I/O Issues in a Multimedia System

A.L. Narasimha Reddy and James C. Wyllie
IBM Almaden Research Center

With disk-scheduling deadlines set longer than request periods, a multimedia server can support more data streams and efficiently use multiple disks on a SCSI bus.

Current emphasis in computer-system design is on processor performance rather than I/O systems. However, future I/O systems will have to support continuous media such as video and audio, whose system demands are different from those of data such as text. Previous studies have looked at how to improve I/O system throughput,[1-4] but multimedia computing requires us to focus on designing I/O systems that can handle real-time demands.

Video- and audio-stream playback and teleconferencing are real-time applications with different I/O demands. Teleconferencing requires very small latencies of delivery, but playback applications require guaranteed real-time I/O throughput. In this article, we primarily consider playback applications.

In a multimedia server, different service phases of a real-time request are disk, Small Computer Systems Interface (SCSI) bus, and processor scheduling. Additional service might be needed if the request must be satisfied across a local area network. We restrict ourselves to the support provided at the server, with special emphasis on two service phases: disk scheduling and SCSI bus contention.

When requests have to be satisfied within deadlines, traditional real-time systems use scheduling algorithms such as earliest deadline first (EDF) and least slack time first. However, EDF makes the assumption that disks are preemptable, and the seek-time overheads of its strict real-time scheduling result in poor disk utilization.

Traditional seek optimization techniques such as Scan or shortest seek time first improve disk-arm utilization by serving requests close to the disk arm. The request queue is ordered by the requests' relative positions on the disk surface. However, such techniques might not be suitable for real-time environments because they do not account for time or deadlines in scheduling decisions. (The time at which a request is started is its *release time*, and the time at which it must be completed is its *deadline*.)

We can provide the constant data rate necessary for real-time requests in various ways that require trade-offs. Here we analyze how trade-offs that involve buffer space affect the performance of scheduling policies. We show that deferred deadlines, which increase buffer requirements, improve system performance significantly.

0018-9162/94/ $4.00 © 1994 IEEE

When continuous-media I/O systems also have to serve *aperiodic requests* (requests that do not have real-time requirements), they must ensure that periodic requests do not miss their deadlines *and* ensure reasonable response times for the aperiodic requests.[5] Solutions other than the one we present here include a work-ahead scheduling algorithm based on least slack time first,[6] a periodic fill policy for scheduling multimedia requests at the disk,[7] and a file system for handling audio/video data.[8] IBM's Ultimedia server and Starlight's video server are commercial products supporting audio and video data.

Disk-scheduling algorithms

Because a scheduling algorithm should be fair and avoid starvation of service to requests, we do not consider shortest seek time first algorithms. While meeting the deadlines of real-time requests, the scheduling policy must provide low response time to aperiodic requests, which are known to be bursty. If we give them unlimited service, a burst of aperiodic requests can disturb the service of real-time requests considerably. To limit the number of aperiodic requests served in a given period of time, we can maintain a separate queue that releases these requests at a bounded rate. If aperiodic requests are generated faster than they are served, they are queued separately and scheduled in a way that does not interfere with the real-time handling of periodic requests.

EDF. The earliest deadline first (EDF) algorithm[9] is optimal if requests' service times are known in advance. However, the disk-service time for a request depends on its position relative to the read-write head's current position. The original EDF algorithm assumed that tasks were preemptable with zero preemption cost and showed that EDF could schedule tasks if and only if the task utilization were less than 1. However, current disks are not preemptable. EDF gives each real-time request a deadline (we explain how deadlines are set in a later section) and serves requests strictly in order. Strict real-time scheduling of the disk arm might result in excessive seek-time cost and poor disk utilization.

While EDF schedules real-time requests, the immediate server approach[5] serves aperiodic requests, giving them priority for service immediately after the current real-time request. This schedule allows a certain number of aperiodic requests during each round of service. When there are only a few aperiodic requests, the real-time requests use the remaining service time during that round. This policy, which provides reasonable response times for aperiodic requests while guaranteeing deadlines for real-time requests, contrasts with earlier approaches that guaranteed timely service only for real-time requests.

CScan. CScan (for circular Scan) is a Scan type of disk-scheduling algorithm.

When there are only a few aperiodic EDF requests, the real-time requests use the remaining service time during that round.

