On Data Logging in Real-Time Process Control Systems *

Jinho Kim

Dept. of Computer Science
Kangwon National University
Chooncheon, KOREA 200-701

Kang G. Shin

Real-Time Computing Laboratory
Dept. of EECS
University of Michigan
Ann Arbor, Michigan 48109

Abstract

Real-time process control systems require hard real-time data logging tasks to access all of the data sampled from plants and monitoring tasks to analyze the status of the plants. This paper presents an integrated scheduling scheme for hard real-time data logging tasks and soft real-time monitoring tasks, both types of which require disk I/Os. Conventional disk scheduling policies like Shortest-Seek-Time-First (SSTF), SCAN, and C-SCAN, reorder disk I/O requests to minimize the disk head's seek time, thus making it very difficult to guarantee the timely completion of the tasks requiring disk I/Os. Our scheduling scheme provides timeliness guarantees by serializing all disk I/O requests of these tasks based on their priorities. Because it doesn't require any special-purpose real-time disk scheduling algorithm, the proposed scheme can be implemented on top of general-purpose operating systems like UNIX.

1 Introduction

Real-time monitoring and control applications such as manufacturing automation, distributed process control (DCS), power plant control, and network management, need to access the data sampled from plants and maintain the log of the data in disk files. The data log is a history of plants' state, and can be used for analyzing the trend of certain subsystems, controlling the activities of plants (e.g., quality control), generating reports, etc.[7, 13]. The overall architecture of a real-time monitoring and control system is shown in Figure 1.

As shown in this figure, real-time monitoring and control systems have three major types of tasks: sensor reading, data logging, and monitoring. Sensor reading tasks acquire the current state of equipment or sensors at fixed intervals and store them in their own main-memory buffers. In order to monitor and control the plants effectively, these tasks have to obtain the plants' status within their sampling interval. The sampled data will be sent to data logging tasks and monitoring tasks.

Data logging tasks store the data acquired by sensor reading tasks in disk files, which maintain the entire history of the data. These tasks must save all of the sampled data in disk files before the buffer of sensor reading tasks runs out. Therefore, data logging must meet the hard deadline imposed by the buffer size and the sampling rate of each sensor reading task.

Monitoring tasks receive the current status of plants from sensor reading tasks and display it on a graphical terminal. They also access some of the logged data stored in disk files and will present their statistical trend or generate some reports. The statistics and reports will be used to analyze and control the plant activities. The monitoring tasks will be invoked by plant operators and their deadlines are usually not so "hard". Thus, we assume that these tasks are aperiodic and soft real-time.

Several approaches to scheduling both periodic hard real-time tasks and soft aperiodic real-time tasks have been proposed in the literature [2, 3, 4, 8, 9, 11]. Based on the Rate Monotonic (RM) scheduling algorithm [5], Deferrable Server (DS) [3], Priority Exchange (PE) [3], Extended Priority Exchange (EPE) [8], and Slack Stealing (SS) [4] algorithms have been proposed to schedule aperiodic tasks while meeting the deadline of all periodic tasks. In order to enhance processor utilization, the authors of [9] extended these algorithms to the EDF-based algorithms called Dynamic Priority Exchange (DPE) server, Total Bandwidth Server(TBS), and EDL server [1]. Shin and Chang [12] proposed a so-called Reservation-Based algorithm that maximizes the probability of meeting

*The work reported in this paper was supported in part by the Office of Naval Research under grant N00014-92-J-1080, and by the NSF under grants DDM-9313222 and IRI-9504412 and by the LG Yonam Foundation of Korea.

