Improving Control Performance using Adaptive Quality of Service in a Real-Time System

Technical Report (UCSC-CRL-04-04)

Caixue Lin, Pau Martí, Scott A. Brandt, Scott Banachowski
Computer Science Department
University of California
Santa Cruz, CA, USA
{lcx,pmarti,sbrandt,sbanacho}@cs.ucsc.edu

Manel Velasco and Josep M. Fuertes
Automatic Control Department
Universitat Politècnica de Cataluña
Barcelona, Spain
{manel.velasco,josep.m.fuertes}@upc.es

Abstract

Traditional control systems employ fixed sampling intervals. Recent work in integrated control and real-time systems has resulted in control systems in which the sampling interval may vary based on the state of controller performance. Soft real-time systems provide mechanisms for dynamically adapting application resource usage based on system state and application needs. In this paper we investigate employing this mechanism to allow several control applications to dynamically adapt their resource usage so that they receive enough of the limited resources to achieve their goals, but do not greedily consume resources, allowing the system to be utilized by other applications as well. This paper presents a framework in which a flexible, integrated real-time system directly supports adaptive control applications. Our results show that this technique can result in significantly lower controller error (by an average of over 20% in our experiments) with no increase in overall resource usage.

Keywords: real-time systems, control theory, optimization, quality of service

1. Introduction

Control systems and hard real-time systems have evolved hand-in-hand. Control systems sample the world and provide signal outputs at fixed intervals, while hard real-time systems provide the timeliness guarantees necessary to complete these operations on time. However, providing hard guarantees comes with a price—in order to make guarantees, the system must dedicate resources to applications based on their worst-case processing constraints, even though their average-case resource needs may be far lower than the worst-case. This typically corresponds to poor system utilization and/or increased power consumption.

On the other hand, if resources are allocated to tasks based on their average requirements, the system may become overloaded when a task exceeds its average processing requirements and deadlines may be missed. Recent work in soft real-time addresses this problem by providing applications the opportunity to adapt to available system resources [2, 12, 14]. When processes do not meet enough deadlines to provide adequate Quality of Service (QoS), they adapt by changing their processing requirements, either continuously or among a set of predetermined levels. A global resource allocation mech-

anism sets all application levels such that overall, the combination of active levels yields the highest overall benefit to the user within the available resources.

Recently, researchers have examined ways to provide similar soft functionality to control applications by allowing the applications to change their sampling intervals, along with their control law, on the fly [10]. The reasoning for this is analogous to the worst-case execution time example for traditional real-time systems: in the case where the system is stable, the control application need not monitor and control the plant as aggressively, and by reducing the sampling interval, the task consumes less resources than the worse-case. Generally, smaller sampling intervals consume more resources and yield better control, while larger sampling intervals consume less resources but may yield larger errors.

Mirroring the continued evolution of control and real-time systems, this paper examines using an adaptive Quality of Service (QoS) framework to support these new adaptive control applications. In an adaptive QoS framework, the system (and corresponding scheduler) directly supports applications that are designed to provide different modes of operation for delivering varying levels of quality. One of the difficult problems facing previous adaptive soft real-time scheduling algorithms is determining the desired outputs, because they must be based on relative qualities produced among several applications. Functions describing quality are often assigned, arbitrarily or through trial-and-error, by the user. In the control case, this problem is solved because the control task exhibiting the most stability is the likely candidate to reduce its resource consumption.

The contributions of this work are the implementation of an adaptive QoS soft real-time scheduling framework in an integrated real-time scheduler, and the application of this framework to the problem of supporting adaptive control applications. The novel aspects of the design include the elimination of static benefit functions associated with different levels of QoS in favor of dynamic benefits based on a novel function of instantaneous controller error. Overall, we show scenarios in which this technique reduces the error of an individual controller by as much as 40% and reduces the average error of all concurrently executing controllers by about 22%.

Section 2 first discusses the implementation of the Adaptive QoS system in the Rate-Based Earliest Deadline (RBED) scheduler, which provides a general framework for supporting adaptive QoS applications in a multi-purpose, real-time scheduler. Section 3 introduces the concepts of adaptive control applications, and Section 4 describes how the scheduling framework was adapted to support these applications. The remaining sections provide some performance results, related works, and conclusions.