The disk read-write head scans for requests in one direction, from the outermost to the innermost track or vice versa, serving requests in the order of its scan direction. If it is scanning inward, after it serves the innermost request the head moves to the outermost pending request. This policy does *seek optimization* while guaranteeing that no request gets starved of service. Since it has no notion of time or deadlines, it serves real-time requests strictly in the order of their location on the disk surface. It also serves aperiodic requests in scan order, and they might have to wait behind a number of real-time requests.

Scan-EDF. Scan-EDF[10] is a hybrid scheduling algorithm that provides both seek optimization and earliest deadline first service. Requests are normally served in EDF order. If several requests have the same deadline, they are served according to their track locations on the disk.

Because Scan-EDF applies seek optimization only to requests having the same deadline, its efficiency depends on how often seek optimization can be applied. We can improve its efficiency with techniques that give various requests the same deadlines. Scan-EDF prescribes that requests have release times that are multiples of the period p. Hence, requests are 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 give all the requests the same deadline. Requests are served in a cycle, and each request gets service time proportional to its required data rate. The length of the cycle is the sum of the service times of all requests. All requests in the current cycle are given a deadline at the end of the current cycle.

Here is a more precise description of the 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**.

We can also implement Scan-EDF with a slight modification to EDF. Let D_i be the deadlines of the tasks and N_i be their track positions. Then we modify the deadlines 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 so $D_i + f(N_i) > D_j + f(N_j)$, 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 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, suppose tasks A, B, C, and D have deadlines 500, 500, 500, and 600, and ask for data from tracks 347, 113, 851, and 256, respectively. If N_{max} = 1,000, the modified deadlines of A, B, C, and D become 499.347, 499.113, 499.851, and 599.256 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 described with the EDF scheduling policy.

Buffer space trade-off

Real-time requests typically need some kind of response before the requester issues the next request. Hence, we set the deadline for a request equal to the release time plus the period of the request. The multimedia I/O system must provide a constant data rate for each request stream. Hence, when the available buffer space is small, the request stream asks for small pieces of data in each period. When the available buffer space is large, it asks for larger pieces with correspondingly larger periods between requests. This trade-off is significant since disk-service efficiency is a varying function of the request size.

Each request stream requires a buffer for the consuming process and another 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 $2nS$. Here another trade-off 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 so it is equal to release time plus $2p$. Then the consumer of real-time data works $2p$ time behind the producer. This gives the disk arm more time to serve a request and might allow more seek optimizations than are possible when the deadlines are equal to release time plus p. In the resulting scenario, the consuming process consumes buffer 1, the producing process (disk) reads data into buffer 3, and buffer 2 is filled earlier by the producer and waits for consumption. (It is possible that when the consumer is consuming buffer 1, both buffers 2 and 3 are waiting for service at the disk, with the buffer 2 request having an earlier deadline.)

This arrangement raises the buffer requirements to $3S$ for each request stream. In general, when the requests set deadlines equal to $r + mp$ (r is 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. The evaluation we present in the next section shows that this strategy has significant benefits. The extra time available for serving a given request lets the system apply seek-optimization techniques more frequently to the request queue at the disk. The disk arm works more efficiently, and a single disk can support a larger number of request streams. (The work-ahead technique[6] is similar to this approach.)

When all deadlines are extended by a multiple of the periods, rate monotonic scheduling could achieve better resource utilization.[11] Moreover, if the periods of all the requests are extended by the largest period, a modified-rate monotone scheduling algorithm is optimal.[12] However, both these strategies assume that tasks are preemptable with zero cost. On a disk, the cost of such preemption is not negligible; a seek and a rotational latency are required.

Larger requests add to the response time for aperiodic requests, waiting longer for the current real-time request being served.

When we use larger requests with larger periods or delayed deadlines, we increase the latency of service for a real-time stream. When the deadlines are delayed, the consumption of a multimedia data stream can be started only after release time + $2p$, as opposed to release time + p when deadlines are not delayed. With larger requests, buffers fill more slowly and more time elapses before the multimedia stream can start. Larger requests also increase the response time for aperiodic requests: They must wait longer behind the current real-time request being served. We must weigh the improved efficiency of these techniques against the higher buffer requirements and the higher latency for starting a stream.

Table 1. Disk parameters.

