely to be effective for managing this space in multimedia systems. Moreover, multimedia streams increase the demands on the buffer space provided in the system. Policies for better managing this space among multimedia streams and other non-realtime requests need to be devised.

Large block sizes were also found to be an effective way for ensuring real-time service. This may imply that the file system needs to allocate data in larger blocks, in the range of 64 kbytes or larger. Most current systems use block sizes of the order of 4 kbytes, which is far too small for the real-time applications. However, when space is allocated in large blocks on the disk, internal fragmentation may result in large fraction of the disk space being wasted. Hence, the future file systems may have to be designed to support multiple block sizes. We observed that new disk scheduling algorithms that combine the features of traditional seek optimization techniques with traditional real-time scheduling policies may be required. We also observed that priority driven peripheral device buses such as SCSI may not be ideally suited for real-time applications. In delivering the multimedia I/O streams to the workstation, several more aspects of the system need to be investigated. For example, scheduling the various processes on the processor impacts the real-time delivery guarantees. We also need to look at the problems of delivering real-time I/O over the local area network and real-time scheduling of the processor.

7 Conclusions

We looked at several I/O issues for supporting multimedia streams. This paper focused on disk scheduling, SCSI bus contention and effect of buffer space on the performance of the real-time requests and aperiodic requests. A new scheduling policy that combines real-time scheduling policies and disk arm scheduling policies is quite affective in serving both real-time and aperiodic requests. Buffer space was shown to have a significant impact on the performance of the system. It was shown that delaying deadlines is a more effective way of utilizing the buffer space than using larger request sizes. It was shown that SCSI bus contention can

considerably reduce the achievable performance from the individual disks. Implications of these results on computer system design have been briefly discussed.

Acknowledgments

Discussions with Barbara Simons, Robert Morris, and Roger Haskin, have helped in clarifying the presentation.

References

[1] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). *ACM SIGMOD Conference*, June 1988.

[2] M. Y. Kim. Synchronized disk interleaving. *IEEE Trans. Comput.*, C-35, no. 11:978–988, Nov. 1986.

[3] K. Salem and H. Garcia-Molina. Disk striping. *Int. Conf. on Data Engineering*, pages 336–342, 1986.

[4] A. L. Narasimha Reddy and P. Banerjee. An evaluation of multiple-disk I/O systems. *IEEE Trans. Comput.*, C-38, no. 12:1680–1690, Dec. 1989.

[5] T. H. Lin and W. Tarng. Scheduling periodic and aperiodic tasks in hard real-time computing systems. *Proc. of SIGMETRICS*, pages 31–38, May 1991.

[6] D. P. Anderson, Y. Osawa, and R. Govindan. Real-time disk storage and retrieval of digital audio/video data. *Tech. report UCB/CSD 91/646, Univ. of Cal., Berkeley*, Aug. 1991.

[7] J. Yee and P. Varaiya. Disk scheduling policies for real-time multimedia applications. *Tech. report, Univ. of California, Bekeley*, Aug. 1992.

[8] H. M. Vin and P. V. Rangan. Designing file systems for digital video and audio. *Proc. of 13th ACM Symp. on Oper. Sys. Principles*, 1991.

[9] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of ACM*, pages 46–61, 1973.

[10] A. L. Narasimha Reddy and Jim Wyllie. Disk scheduling in a multimedia I/O system. *Proc. of ACM Multimedia Conf.*, Aug. 1992.

[11] J.P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. *Proc. of Real-time Systems Symp.*, pages 201–212, Dec. 1990.

[12] W. K. Shih, J. W. Liu, and C. L. Liu. Modified rate monotone algorithm for scheduling periodic jobs with deferred deadlines. *Tech. Report, Univ. of Illinois, Urbana-Champaign*, Sept. 1992.

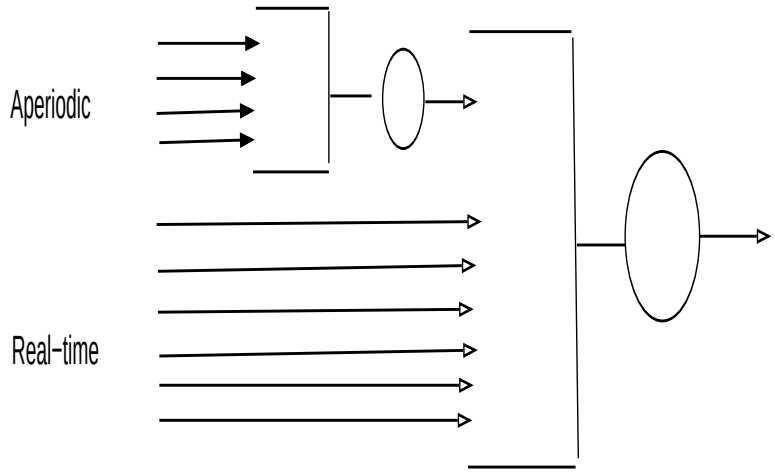

Fig. 1. Service model for serving real-time and aperiodic requests.