0-8186-7626-4/96 $5.00 © 1996 IEEE

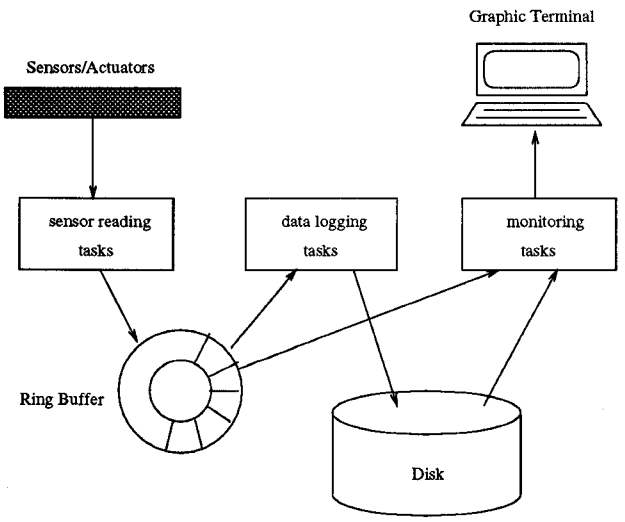

Figure 1: Overall architecture of real-time monitoring and control systems.

aperiodic task deadlines while guaranteeing all periodic deadlines. All of these algorithms require *a priori* knowledge of the worst-case execution time of each task. However, this knowledge is difficult to obtain in practice, because under the current multiprogramming environment, it is very difficult to determine the worst-case response time of each disk I/O request. When a task requests to write a disk block, it sends a disk I/O request to the disk controller. The disk controller processes disk I/O requests from different tasks with its own scheduling algorithm such as First-Come-First-Serve (FCFS), Shortest-Seek-Time-First (SSTF), SCAN, and C-SCAN [6, 10, 14]. All of the disk scheduling algorithms except FCFS reorder the sequence of disk I/O requests to minimize the seek time of the disk head. A higher-priority task cannot precede the disk I/O requests which have already been issued by lower-priority tasks, and hence must wait until these disk I/O requests are served completely. This, in turn, makes it difficult to guarantee the timely completion of those tasks requiring disk I/Os such as data logging. It is also unrealistic to use a special disk scheduling algorithm which provides real-time guarantees for disk I/O requests, because it requires modification of the kernel of an operating system to implement the special disk scheduling algorithm.

The primary goal of this paper is to develop an integrated scheduling algorithm for hard real-time data logging tasks and soft real-time monitoring tasks, both of which require disk I/Os. The algorithm is designed on top of existing disk scheduling algorithms. Our basic idea is to serialize all disk I/O requests. Only

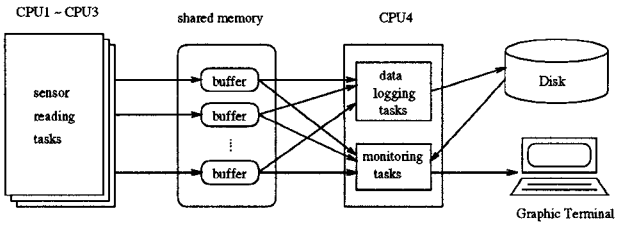

Figure 2: System architecture of data logging.

one disk I/O request at a time can be sent to the disk controller. Thus, we can eliminate the effects of disk scheduling for the disk I/O request. To achieve this, the data logging tasks are scheduled with a non-preemptive rate monotonic policy. A data logging task, therefore, accesses the disk exclusively until it finishes the access completely, while blocking other disk I/O requests. All disk I/O requests of soft real-time monitoring tasks are scheduled in the time intervals left unused by data logging tasks. The monitoring tasks without accessing the disk run with lower priority than data logging tasks. These tasks will run on the CPU when data logging tasks access the disk in order to increase the CPU utilization. They will be preempted by the data logging tasks whenever the data logging tasks need the CPU, so they won't delay the execution of the data logging tasks. Since the proposed scheme doesn't assume any special real-time disk scheduling algorithm, it can be implemented on top of general-purpose operating systems like UNIX.

2 Problem Definition

2.1 System Architecture for Data Logging

As shown in Figure 2, we assume that a data logging system is built on a shared-memory multiprocessor. In order to support timeliness, sensor reading tasks run on dedicated CPU boards and store sensor values into their own buffers in shared memory. The sensor reading tasks can be scheduled with the traditional rate monotonic algorithm [5]. The data logging and sensor reading tasks run on different CPUs. These tasks should copy the buffers into the disk before buffers get overrun. The monitoring tasks execute on the same CPU boards as the data logging tasks, access the logged data, and display them on the graphic terminal as shown in Figure 2.

