

Combined Scheduling of Hard and Soft Real-Time Tasks in Multiprocessor Systems

B. Duwairi and G. Manimaran

Department of Electrical and Computer Engineering
Iowa State University, Ames, IA 50011, USA
{dbasheer, gmani}@iastate.edu

Abstract. Many complex real-time applications involve combined scheduling of hard and soft real-time tasks. In this paper, we propose a combined scheduling algorithm, called *Emergency Algorithm*, for multiprocessor real-time systems. The primary goal of the algorithm is to maximize the schedulability of soft tasks without jeopardizing the schedulability of hard tasks. The algorithm has the inherent feature of dropping already scheduled soft tasks by employing a technique that switches back and forth between two modes of operation: *regular mode*, in which all tasks (hard and soft) are considered for scheduling without distinction; *emergency mode*, in which one or more already scheduled soft tasks can be dropped in order to accommodate a hard task in the schedule. The proposed algorithm has been evaluated by comparing it with the iterative server rate adjustment algorithm (ISRA) [6]. The simulation results show that emergency algorithm outperforms ISRA by offering better schedulability for soft tasks while maintaining lower scheduling overhead.

1 Introduction

Many real-time applications involve combined scheduling of hard and soft real-time tasks. Hard real-time tasks have critical deadlines that are to be met in all working scenarios to avoid catastrophic consequences. In contrast, soft real-time tasks (e.g., multimedia tasks) are those whose deadlines are less critical such that missing the deadlines occasionally has minimal effect on the performance of the system.

In military applications, such as attack helicopters, multimedia information is being used to provide tracking and monitoring capabilities, that can be used directly to engage a threat and avoid crashing unexpectedly [6]. It is possible to allocate separate resources for each of hard and soft tasks. However, sharing the available resources among both types of tasks would have enormous economical and functional impact. Therefore, it is necessary to support combined scheduling of hard and soft real-time tasks in such systems, in which multiprocessors are increasingly being used to handle the compute-intensive applications.

Real-time tasks, beside being hard or soft, can be periodic or aperiodic. Therefore, in combined scheduling we may encounter the following task combinations: (1) periodic hard tasks with aperiodic soft tasks, (2) periodic hard tasks with periodic soft tasks, (3) aperiodic hard tasks with aperiodic soft tasks, or (4) aperiodic hard tasks with periodic soft tasks. Scheduling any of these task combinations in multiprocessor systems is a

challenging problem due to heterogeneous mix of tasks and dynamic nature of workload and multiple processors.

Maximizing the schedulability of soft tasks without jeopardizing the schedulability of hard tasks, and minimizing the scheduling overhead are among the most important design objectives that a combined scheduling algorithm attempts to achieve. It is obvious that achieving both objectives at the same time is not trivial as it requires consideration of many parameters and constraints.

Many researchers have investigated mechanisms to support combined scheduling of hard and soft real-time tasks. A number of scheduling approaches were adopted to handle a mix of hard and soft real-time tasks in uniprocessor systems. Examples are, Constant Bandwidth Server (CBS) [11] and [5], Total Bandwidth Server (TBS) [10] and [11], Constant Utilization Server [4], Deferrable Server (DS) [8], Priority Exchange (PE) [8], Extended Priority Exchange (EPE) [12], Dual priority Scheduling [3] and Dynamic Slack Stealing (DSS) [2]. Other scheduling schemes were presented in [1], and [7]. These approaches can be made applicable to multiprocessor systems by static allocation of tasks to processors. Obviously, the main problem of static allocation is that it is not able to reclaim unused time on one processor to schedule tasks allocated to another processor.

In this paper, we propose an algorithm for combined scheduling of hard and soft real-time tasks in multiprocessor systems. The primary goal of the proposed algorithm is to maximize the schedulability of soft tasks without jeopardizing the schedulability of hard tasks. The algorithm has the inherent feature of degrading QoS, by dropping already scheduled soft tasks, by employing a technique that switches back and forth between two modes of operation: regular mode in which all tasks (soft and hard) are considered for scheduling without distinction; emergency mode in which one or more already scheduled soft tasks can be dropped in order to accommodate a hard task in the schedule.