2. DQM Soft Real-Time Processing in RBED

We implemented the Adaptive QoS system in the Rate-Based Earliest Deadline (RBED) Linux system [4]. RBED provides fully integrated scheduling of hard real-time, soft real-time, and best-effort processes. RBED is an instance of the resource allocation/dispatching integrated real-time scheduling model in which applica-

tion resource allocations and dispatching are managed separately. RBED allocates resources to processes as a percentage of the CPU such that the total allocated is less than or equal to 100%, then schedules all processes with the earliest deadline first (EDF) algorithm. Unlike traditional EDF implementations, RBED dynamically adjusts both the utilizations and periods of applications so that it flexibly supports several flavors of real-time, soft real-time and best-effort processing. Changes to process resource allocations are made without violating EDF constraints [4], so a task never misses its assigned deadlines—this allows it to execute a mix of hard real-time applications with sets of other tasks in a dynamic, integrated environment.

To support Adaptive QoS applications, we have implemented a modified version of the DQM QoS Level soft real-time system [2] in the RBED system. QoS Levels allow discrete application adaptation. Each application provides to the system a table specifying the discrete *levels* at which it can operate, the relative amount of resources required to run at each level, and the relative *benefit* of running at each level. Each level corresponds to a particular algorithm appropriate for meeting the application goals using a different amount of resources. For example algorithms may change their sampling interval, frame rate, bit rate, display size, compression algorithm, etc. The resulting different levels each consume different amounts of resources and provide different output quality (described as the benefit). The relative benefit of each level is a static quantity reflecting the quality of the output at each level relative to the quality at the highest level—i.e. the maximum quality of the application.

In traditional soft real-time systems, the overall benefit of an application is a static quantity based on the user's needs and is specified at job admission time. Determining this value is problematic and the nearest analog, priority of best-effort applications, is almost always set to the default value. When running adaptive control applications in this system, the benefit of the application is dynamically determined by the *instantaneous error* of the application (discussed further below). Unlike a static benefit specification, instantaneous error provides a specification of the current criticality of each application and thus allows the system to dynamically optimize the resource allocations to the needs of the applications.

2.1. RBED

RBED allocates resources to processes as a percentage of the CPU such that the total allocated to all processes is less than or equal to 100%. Hard real-time processes have period p and worst-case execution time e and are either granted their desired rate (e/p) or rejected if resources are not available. Soft real-time processes receive their desired rate (e/p) if possible, or are assigned a lesser rate, possibly based upon a QoS specification if one is available. The rate of each best-effort process is determined from the remaining resources after the rates of the other processes in the system are set. A reservation mechanism guarantees that a minimum or maximum allocation is available to any particular class of processes, ensuring, for example, that there are always some resources available to the best-effort processes.

By allocating the resources appropriately and choosing appropriate deadlines, RBED presents a feasible workload to EDF, guaranteeing that all applications will receive the correct amount of CPU, on time. However, unlike processes in traditional hard real-time systems, the rates of soft real-time and best-effort processes may change as processes enter and leave the system, and the periods of soft real-time processes may change as they adjust to the available resources.

2.2. The DQM QoS Level Soft Real-Time Processing Model

In the Dynamic QoS Manager (DQM) system, soft real-time support is provided to a community of cooperating applications by incorporating a QoS manager that optimizes resource allocation according to the global benefit of each application to the community within the currently available resources. Each application provides the QoS manager with a model of its application benefit, processing time, and period. Applications that provide relatively high benefit to the user receive correspondingly more resources than ones that provide lower benefit at that same instant. In addition, by allowing applications to determine which real-time processing constraints to change as they change their resource usage, QoS Levels effectively separate soft real-time policy from soft real-time mechanism—the system provides the soft real-time mechanism and the applications themselves define their own adaptive soft real-time policies.