2.2 Problem Formulation

Let n be the number of sensor reading tasks, SR_1, SR_2, ..., SR_n. Each SR_i reads the values of external objects (e.g., sensors) and stores them in its own buffer B_i in shared memory. Each buffer has the limited size (the size of B_i is denoted as $\mid B_i \mid$) and is used as a circular queue.

SR_i accesses a finite set of objects $O_i = \{ o_{i,1}, o_{i,2}, ..., o_{i,p} \}$. We assume that $O_i \cap O_j = \phi$, for any i, j where $i, j \leq n, i \neq j$.

The value of each object $o_{i,j}$ is represented with its own size in bytes (for example, 4 bytes for an integer number, 8 bytes for a double precision real number, and so on). We denote the size of $o_{i,j}$ as $\mid o_{i,j} \mid$. In general, each object, $o_{i,j}$, is sampled periodically, with period $p_{i,j}$. The size of the history of an object $o_{i,j}$ is a function of time and is defined as:

$$\mid hist(o_{i,j}, t) \mid = \mid o_{i,j} \mid \cdot \frac{t}{p_{i,j}}$$

where t is the time measured from the system start. The total size of all object histories is:

$$\begin{aligned} \mid hist(SR_i, t) \mid &= \sum_{o_{i,j} \in O_i} \{ \mid hist(o_{i,j}, t) \mid \} \\ &= \sum_{o_{i,j} \in O_i} \left\{ \frac{\mid o_{i,j} \mid}{p_{i,j}} \cdot t \right\} \end{aligned}$$

Using $\mid B_i \mid$ and the sampling rate of objects, we can calculate the time, PBF_i, to fill B_i as: $PBF_i = \mid B_i \mid / \mid hist(SR_i, t) \mid$. Unless we dump a portion of B_i within the period PBF_i, B_i will get overrun. Therefore, data logging (DL) tasks have to save some portion (actually one half, in order to keep running SR tasks concurrently during the other half) of each buffer in the disk before its deadline ($= PBF_i/2$).

Next, we define the worst-case delay in copying the entire B_i into the disk. The worst-case delay in accessing one disk block is $t_{blockIO} = S_{max} + L + L * \mid block \mid / \mid track \mid$, where S_{max} is the max seek time, L is the time of one revolution of the disk, $\mid block \mid$ is the length of a disk block, and $\mid track \mid$ is the length of a disk track. The maximum delay in copying B_i's content into the disk is:

$$TBS_i = \frac{\mid B_i \mid}{\mid block \mid} \cdot t_{blockIO} + CPU_{max}$$

where CPU_{max} is the maximum CPU processing time to save B_i in the disk, including context switching overhead and main-memory access delay.

While sensor reading tasks are invoked periodically, monitoring tasks are requested aperiodically and dynamically by plant operators (or other users). Graphic presentation is, in general, less important than data acquisition. Thus, we assume that monitoring tasks are soft real-time (and aperiodic). A set, M, of monitoring tasks are divided into two categories: short-term monitoring tasks $M_s = \{ m_{s,1}, m_{s,2}, ..., m_{s,p} \}$ and and long-term monitoring tasks $M_l = \{ m_{l,1}, m_{l,2}, ..., m_{l,q} \}$. Each short-term monitoring task, $m_{s,i}$, is assumed to access the current status (or the recent short-term history) of objects in main-memory buffers, so it doesn't require any disk access. On the other hand, each long-term monitoring task, $m_{l,i}$, needs to access a long history of objects stored in the disk and analyze their statistical trends.