The rest of the paper is organized as follows: in section 2, we provide a description of the related work to this paper. The proposed algorithm is presented in section 3. Simulation studies are presented in section 4, followed by the conclusion in section 5.

2 Related Work

In multiprocessor systems, the multimedia server based combined scheduling algorithm [6] is the most relevant work. We first introduce the system model which is the same as used in [6], then we describe combined scheduling in the context of multimedia server.

2.1 System Model

The system model can be outlined as follows:

1. We consider multiprocessor system with (P) identical processors. The system has tasks with the following characteristics:
 - (a) Aperiodic hard real-time tasks. Each task T_{ih} is characterized by its ready time (r_{ih}), worst-case computation time (c_{ih}), and absolute deadline (d_{ih}).

- (b) n multimedia streams, each stream S_i is characterized by its period (P_i) and worst-case computation time (C_i).
- 2. All multimedia streams are mapped to one multimedia server, S_s , of period P_s and worst-case computation time C_s (multimedia server concept is described in subsection 2.2).
- 3. Each multimedia server instance is considered as a separate task. For example, the j th server instance has ready time $(j - 1)P_s$, deadline jP_s , and computation time C_s .
- 4. No task migration or preemption is allowed.
- 5. The scheduling algorithm has complete knowledge about the currently active task set, but not about any new tasks that may arrive while scheduling the current set.

2.2 Multimedia Server [6]

The combined scheduling of aperiodic hard real-time tasks and multimedia streams in multiprocessor environment was addressed in [6] by introducing the concept of *Multimedia Server*. The server is a periodic task generated dynamically to accommodate multimedia streams and basically used for the purpose of reducing the total number of soft tasks considered for scheduling, because considering each multimedia task instance as a hard real-time task and scheduling it without having the multimedia server will increase the scheduling cost.

The creation of multimedia server can be done according to *proportional* or *individual* allocation schemes. For example, in proportional allocation, which is used in our experiments, the multimedia server is created as a periodic stream with period P_s equals to the smallest period of all multimedia streams in the system. Assuming that there are n different multimedia streams in the system, where each stream S_i is characterized by its period P_i and execution time C_i , then each stream instance is divided among P_i/P_s server instances such that the computation time of the multimedia server instance C_s is given by $\sum(C_i \times \frac{P_s}{P_i})$.

An incoming multimedia stream is initially mapped to an already existing multimedia server – otherwise a new server is created – to be scheduled with tasks waiting in the queue. If the whole task set is not schedulable, the incoming multimedia stream is rejected and removed from the multimedia server. The situation is different for incoming hard tasks; upon the arrival of a hard task, the scheduler tries to schedule this new task with all other already guaranteed tasks, hard and soft, if the hard task is found to be infeasible, either it is rejected directly – if hard tasks have no priority over already guaranteed tasks – or the QoS of already scheduled multimedia server must be degraded repeatedly until the new hard task become feasible. For this purpose, an **Iterative Server Rate Adjustment (ISRA)** algorithm is used to achieve the required QoS degradation. We refer the readers to [6] for more details about ISRA.

2.3 Motivation and Contribution

The multimedia server based scheduling strategy proposed in [6] performs QoS degradation outside the dynamic scheduler (i.e., using separate QoS degradation algorithm),

which means that the scheduler has to be invoked several times to test the schedulability of the task set after each time the QoS is degraded. In order to avoid all these complications included in quantifying the level of degradation, and to avoid the high scheduling overhead resulting from the fact of invoking the scheduler several times for the same task set, we propose an algorithm that is inherently capable of degrading QoS of multimedia server by dropping some of its instances, when necessary, while building the schedule (i.e., the scheduler is invoked only once for each task set).