Parameter	Value
Time for one rotation	11.1 ms
Average seek	9.4 ms
Sectors per track	84
Sector size	512 bytes
Tracks per cylinder	15
Cylinders per disk	2,577
Seek cost function	Nonlinear
Minimum seek time s_0	1.0 ms

Performance evaluation

System model. For our evaluation, we used IBM's 3.5-inch 2-Gbyte Allicat disk with the parameters listed in Table 1. We assumed each real-time request stream required a constant data rate of 150 Kbytes per second, which roughly corresponds to the requirements for a CD-ROM data stream. Aperiodic requests had Poisson arrivals. The mean time between arrivals varied from 25 to 200 milliseconds. We assumed each aperiodic request asked for a track of data. The request size for the real-time requests varied among 1, 2, 5, or 15 tracks. The period between two requests of a real-time request stream varied depending on the request size to support a constant data rate of 150 KBps. We assumed the requests to be uniformly distributed over the disk surface.

We modeled two systems, one with deadlines equal to release time + p and the second with release time + $2p$. Our primary performance measure was the number of real-time streams supported by each scheduling policy. We also looked at the response time for aperiodic requests, which we did not want to be unduly large. A good policy offers 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. We obtained the maximum number of supportable streams n by increasing the number of streams incrementally until $n + 1$, where the deadlines could not be met. For each point in the three accompanying figures, we conducted 20 experiments with different seeds for random number generation. We graphed

the minimum of these values as the maximum number of streams that could be supported.

Maximum number of streams. Figure 1 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 to a system in which deadlines were not delayed.

Figure 1 shows that deferring deadlines improves the number of supportable streams significantly for all the scheduling policies. The performance improvement ranges from four streams for CScan to nine streams for Scan-EDF at a request size of one track. When deadlines are deferred, CScan has the best performance, Scan-EDF is very close to CScan, and EDF has the worst performance. EDF scheduling results in random disk-arm movement and hence poorer performance than that attained by policies using seek-optimization techniques.

With larger request sizes, we see less difference in the performance of the scheduling policies. Transfer time becomes more important: When seek-time overhead is a smaller fraction of service time, the seek-optimization techniques do not significantly improve performance. At a request size of five tracks — that is, at roughly 200 Kbytes per buffer — a minimum of two buffers per stream corresponds to 400 Kbytes of buffer space per stream. To support 20 streams, we need 400 Kbytes times 20, or 8 Mbytes, of buffer space at the I/O system. Deferring deadlines brings the requirement to 12 Mbytes. When such space is not available, we must use smaller request sizes.

At smaller request sizes, deferring the deadlines has a better impact on performance than increasing the request size. For example, with a request size of one track and deferred deadlines (with buffer requirements of three tracks per stream), EDF supports 13 streams. When deadlines are not deferred, with a larger request size of two tracks and buffer requirements of four tracks, EDF supports only 12 streams. Figure 1 shows a similar trend with other policies as well — for example, when we compare request sizes of two and five tracks.

The results in Figure 1 might be counterintuitive. CScan, a non-real-time scheduling policy, supports the most streams, and EDF, a strict real-time scheduling policy, supports the least. CScan's efficient disk-arm utilization reduces service times and variability in serving many real-time requests in a single scan. The reduction in service variability lets CScan meet deadlines more predictably. EDF also gives higher priority to aperiodic requests, so it cannot support as many real-time requests.

Aperiodic response time. Figure 2 shows aperiodic response times. Different scheduling policies support different maximum numbers of real-time streams. For a fair comparison, we used 8, 12, 15, and 18 real-time streams at request sizes of 1, 2, 5, and 15 tracks respectively, for all scheduling policies. CScan had the worst aperiodic request performance, and Scan-EDF had the best. With CScan, an aperiodic request must wait an average of half a sweep for service. As a result, it might have to wait behind many real-time requests. Scan-EDF and EDF give aperiodic requests higher priorities by using shorter deadlines (100 ms from the issuing time). As a result, aperiodic requests typically wait behind only the current real-time request being served. Scan-EDF's slightly better performance is due to better arm utilization.