3 The Proposed Scheduling Scheme

This section describes an integrated algorithm to schedule data logging and monitoring tasks. The algorithm classifies tasks into three types and uses different scheduling policies for data logging tasks, short-term monitoring tasks, and long-term monitoring tasks. The algorithm schedules tasks to achieve two goals: (1) the data logging tasks must save *all* of the data sampled from sensors in the disk without any data loss; (2) the number of the monitoring tasks missing their deadlines should be minimized.

3.1 Scheduling Data Logging Tasks

As shown in Figure 2, the sampled data of (external) objects are stored first in buffers. Each buffer, B_i, is associated with two parameters: PBF_i, TBS_i, where PBF_i is the same as defined before and TBS_i is the maximum delay in copying B_i into the disk. If we keep all log of data sampled from external objects, some portion of B_i should be saved into the disk periodically before PBF_i and then vacated for new sampled data. (Actually, we should save one half of B_i in order to run the sensor reading tasks concurrently with the saving of the buffer content into the disk. $PBF_i/2$ is the time to fill one half of B_i.)

If the data logging tasks are scheduled with the rate monotonic scheduling algorithm, we can guarantee that one half of each buffer is saved within their deadline, $PBF_i/2$. Their schedulability condition can then be expressed as follows:

$$\frac{TBS_1/2}{PBF_1/2} + \frac{TBS_2/2}{PBF_2/2} + \ldots + \frac{TBS_n/2}{PBF_n/2}$$

$$= \frac{TBS_1}{PBF_1} + \frac{TBS_2}{PBF_2} + \ldots + \frac{TBS_n}{PBF_n}$$
$$\leq n\left(2^{1/n} - 1\right).$$

Under the existing disk scheduling algorithms, the disk I/O requests of a higher-priority task can't preempt the other low-priority disk I/O requests which have already been issued. Thus, we employ a non-preemptive rate-monotonic algorithm so as to prevent any lower-priority task from interfering with the execution of higher-priority tasks. This algorithm will guarantee that the disk I/O requests of data logging tasks be sent to the disk, one at a time. The maximum delay in copying one half of B_i's content into the disk is:

$$TBS_i/2 = \frac{1}{2} \cdot \frac{\mid B_i \mid}{\mid block \mid} \cdot t_{blockIO} + CPU_{max}.$$

3.2 Scheduling Long-Term Monitoring Tasks

Long-term monitoring tasks will access the history of objects stored in the disk to analyze the trend of plant activities or generate reports. The disk I/Os of these tasks shouldn't compromise the deadline guarantees of data logging tasks. When a disk I/O is issued by one (say $M_{l,i}$) of long-term monitoring tasks just before a data logging task (DL_j) starts, under the existing disk scheduling algorithms such as SCAN or SSTF, DL_j's disk I/O must wait until the disk I/O of $M_{l,j}$ is completed. Thus, DL_j may miss its deadline.

In order to avoid any deadline miss, we have chosen to schedule the disk I/O requests of long-term monitoring tasks within the time interval left unused by data logging tasks. To implement this scheduling algorithm, we need to calculate the time interval left unused by every data logging task. By using this unused time interval, we define the disk available (DA) time at a given time t as :

$$DA(t) = \begin{cases} w & \text{if } t \text{ is in the unused interval and the} \\ & \text{next data logging task starts at } w + t, \\ 0 & \text{if } t \text{ is not in the unused interval.} \end{cases}$$

$DA(t)$ represents the disk available interval at time t until the next data logging task starts.

The disk I/O requests of long-term monitoring tasks can be scheduled if their maximum execution time is less than, or equal to, $DA(t)$ at time t. In order to serialize the execution of long-term monitoring tasks, we use an exclusive lock for the disk. Every long-term monitoring task should acquire the lock before accessing the disk, and release the lock after accessing the disk. With this lock, while a long-term monitoring task is accessing the disk, no other task can access the disk.

3.3 Scheduling Short-Term Monitoring Tasks

We assume that short-term monitoring tasks access the current data (or the most recent history) of objects stored in main memory buffers. Short-term monitoring tasks will be invoked aperiodically by plant operators and use the CPU without accessing the disk.