3 Proposed Work

In this section, we describe the proposed emergency algorithm, which is a more sophisticated version of myopic algorithm [9]. The main difference is that our algorithm relies on a distinction between hard and soft real-time tasks when making the scheduling decisions in such a way that the schedulability of hard tasks is not affected by the existence of soft tasks. Therefore, by maintaining some additional information, the algorithm has the inherent feature for QoS degradation.

3.1 Main Idea

The emergency algorithm proposed in this paper, keeps track of the maximum *spare time* available on each processor. The value of this maximum spare time on processor P_j is defined as the difference between the earliest available time (EAT) of P_j in regular mode (EAT_{jr}) and that in emergency mode (EAT_{je}), where EAT of a processor is the earliest time at which the processor is ready for executing a task. EAT_{jr} and EAT_{je} are updated for each processor while building the schedule. The spare time of a certain processor represents the maximum amount of CPU time that can be reclaimed on that processor by dropping specific soft tasks. The dropping policy is explained below.

The algorithm is designed such that the spare time can be made available at the end of the schedule on each processor. When a new hard task is found to be infeasible (Task T_i is considered feasible if $EST_i + C_i \leq d_i$, where C_i is the worst-case execution time, d_i is the deadline, and EST_i is the earliest start time of task T_i . Otherwise, it is considered infeasible), the scheduler checks the information it keeps about already scheduled tasks to see if it is possible to accommodate the new hard task by reclaiming the spare time on one of the processors. A straightforward but expensive approach to do this is by initially dropping some of the already scheduled soft tasks on that processor and then reschedule the new hard task with the remaining already scheduled tasks.

In our algorithm, we follow a more computationally efficient approach to add the new hard task to the schedule. We start by selecting the processor that have enough spare time to make the new hard task feasible, then we reclaim that time as follows: the soft tasks that are scheduled after the last hard task on that processor are dropped directly. If their total computation time is equal to the spare time, the new hard task is added on that processor. Otherwise, the last hard task is shifted back by an amount of time equals to spare time minus total computation time of soft tasks dropped in the previous step. This may result in overlapping between the shifted task and other hard tasks. To resolve this overlapping, the next hard task is also shifted back such that it will

have a finish time equals to the start time of the previously shifted task. We keep doing this shifting process until there is no overlapping between hard tasks.

Any soft task that overlaps with the shifted hard tasks is dropped directly and its computation time is decremented from the spare time. After reclaiming the total spare time, the new hard task is added to the end of the schedule on that processor. We emphasize here that no task is shifted beyond its ready time because this was taken care of originally when the spare time is updated on each processor. It is also important to notice that the order of execution of the tasks scheduled on the selected processor remains the same and only some soft tasks are dropped on the way of shifting the hard tasks back.

3.2 Algorithm Details

Similar to myopic [9], emergency algorithm is a heuristic search algorithm that tries to construct a full feasible schedule by extending a partial schedule by one task at a time from window of k tasks (called the feasibility check window). The algorithm (shown on next page), starts its operation in the regular mode and stays there as long as all hard tasks encountered so far are feasible (step 2). For simplicity and convenience, the algorithm drops any infeasible soft task directly rather than backtracking or considering other options (step 2a). Among the feasible tasks considered in the feasibility check window, the algorithm always selects the one with the minimum heuristic value (the heuristic function for task T_i is given by $H_i = d_i + EST_i$). The selected task is scheduled on processor P_j that has the minimum earliest available time (EAT) to extend the current partial schedule (step 2b).

We mentioned earlier that this algorithm keeps track of resources reserved for soft tasks as they get scheduled such that they can be reclaimed if necessary. This can be achieved by building an array side by side with the schedule to keep the information about resources allocated for soft tasks. We adopted a simpler approach by maintaining two values for the minimum available time of each processor; the value of EAT_{jr} defines the earliest available time of processor P_j in regular mode, while the value EAT_{je} defines the earliest available time of processor P_j in emergency mode.