The Dynamic QoS Manager (DQM) was originally developed as a middleware mechanism that operates on the collective QoS Level specifications. It analyzes the optimization functions provided by the community of processes to determine its allocation strategy. Once the DQM determines how resources should be allocated, it sets the level at which each application operates in order to optimize the global benefit. In the original DQM middleware solution, each application is informed of the level at which it should execute to maximize global benefit, but the middleware manager does not ensure that the application will actually execute at the recommended level, nor that the level will consume the resources that it is supposed to consume. Thus the DQM requires that the applications cooperate to achieve maximized behavior. An advantage of our RBED implementation is that this is no longer required—RBED forces applications to run within their resource allocations so that the the DQM model is guaranteed to work correctly in a mixed environment or in the presence of misbehaving applications.

2.3. Implementing DQM in RBED

Because the RBED system allows run-time flexibility in task utilization and period, it is well-suited for running DQM-style QoS Level adaptive soft real-time processes. The DQM middleware manager is executed as part of the RBED run-time system and QoS Level application utilizations and periods are adjusted as necessary based on a global calculation taking into account both available resources and application QoS Level specifications. Adjustments are made whenever available resources change (due to other applications entering or leaving the system), application processing requirements change (due to a mode change or other application-specific processing changes), or when ben-

efits change (in this case, due to changes in the error of one or more controllers). RBED mode-change theory guarantees that these changes do not interfere with the processing of other applications.

To demonstrate the performance of the DQM as implemented in RBED, we evaluated the performance of several QoS Level SRT applications. Each application has a table describing the Quality of Service (QoS) levels it provides (shown in Table 1), each level corresponds to an algorithm, and each algorithm has a different set of real-time constraints and benefit to the user when executing at that level. The first level represents the highest resource usage, and provides the maximum benefit. A kernel task monitors the utilization of the system and sets QoS levels for these applications, attempting to maximize the global benefit density (the benefit to resource use ratio $\frac{benefit}{resource\ rate}$) [2].

Figure 1 shows the simulated scheduling behavior when dynamically changing QoS levels of the soft real-time processes in the RBED algorithm. Initially, a hard real-time process uses 60% of the resource, and executes for 46.2 seconds. The initial resource rates of soft real-time processes are set to their lowest QoS levels (10%,10%, and 10%, respectively). When the hard real-time process leaves, the levels of the three soft real-time processes are adjusted to provide a higher level of benefit within the available resources. The heuristic resource allocation algorithm iteratively increases the level of the tasks until no more increases are possible within the available resources. It does this by always choosing the task whose level increase provides the greatest increase in benefit density, i.e. the one with the greatest $\frac{\Delta benefit}{\Delta resource\ usage}$. Similarly, when lowering resources it always chooses the level whose removal decreases overall benefit density the least. In this case, the result is that SRT-1 increases to level 1, SRT-2 increases to level 1, and SRT-3 remains at level 4.

Although the heuristic algorithm found the resource allocation that provides the highest possible benefit, this is not always guaranteed nor is it always possible. The discrete QoS Level approach is in general NP-complete (by a straightforward reduction to the Knapsack Problem [7]). It is therefore not generally possible to calculate an optimal resource allocation that maximizes benefit for a given set of process. QoS Levels managed with heuristic resource allocation algorithms that come close to the optimal solution are an appropriate platform for implementing soft control applications. Furthermore, the benefit density heuristic performs quite well in practice and has the nice property that application level changes are monotonic in regards to the change in available resources, a property that is not present in the optimal solution [3].

3. Adaptive Control

Control systems deal with meeting system specifications for performance and stability. Performance is usually specified in terms of the controlled system response (both transient and steady-state), for example optimizing cost functions or meeting power consumption or response time requirements. Stability implies that the controlled system must achieve the expected results regardless of how they are reached, i.e. during the system lifetime the controlled system must not crash.