When data logging tasks or long-term monitoring tasks access the disk, they don't use the CPU. Disk I/O time is generally much longer than CPU processing time. If we schedule short-term monitoring tasks while the other tasks are accessing the disk, the utilization of CPU will improve. We assign lower priority to short-term monitoring tasks than data logging tasks and long-term monitoring tasks. Upon completion of a data logging task's access to the disk, it will preempt the short-term monitoring task and will continue its execution. So, short-term monitoring tasks will not delay the execution of data logging tasks and long-term monitoring tasks.

4 Analysis and Discussion

We proposed a scheduling algorithm for hard real-time tasks requiring disk I/Os under general operating systems and existing disk scheduling algorithms. The proposed scheme specializes disk I/O requests in order to preserve the deadline guarantees of disk I/O requests. Thus, the scheme may suffer from low utilization of disk bandwidth, and each disk I/O will require a long seek time.

Several disk scheduling algorithms have been studied to enhance the performance of disk I/Os, and their performance has been evaluated extensively [6, 10]. Typical disk scheduling algorithms are first-come-first-serve (FCFS), shortest-seek-time-first (SSTF), and SCAN. These algorithms except FCFS hold disk I/O requests in the disk queue and reorder the sequence of disk I/O requests to minimize the seek time. The previous studies have shown them to achieve high disk bandwidth utilization and low response time. However, they considered neither a real-time environment nor real-time data logging. We will compare the existing disk scheduling algorithms with the proposed scheme in the context of hard real-time and data logging applications. We adapt the disk model and the parameters used in [10] to analyze the performance of

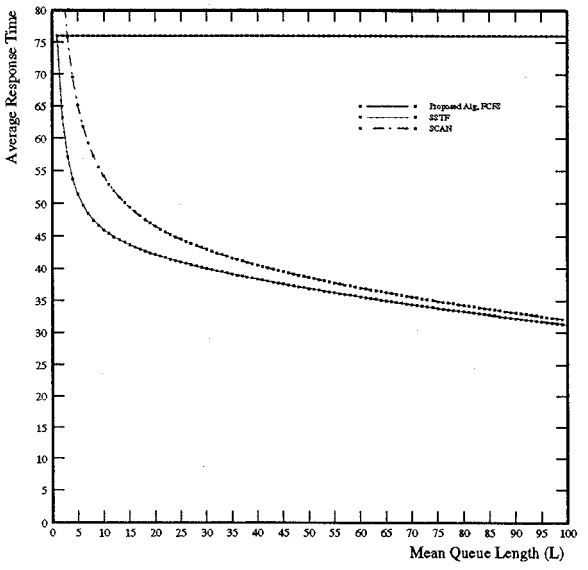

Figure 3: Average response time of disk I/O requests.

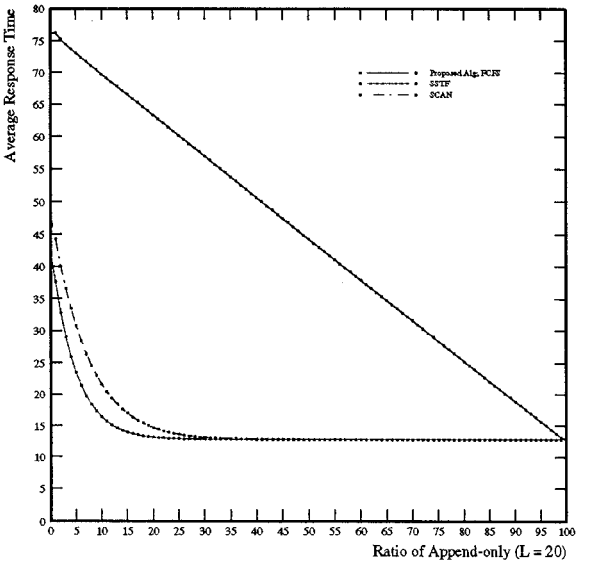

Figure 4: Average response time of disk I/O requests over the ratio of append-only.

disk scheduling algorithms. Table 1 summarizes the parameters of the disk model.

cylinders/disk	200
tracks/cylinder	20
sectors/track	50
minimum seek time	30ms
maximum seek time	130ms
average rotational latency	12.5ms

Table 1: Parameters of the disk model.