The difference between the two values defines the maximum spare time available on processor P_j . EAT_{jr} is always updated as follows: $EAT_{jr} = EAT_i + C_i$ for a selected task T_i . On the other hand, EAT_{je} is updated differently for hard and soft tasks: if the selected task T_i is hard, then $EAT_{je} = EAT_i + C_i$. EAT_{je} remains the same. In fact, the value of EAT_{je} tells the scheduler about the minimum available time of processor P_j if a specific soft tasks scheduled on it are to be dropped. This is true because the computation time allocated for these tasks on that processor is not included in this value. It is important to realize that task feasibility check is done based on EAT_{jr} in the regular mode.

The algorithm switches to emergency mode if at least one hard task in the feasibility check window is found infeasible (step 2). In this mode, task feasibility is checked based on the value of EAT_{je} of each processor. If all hard tasks in the feasibility check window are feasible (step 2a) we choose to extend the current partial schedule by the hard task with the minimum heuristic value on the processor P_j that provides the minimum

EAT_{je} while dropping any infeasible soft task in the window (step 2a-i). To accommodate the new hard task on the selected processor P_j we defer tasks already scheduled on that processor back to their start times to obtain the EAT_{je} of that processor (step 2a-ii). This requires the scheduler to drop one or more of the already scheduled soft tasks on that particular processor but not on any other processor. The worst situation occurs when at least one hard task is infeasible in the emergency mode (step 2b), in such case backtracking is required.

Emergency Algorithm ()

1. Tasks in the task queue are ordered in non-decreasing order of deadline.
2. if (all hard tasks in the feasibility check window are feasible) // Regular mode.
 - (a) Drop any infeasible soft tasks from the window.
 - (b) Extend the current partial schedule by the task with minimum heuristic value.
 - (c) Update the earliest available times (EAT_{jr} and EAT_{je}) of the selected processor P_j accordingly.
3. else // At least one hard task in the feasibility check window is infeasible.
 - (a) Switch to Emergency mode.
 - (b) if (all hard tasks are feasible)
 - i. Drop any infeasible soft tasks from the window.
 - ii. Defer tasks scheduled on processor P_j (the one with minimum emergency earliest available time EAT_{je}). This forces us to drop one or more soft tasks that already scheduled on P_j to make a room for the new selected hard task.
 - iii. Extend the current partial schedule by the hard task with minimum heuristic value on processor P_j .
 - iv. Update the earliest available times (EAT_{jr} and EAT_{je}) of the selected processor P_j accordingly.
 - (c) else // At least one hard task is infeasible.
 - i. Backtrack to the search level at which last hard task is scheduled.
 - ii. Extend the schedule with the hard task having the next best heuristic value.
4. Repeat steps (2- 4) until a feasible or hard feasible schedule is obtained.

It is to be noted that the correctness of emergency algorithm can be verified by following the algorithm steps one by one.

3.3 Complexity Analysis

It is easy to see that emergency algorithm requires $(n + m)$ steps to construct a schedule for n hard tasks and m soft tasks. In the worst case scenario, the algorithm will switch from regular mode to emergency mode each time it encounters a hard task. By recalling that the algorithm checks the feasibility and evaluates heuristic functions for at most k (the feasibility check window size) tasks in each mode at each step, the complexity of emergency algorithm can be expressed as $O(k(n + m)) + \text{number of backtracks}$. In this algorithm, we limit the number of backtracks to avoid exponential backtracking time.

A single invocation of myopic algorithm to schedule the same set of tasks takes $O(k(n + m)) + \text{backtracking time}$. Using myopic algorithm in the context of multilevel QoS

degradation algorithm requires i invocations of myopic algorithm to schedule the same task set. Therefore, such QoS degradation algorithm takes $O(ik(n+m)) + i(\text{number of backtracks})$, which is higher than that required by emergency algorithm whenever a QoS degradation is required.