Table 1: Sample Benefit Tables (SRT-1, SRT-2 and SRT-3 in Figure 1)

Number of QoS Levels: 4			
Level	Benefit	Rate	Period
1	**1.0**	**0.35**	**100 ms**
2	0.7	0.30	100 ms
3	0.5	0.20	100 ms
4	0.3	0.10	100 ms

(a) SRT-1

Number of QoS Levels: 4			
Level	Benefit	Rate	Period
1	**1.0**	**0.45**	**100 ms**
2	0.8	0.40	100 ms
3	0.6	0.30	100 ms
4	0.4	0.10	100 ms

(b) SRT-2

Number of QoS Levels: 4			
Level	Benefit	Rate	Period
1	1.0	0.60	100 ms
2	0.9	0.50	100 ms
3	0.7	0.40	100 ms
4	**0.5**	**0.10**	**100 ms**

(c) SRT-3

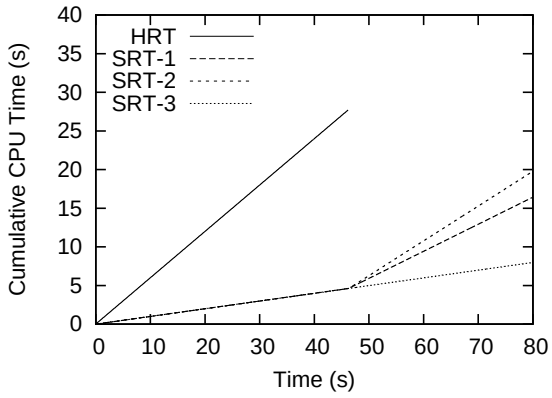


Figure 1: SRT QoS Management in RBED.

Traditional control specifications are used to obtain static controllers; given a set of different control application scenarios, a controller is designed with enough robustness to cope with all of them while achieving the desired application performance. A traditional controller is designed to meet the stability and performance specifications for any application scenario, allowing the same static controller to execute regardless of changes in the environment. It is important to stress that for digital controllers, the choice of the sampling period will determine if its requirements may be met. Once the sampling period is established, the controller resource requirements are set and the controller task will demand a constant CPU load.

Although this traditional way of designing control applications works fine, better approaches can be used in dynamic environments. When using a static controller we obtain the same average performance (e.g., benefit) for all application scenarios. However, if we execute specific controllers for each application scenario, we maximize the overall benefit by executing each controller according to the application dynamics [17]. Moreover, static controllers force a static CPU allocation to each control task although for specific application scenarios (e.g., when the system is in a safe state), the allocated resources may be reduced without losing the specified performance [16].

Flexible control strategies can be supported by a framework that is able to dynamically allocate computing resources according to the control application dynamics. Instead of designing one single controller for each control task to cope with all application scenarios,

we design different controllers for different scenarios, each one delivering a specific performance level (benefit) and demanding a specific CPU share (utilization). Note that each controller's benefit relates to control performance in the sense that higher execution rates yield better control performance. Therefore, it is desirable to execute each task with the highest rate controller. However, this is not always feasible due to the limited availability of computing resources. In a situation with several tasks, each one with few candidate controllers, we must choose for each task the appropriate controller such that the overall benefit is maximized taking into account resource availability.

Although a maximum benefit may be achieved by several combinations of levels for the set of control tasks, some of these combinations may be more appropriate than others for the current application dynamics. To illustrate the problem, suppose there are two control tasks, each of which may use one of two equal (in terms of benefit and rate) controllers, one of a higher rate and one lower. Because the amount of computing resources is limited, the system cannot run both controllers at their highest rate simultaneously. Therefore, the only choice is to choose the higher rate controller for one task and the lower rate controller for the other, or vice-versa. In terms of static benefits, both choices are equal. Nevertheless, it may happen that one controlled system is in equilibrium and the other isn't. In this case, the best choice is for the task of the system in equilibrium to use the lower rate controller and the other task to use the higher rate controller. To make this possible, we need to have feedback information from the controlled plants available to the system.

To solve the problem of choosing the appropriate combination of controller levels, we allow the system to obtain feedback information from the controlled plants. With this information the system is able to re-scale the benefits associated to each controller for all control tasks. Thus, the maximum benefit will be obtained not only in terms of system resource but also in terms of current application demands.

4. Implementing Adaptive Control on DQM

In order to integrate into the DQM framework the management of the control applications we must first define how to associate benefits to each controller and then what information to feedback from the application. As explained in the previous section, we have configured each control task with a set of candidate controllers to choose from. In previous QoS Level work, the association of benefits to each candidate is made according to any policy which most benefits the user, for example as a function of power consumption, CPU load, communication bandwidth, execution rate, etc. In this control implementation, we assign relative benefits reflecting the execution rate, and thus controller performance: the higher the rate, the greater the relative benefit.