Figure 3 shows the average response time of disk I/O requests over average queue length (denoted as L) under the assumption of the uniform distribution of disk sectors. The proposed scheme checks the schedulability of tasks in advance and accepts only these tasks which can be finished in time. If we don't consider this admission control, the proposed scheme processes disk I/O requests in the same way as FCFS. As shown in this figure, the proposed scheme (and FCFS) has better performance than (or equal to) SSTF and SCAN when the average queue length ≤ 3. The longer queue length, the worse performance the proposed scheme has than SSTF and SCAN. In hard real-time systems, usually, there aren't many tasks to be activated simultaneously, so we don't expect the average queue length to be so large in real-time systems.

Data logging tasks store the sampled data in sequential files, thus producing append-only operations. Append operations will access the same disk cylinder or adjacent cylinder of the previous disk I/O request. The ratio of append-only operations will therefore affect the performance of disk scheduling algorithms. Figure 4 shows the average response time of SSTF, SCAN, and the proposed scheme over the ratio of append-only operations. If the ratio is higher than 35%, SSTF and SCAN will have the same response time, meaning that the disk driver will process dominantly append-only operations under SSTF and SCAN. At the same time, the other type of disk I/O requests can hardly be selected for service, and thus, the possibility of missing their deadline would be high. If every operation is append-only, all of disk scheduling algorithms have the same performance. At a high ratio of append-only operations, the proposed scheme would have a similar utilization of disk bandwidth to SCAN and SSTF.

Figure 5 shows the average response time of SCAN for four different mean queue lengths over the ratio of append-only operations. If the mean queue length gets smaller, the performance of the proposed scheme will get closer to that of SCAN. In process monitoring systems, data logging activities are more dominant than long-term monitoring activities such as report generation, quality control, etc.

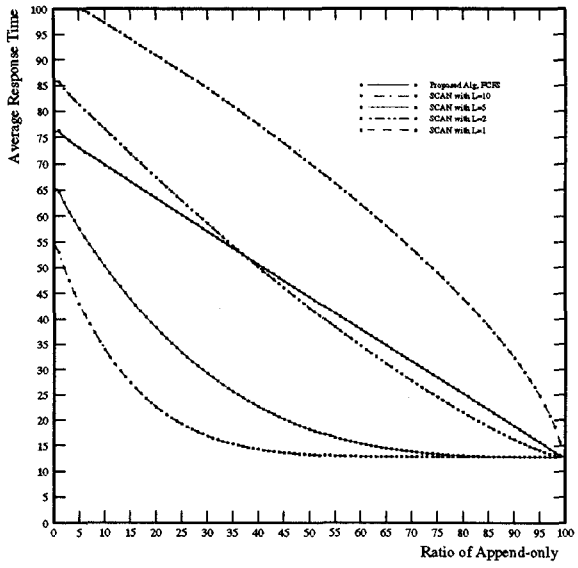

Figure 5: Average response time of SCAN with various mean queue lengths.

5 Conclusion

In this paper, we have looked into the problem of scheduling tasks in real-time process monitoring/control systems, which require disk I/Os. Specifically, we proposed an integrated scheme for scheduling hard periodic real-time data logging tasks and soft aperiodic real-time monitoring tasks. All of the existing disk scheduling algorithms such as SCAN, C-SCAN, or SSTF, reorder disk I/O requests to minimize the disk seek time. Hence, application tasks can't control the sequence of processing their disk I/O requests, thereby making it difficult to guarantee the deadline of a task requiring disk I/Os.