4 Simulation Studies

In our simulation experiments, we compare the proposed emergency algorithm with the Iterative Server Rate Adjustment (ISRA) algorithm described in [6] based on the following two metrics:

1. The multimedia server utilization (MSU) defined as the amount of time allocated to multimedia server to the total amount of time requested originally by the multimedia server during the scheduling interval.
2. The scheduling overhead ratio (OHR) defined as the number of scheduling steps performed by emergency algorithm to that performed by ISRA algorithm to schedule the same task set.

In the following subsections we describe load generation process, and simulation results.

4.1 Load Generation

We adopted a method similar to that presented in [6] for load generation. To generate a set of hard real-time tasks, a two dimensional matrix with time as one dimension and processors as the other dimension is used to generate a task set in such way a schedule of tasks is created by arranging these tasks in the matrix one after another. A task is generated by selecting one of the processors with the earliest available time. The generated task is assigned a ready time equals to that earliest available time and a computation time chosen randomly between two input parameters (MinC, MaxC). Also, it is assigned a deadline that is chosen randomly between two other input parameters (MinD, MaxD). The earliest available time of the selected processor becomes equal to the finish time of the generated task. This process continues until the remaining unused time for each processor, up to L (the schedule length), is smaller than the minimum processing time of a task, which means no more tasks can be generated to use the processors.

Multimedia streams used in our experiments are generated according to the following parameters:

- Baseline multimedia computation time, C_b .
- Baseline multimedia period, P_b .
- Period increase ratio, r .
- Number of multimedia streams, m .

The baseline multimedia stream period P_b is the smallest period of all generated streams. The periods of other streams are made larger than P_b by multiples of the period increase ratio r . That is, for any stream S_i , the period $P_i = (1 + (i - 1)r)P_b$. Also, the baseline multimedia stream is assigned a computation time equal to C_b , while

other streams are assigned computation time larger than C_b by multiples of the period increase ratio r . That is, for any stream S_i , the computation time $C_i = (1 + (i - 1)r)C_b$. By this method, we can generate any number of streams with different periods and different computation times, such that all streams are related to one baseline stream. This enables us to study the effect of multimedia load by only changing the parameters of the baseline stream.

Load generation was done according to the default parameters shown in Table 4.1 unless otherwise specified, where H-Parameters refers to hard tasks generation parameters, and S-Parameters refers to soft tasks (i.e., multimedia streams) generation parameters. Each of the results presented in the following subsection represents the average of four simulation runs. Each simulation run involves 400 experiments (i.e., 400 different tasks sets were generated for each run). Throughout our experiments we used a system composed of 3 processors and we fixed the schedule length parameter (L) to 400. This corresponds to an average number of 60 hard tasks in each task set and about 13 multimedia server instances. Therefore, an average of 73 tasks were generated for each experiment.

Table 4.1. Simulation parameters

<i>H - Parameters</i>	<i>Value</i>	<i>S - Parameters</i>	<i>Value</i>
MinC	10	P_b	33
MaxC	30	C_b	2.5
MinD	60	r	0.2
MaxD	90	m	5

4.2 Simulation Results

Effect of Baseline Multimedia Computation Time: Fig.1 shows the effect of the baseline multimedia computation time, which reflects the load imposed by the multimedia streams on the system, on the multimedia server utilization (MSU) as well as on overhead ratio (OHR) for the same set of experiments. As expected, the value of MSU keeps decreasing by increasing C_b . When C_b is less than 1.5, both algorithms offer an MSU greater than 0.8. But after that, MSU drops to about 0.5. This can be explained by recalling that the main objective of both algorithms is to guarantee the schedulability of hard tasks. As the load of multimedia streams increases (represented by increasing C_b) more service degradation is required and therefore the fraction of time allocated to multimedia streams keeps decreasing.