Figures 1 and 2 show that CScan supports real-time requests well but not aperiodic requests. EDF does not support real-time requests well but provides good response times for aperiodic requests. Scan-EDF performs well for both measures. It supports almost as many real-time streams as CScan and at the same time offers the best response times for aperiodic requests. (We also considered StagEDF, a variation of EDF with staggered request release times, and PCScan, a variation of CScan that gives priority to aperiodic requests. Neither per-

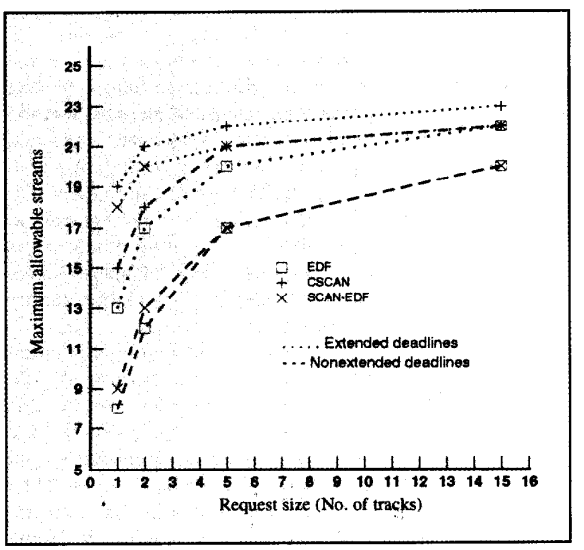


Figure 1. Performance of different scheduling policies.

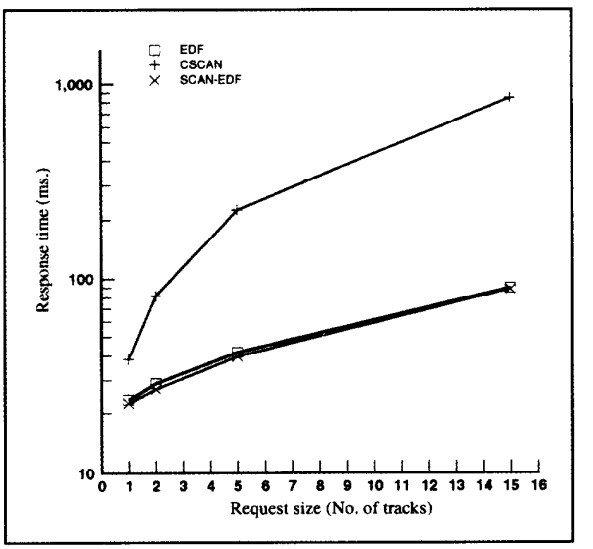


Figure 2. Aperiodic response time with different scheduling policies.

formed as well as Scan-EDF.[10])

Deferred deadlines let us support the same number of real-time streams at the disk with smaller request sizes than with deadlines that are not deferred. Hence, deferred deadlines also reduce aperiodic request response times because aperiodic requests do not wait behind large real-time requests. Other factors affecting response times are aperiodic request arrival, multiple data stream rates, and the use of disk arrays.[10] However, they do not significantly affect the performance trends presented in Figure 2.

Effect of SCSI bus contention. In today's systems, a peripheral-device bus such as SCSI or IPI (Intelligent Peripheral Interface) connects disks to the rest of the components. To amortize the costs of SCSI controllers, we can connect multiple disks to the system on a single bus. For example, a SCSI bus can support 10 MBps (some wider SCSI buses can support 20 MBps). Most disks have raw data rates in the range of 3 to 5 MBps, so we can attach two or three disks to a single SCSI bus without affecting the disks' throughput. However, even when the raw data rate of the SCSI bus is fast enough to support two or three disks, in a real-time environment a shared bus can add delays to individual transfers and cause deadlines to be missed.

To study the effect of SCSI bus contention on the throughput of real-time streams, we simulated three Allicat disks attached to a single SCSI bus. The raw data rate of each disk was 3.8 MBps, which implied a total throughput that slightly exceeded the SCSI bandwidth of 10 MBps. However, because of seek and latency penalties for each access, the disks did not sustain their 3.8 MBps rates for long periods of time.

The SCSI bus is priority arbitrated: If more than one disk tries to transfer data, the disk with higher priority always gets the bus. Hence, real-time streams supported by lower priority disks can get starved if a disk with higher priority continues to transmit data. We can obtain better performance with arbitration driven by other policies such as round-robin or round-robin with a time slice. For multimedia applications, other channels such as IBM's proposed SSA (Serial Storage Architecture), which operates as a time-division-multiplexed channel, are less complex and better able to guarantee deadlines.