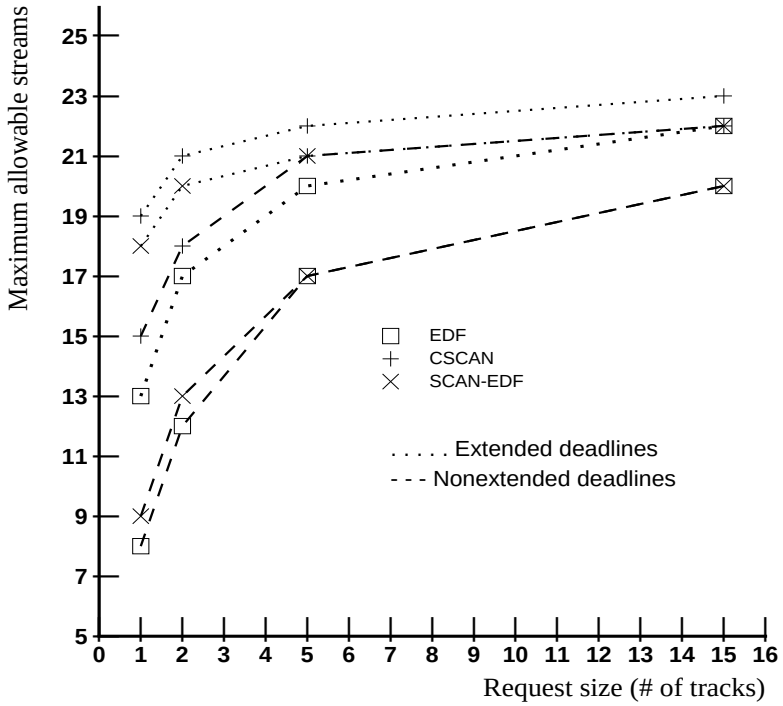

Fig. 2. Performance of different scheduling policies.

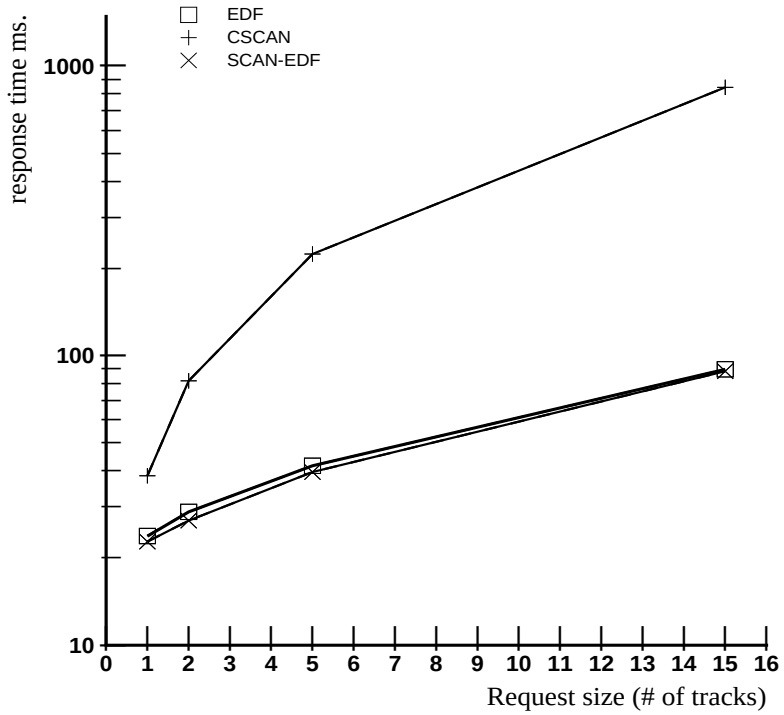

Fig. 3. Aperiodic response time with different scheduling policies.

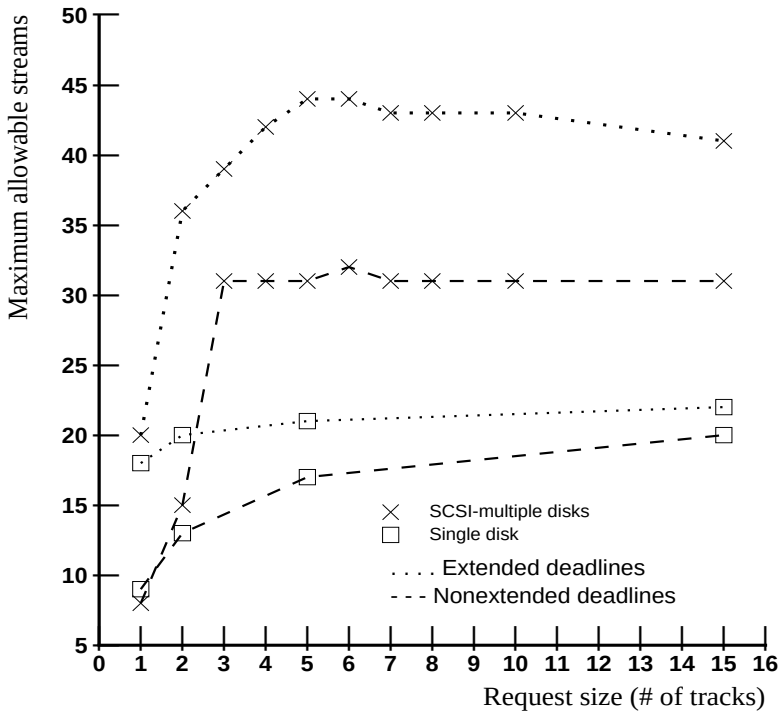

Fig. 4. Performance of SCAN-EDF policy with SCSI bus contention.