At the same time, OHR curve suggests that the two algorithms incur almost the same amount of scheduling overhead when the multimedia load is light. However, by increasing the multimedia load, OHR decreases substantially because the number of rescheduling attempts made by ISRA increases to achieve the necessary QoS degradation, which emphasizes that emergency algorithm has much lower scheduling overhead than ISRA.

Effect of Average Worst Case Execution Time of Hard Real-Time Tasks: Fig.2 illustrates the effect of changing parameter (MaxC) from 20 to 40 in steps of 10, while fixing MinC to 10. By increasing MaxC, the worst case computation time of the generated

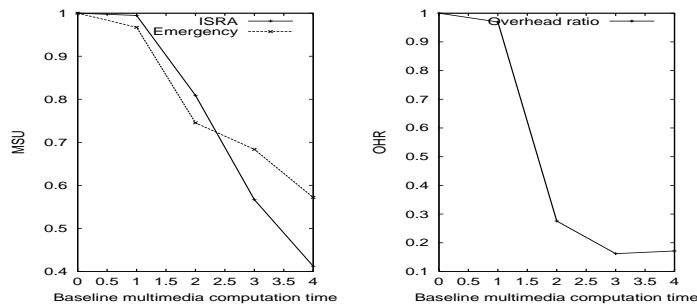


Fig. 1. Effect of baseline multimedia computation time on MSU and OHR

hard tasks increases as well. Therefore, it becomes more difficult to fit such hard tasks in the schedule without degrading the service of soft tasks either by dropping some of them as in emergency algorithm or by iterative adjustments of the server utilization as in ISRA. This explains the continuous decrease of MSU observed by both algorithms. However, the rate of decrease obtained by ISRA is higher than that obtained by emergency algorithm. This supports our theory about inherent QoS degradation by selective task dropping. For the same experiments, OHR remains almost constant.

For the same experiments, OHR remains almost constant. This can be interpreted by looking at the results of previous experiment when C_b was equal to 2.5. OHR value reported at that point is about 0.2 which is almost the same value obtained in this experiment. This observation suggests that rate of increase of scheduling overhead of both algorithms is relatively the same, and because of that OHR remains constant.

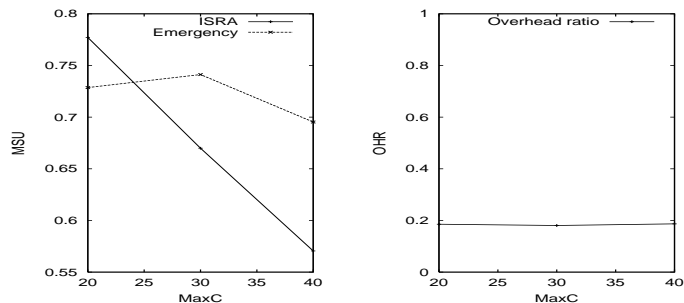


Fig. 2. Effect of average worst case execution time of hard real-time tasks on MSU and OHR

Effect of Average Laxity of Hard Real-Time Tasks: The effect of laxity (the maximum amount of time a task can wait before it can start its execution) of hard tasks can be captured by adjusting MinD and MaxD. Fig.3 shows our results for this experiment. Increasing MinD results in implicit increase in the laxity of hard tasks, and therefore, such tasks become more relaxed. As a result, soft tasks can be given more chances. This explains the continuous increase of MSU by both algorithms. For the same reasons explained in the previous experiment, OHR remains almost constant (about 0.2) in this experiment which also means that more scheduling steps are required by ISRA than required by emergency algorithm.