The basic idea of the proposed scheme is to serialize the disk I/O requests from data logging tasks which are periodic and hard real-time. These requests are scheduled using a non-preemptive rate monotonic policy. The disk I/O requests of monitoring tasks are scheduled during the time intervals left unused by the data logging tasks. Monitoring tasks must check if their disk I/O requests can be finished before a data logging task starts, because a monitoring task's disk I/O is not preempted once it is allowed to start. In order to enhance the CPU utilization, the monitoring tasks without disk I/Os (i.e, short-term monitoring tasks) are scheduled concurrently with the disk accesses by data logging tasks (or long-term monitoring tasks). The short-term monitoring tasks are assigned lower priorities than data logging tasks or long-term monitoring tasks. They will be preempted as soon as a data logging task needs to be executed on the CPU (e.g., after a disk I/O).

We also evaluated the disk I/O performance of the proposed scheme over the existing disk scheduling algorithms (SSTF, SCAN, and FCFS) in the context of process monitoring systems. The proposed scheme may suffer from low utilization of disk bandwidth, but it doesn't require any real-time sensitive disk scheduling algorithm and hence, can be used to implement a real-time monitoring and control system on a general-purpose operating system like UNIX. Moreover, in process monitoring systems, disk I/O requests are dominated by "append-only" data logging activities. The higher the ratio of append-only operations, the closer to SSTF and SCAN the disk bandwidth utilization of the scheme becomes. The performance of the proposed scheme depends on the nature of applications (i.e., the ratio of the append-only operations). We are currently evaluating the proposed scheme for real data logging applications.

References

[1] H. Chetto and M. Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm," *IEEE Trans. on Software Engineering*, vol.15, no.10, pp.1261-1269, October 1989.

[2] K. Jeffay, "Scheduling Sporadic Tasks with Shared Resources in Hard Real-Time Systems", *Proc. Real-Time Systems Symposium*, pp.89-99, 1992.

[3] J. P. Lehoczky, L. Sha, and J. K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," *Proc. Real-Time Systems Symposium*, pp. 261–270, 1987.

[4] J. P. Lehoczky and S. R. Thuel, "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems," *Proc. Real-Time Systems Symposium*, pp. 110–123, 1992.

[5] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[6] M. Seltzer, P. Chen, and J. Ousterhout, "Disk Scheduling Revisited," *Proc. USENIX Winter 1990*, pp. 313–323, 1990.

[7] H. Shimakawa et al., "Acquisition and Service of Temporal Data for Real-Time Plant Monitoring,"

Proc. Real-Time Systems Symposium, pp. 112–118, 1993.

[8] B. Sprunt, J. Lehoczky, and L. Sha, "Exploiting Unused Periodic Time for Aperiodic Service Using The Extended Priority Exchange Algorithm," *Proc. Real-Time Systems Symposium*, pp. 251–258, 1988.

[9] M. Spuri and G. C. Buttazzo, "Efficient Aperiodic Service Under Earliest Deadline Scheduling," *Proc. Real-Time Systems Symposium*, pp. 2–11, 1994.

[10] T. Teorey and T. Pinkerton, "A Comparative Analysis of Disk Scheduling Policies," *Communications of the ACM*, vol. 15, no. 3, pp. 177–184, 1972.

[11] S. R. Thuel and J. P. Lehoczky, "Algorithms for Scheduling Hard Aperiodic Tasks in Fixed-Priority Systems Using Slack Stealing," *Proc. Real-Time Systems Symposium*, pp.22–33, 1994.

[12] K. G. Shin and Y.-C. Chang, "A Reservation-Based Algorithm for Scheduling Both Periodic and Aperiodic Real-Time Tasks," *IEEE Trans. on Computers*, vol. 44, no. 12, pp. 1405–1419, December 1995.

[13] D. D. Val and A. Vina, "Applying RMA to improve a high-speed, real time data acquisition system", *Proc. Real-Time Systems Symposium*, pp.159-164, 1994.

[14] N. Wilhelm, "An anomaly in disk scheduling: a comparison of FCFS and SSTF seek scheduling using an empirical model for disk accesses", *Communications of the ACM*, vol. 19, no. 1, pp.13-17, 1976.