Figure 3 shows the impact of SCSI bus contention on the number of streams a system can support. The streams supported are less than three times those of an individual disk's real-time request capacity, mainly because of contention on the bus. At a five-track request size, the ratio of the number of streams supported in a three-disk configuration to that of a single-disk configuration varies from 2.1 with extended deadlines to 1.8 without extended deadlines. Deadline extension increases the chances of meeting deadlines, smoothing over the bus-contention delays. In Figure 3, we assumed that the numbers of streams on the three disks differ at most by one. If we let the higher priority disk support more real-time streams, the total throughput of real-time streams for the three disks was lower — even when we increased the number of streams supported at the first disk by one. For example, with a five-track request size and extended deadlines, Scan-EDF supported 15, 14, and 14 streams at the three disks. When we raised the number to 16 at the first disk, Scan-EDF supported only seven streams each at the second and third disks.

The optimal request size is related mainly to the relative transfer speeds of the SCSI bus and the raw disk. Larger blocks make disk transfers more efficient, but disks with lower priority see longer delays and are more likely to miss deadlines. Shorter blocks make disk transfers less efficient, but the latency to get access to the SCSI bus is shorter.

Most modern disk arms have small buffers for storing the data they are currently reading. Normally, a disk-arm buffer fills at the media transfer rate (in our case, 3.8 MBps) and transfers data out at the SCSI bus rate (10 MBps). With a disk-arm buffer, an individual disk can initiate SCSI transfers in an intelligent fashion to maintain the SCSI data rate: Individual transfers are completed across the SCSI bus as they are being completed at the disk surface. IBM's Allicat drive uses such a policy to transfer data in and out of its 512-Kbyte arm buffer, and the simulations we present here take that policy into account. With no arm buffer, the effective data rate of a SCSI bus falls to the media transfer rate or lower.

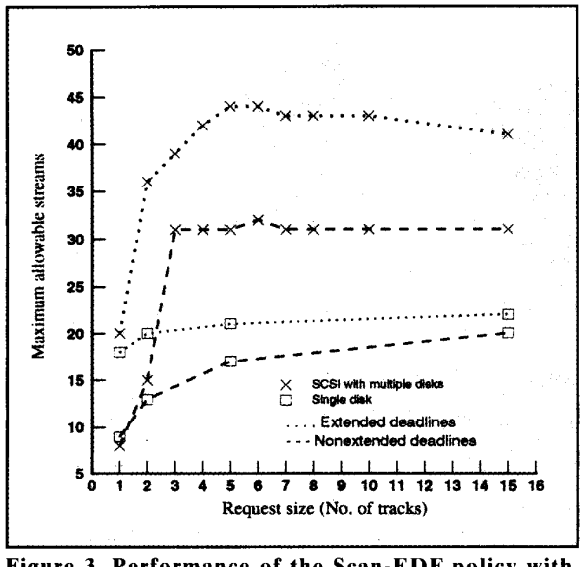


Figure 3. Performance of the Scan-EDF policy with SCSI bus contention.

Admission control

We need to apply some sort of admission control to guarantee the required data rate for a given I/O stream. If the requests are admitted without any limit, the system load might become so large that no stream can meet its deadlines.

Traditionally, real-time system de-

signers make worst-case assumptions about seek and latency overheads to set a lower bound on random disk service. Another way to make service times more predictable is to increase the request size so that overheads form a smaller fraction of the request service time. This approach can result in large demands on buffer space. Our approach is to reduce service-time overhead by achieving more efficient disk-arm use — through an optimized service schedule and large requests. Reduced random overheads make service time more predictable, while deadline extensions further reduce the uncertainties of meeting the deadlines.

An analysis we presented elsewhere showed how the Scan-EDF policy can guarantee deadlines.[10] A complete multimedia system requires several correlated services for each stream (disk, SCSI bus, and the processor). Simultaneous analysis of all these services quickly becomes cumbersome. A simple approach would be to estimate the achievable utilizations through measurement or analysis, and then underutilize the system to decrease the probability of deadline misses to an acceptable level.