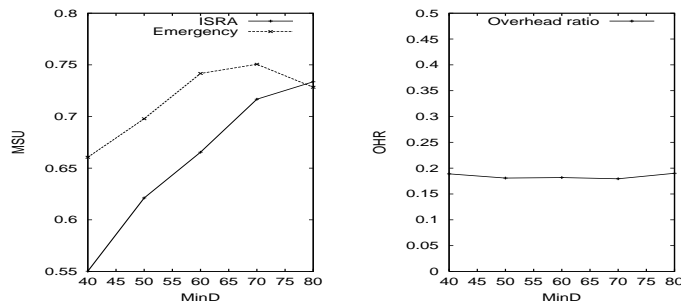


Fig. 3. Effect of average laxity of hard real-time tasks on MSU and OHR

5 Conclusions

In this paper, we addressed the issue of combined scheduling of hard and soft real-time tasks in multiprocessor systems with a primary objective of maximizing the schedulability of soft tasks without jeopardizing the schedulability of hard tasks. For this objective, we proposed the emergency algorithm which has the inherent feature of QoS degradation of soft tasks. This feature removes the burden of repetitive QoS degradation and rescheduling, and therefore, avoids high scheduling overhead. Our algorithm was evaluated by comparing it to Iterative Server Rate Adjustment (ISRA) algorithm proposed in [6]. Simulation results show that the emergency algorithm offers higher schedulability for soft tasks, in most cases, and has much lower scheduling overhead compared to ISRA. Future work includes extending the proposed algorithm to allow task migration and preemption such that higher schedulability can be obtained.

References

1. G. Buttazzo and F. Sensini. Optimal Deadline Assignment for Scheduling Soft Aperiodic Tasks in Hard Real-Time Environments. *IEEE Transactions on Computers*, vol. 48, no. 10, pp. 1035-1051, October 1999.
2. R. Davis, K. W. Tindell, and A. Burns. Scheduling Slack Time in Fixed Priority Pre-emptive Systems. *Proceedings of the 14th Real Time System Symposium*, pp. 222-231, North Carolina, USA, December 1993.
3. R. Davis and A. Wellings. Dual Priority Scheduling. *Proceedings of Real-Time Systems Symposium*, pp. 100 - 109, 1995.
4. Z. Deng, J. W. S. Liu, and J. Sun. A Scheme for Scheduling Hard Real-Time Applications in Open System Environment. *In Proceedings of the 9th Euromicro Workshop on Real-Time Systems*, 1997.
5. T.M. Ghazalie and T. Baker. Aperiodic Servers in a Deadline Scheduling Environment. *Real-Time Systems*, 9, 1995.
6. H. Kaneko, J.A. Stankovic, S. Sen, and K. Ramamritham. Integrated Scheduling of Multimedia and Hard Real-Time Tasks. *Proceedings of Real-Time Systems Symposium*, pp. 206-217, 1996.
7. S. Lee, H. Kim and J. Lee. A Soft Aperiodic Task Scheduling in Dynamic-Priority Systems. *Proceedings of the 2nd International Workshop on Real-Time Computing Systems and Applications*, IEEE 1995.
8. J.P. Lehoczky, L. Sha, and J.k. Strosnider. Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. *Proceedings of Real-Time Systems Symposium*, pp. 261-270, 1987.
9. K. Ramamritham, J.A. Stankovic, and P.-F. Shian. Efficient Scheduling Algorithm for Real-Time Multiprocessor systems *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 2, pp. 184-194, April 1990.
10. M. Spuri and G.C. Buttazzo. Efficient Aperiodic Service under Earliest Deadline Scheduling. *Proceedings of IEEE Real-Time Systems Symposium*, San Juan, Puerto Rico, December 1994.
11. M. Spuri and G.C. Buttazzo. Scheduling Aperiodic Tasks in Dynamic Priority Systems. *Real-Time Systems*, vol. 10, no. 2, 1996.
12. B. Sprunt, J. Lehoczky, and L. Sha. Exploiting Unused Periodic Time for Aperiodic Service Using the Extended Priority Exchange Algorithm. *Proceedings of the 9th IEEE Real-Time Systems Symposium*, December 1988.