Buffering is very effective for minimizing variations in service. However, current systems typically use part of the main memory as an I/O cache. Individual request streams do not have good control over the cache space, and management policies such as least recently used are unlikely to be effective. Because multimedia streams increase the demands for buffer space, we need better policies for distributing this space between real-time and non-real-time requests.

It is more effective to use buffer space for delayed deadlines than for larger request sizes, but large block sizes also help ensure real-time service. File systems should allocate data in blocks of, say, 64 Kbytes or larger. Most current systems use block sizes on the order of 4 Kbytes — far too small for real-time applications. However, allocating disk space in large blocks can result in internal fragmentation and wasted space. Hence, future file systems might have to support multiple block sizes.

We need to develop new disk-scheduling algorithms that combine the features of traditional seek-optimization techniques with traditional real-time scheduling. We should also find more effective replacements for priority-driven peripheral-device buses such as SCSI. And we need to investigate how operation over a local area network and processor scheduling affect real-time delivery guarantees. ■

Acknowledgments

Discussions with Barbara Simons, Robert Morris, and Roger Haskin 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 Conf.*, ACM Press, New York, 1988, pp. 109-116.

2. M.Y. Kim, "Synchronized Disk Interleaving," *IEEE Trans. Computers*, Vol. 35, No. 11, Nov. 1986, pp. 978-988.

3. K. Salem and H. Garcia-Molina, "Disk Striping," *Fourth Int'l Conf. Data Eng.*, IEEE CS Press, Los Alamitos, Calif., Order No. 827 (microfiche only), 1986, pp. 336-342.

4. A.L.N. Reddy and P. Banerjee, "An Evaluation of Multiple-Disk I/O Systems,"*IEEE Trans. Computers*, Vol. 38, No. 12, Dec. 1989, pp. 1,680-1,690.

5. T.H. Lin and W. Tarng, "Scheduling Periodic and Aperiodic Tasks in Hard Real-Time Computing Systems," *Proc. SIGMetrics*, ACM Press, New York, 1991, pp. 31-38.

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, Computer Science Dept., Univ. of California, Berkeley, 1991.

7. J. Yee and P. Varaiya, "Disk Scheduling Policies for Real-Time Multimedia Applications," tech. report, Computer Science Dept., Univ. of California, Berkeley, 1992.

8. H.M. Vin and P.V. Rangan, "Designing File Systems for Digital Video and Audio," *Proc. 13th ACM Symp. Operating System Principles*, ACM Press, New York, 1991, pp. 81-94.

9. C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM*, Vol. 20, No. 1, Jan. 1973, pp. 46-61.

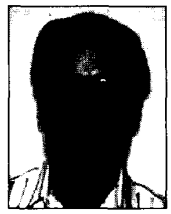
10. A.L.N. Reddy and J. Wyllie, "Disk Scheduling in a Multimedia I/O System," *Proc. ACM Multimedia Conf.*, ACM Press, New York, 1992, pp. 225-233.

11. J.P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," *Proc. 11th Real-Time Systems Symp.*, IEEE CS Press, Los Alamitos, Calif., Order No. 2112, 1990, pp. 201-212.

12. W.K. Shih, J.W. Liu, and C.L. Liu, "Modified Rate Monotone Algorithm for Scheduling Periodic Jobs with Deferred Deadlines," tech. report, Computer Science Dept., Univ. of Illinois, Urbana-Champaign, 1992.

A.L. Narasimha Reddy is a research staff member at the IBM Almaden Research Center in San Jose, California. His research interests are I/O systems, parallel processing, and computer architectures.

Reddy received his BTech in electronics and electrical engineering from the Indian Institute of Technology, Kharagpur, India, in 1985 and his MS and PhD in electrical engineering from the University of Illinois at Urbana-Champaign in 1987 and 1990, respectively. At the University of Illinois, he received an IBM fellowship. He is a member of the IEEE Computer Society.



James C. Wyllie has been a research staff member at the IBM Almaden Research Center since 1979, where he has worked on operating systems, file systems, and application software for personal computers. His current research interest is file servers for advanced interactive digital media.

Wyllie received his BS in computer science from Pennsylvania State University in 1973 and his MS and PhD in computer science from Cornell University in 1976 and 1979, respectively.

Readers can contact the authors at IBM Almaden Research Center, 650 Harry Rd., San Jose, CA 95120; e-mail {reddy, wyllie}@almaden.ibm.com.