The relative importance of each controller will be determined by the performance of that controller. Because the benefit specifications consist of a set of levels with relative benefit values, a set of dynamic benefits is achieved by rescaling the base benefit of the relative benefits. However, the feedback information required by

the system to re-scale the benefits may be application-dependent. Looking at control applications, the choice is still wide open—control theory provides metrics for measuring control performance that are used in the analysis and design of controllers. However these metrics are not defined for run-time measurement of control performance.

For this study, we chose to model the inverted pendulum problem as our control application. Its discrete-time control system model is described by $x(k+1) = \mathbf{A}x(k) + \mathbf{B}u(k)$, where x is the state vector, $\mathbf{A}$ is a matrix describing the system dynamics, and $\mathbf{B}$ is an input matrix that relates the system with the control signal u, computed by the controller at each execution. We can use its components or any combination of them to obtain the feedback information. It is important to note that any choice may be good enough depending on the desired results.

In a pendulum model (Figure 2), we choose four independent state variables ($x = (s_1, s_2, s_3, s_4)$) for a control task (pendulum) :

- s_1: The angle of the pendulum (radians)
- s_2: The variation of the pendulum angle, that is the angular velocity of the pendulum (radians/second)
- s_3: The cart position (meters)
- s_4: The variation of the cart position, that is the velocity of the cart (meters/second)

We investigated a variety of possible instantaneous error measures, based on elements of the state vector, for use as the control error feedback information provided to the scheduler. We ultimately chose the one that gave the best performance. The best performing error function is the normalized value of the state vector ($e = \sqrt{s_1^2 + s_2^2 + s_3^2 + s_4^2}$). This function captures all of the control system dynamics and was also desirable because, unlike some other error functions, it is monotonically non-increasing over time in response to a single step input. Figure 3 shows the angle of the pendulum as the system reacts to a single step perturbation, for each the four controllers we configured for this study.

Based on the feedback information (i.e. the control errors) from the control tasks, the system re-scales the benefits, and from these dynamic benefit values the best controller per task is chosen. In order to add hysteresis and prevent too frequent adjustment, we smooth and adjust the highly varying control errors. A heuristic error quantization function is used to quantize the values of the control errors into four levels:

$$\text{quantized error} = \lceil 2 \cdot e^{\frac{1}{4}} \rceil$$

The decision to use four levels is based on the maximum value of the measured control errors and the number of available sampling periods of the controller in our experiments.

The quantized error is used as the value that dynamically scales the original benefits of the control tasks. As a result, the new dynamic benefit of a control task is calculated as follows:

$$\text{dynamic benefit} = \text{static benefit} \times \text{quantized error}$$

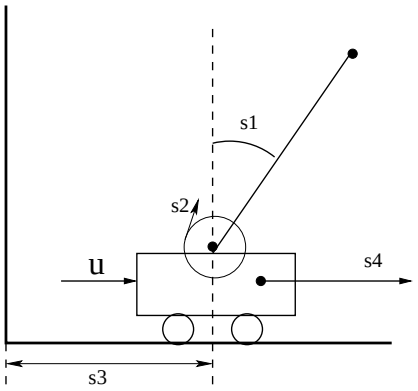


Figure 2: Inverted Pendulum

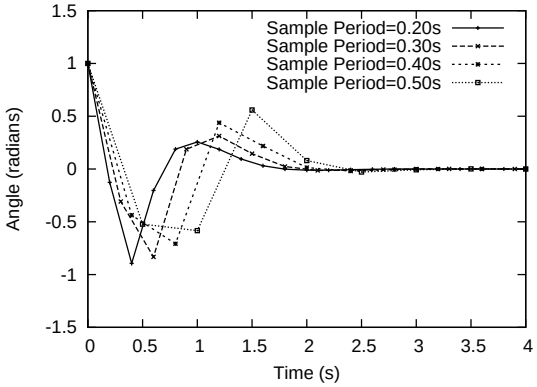


Figure 3: Sampled Control Errors in terms of Pendulum Angle

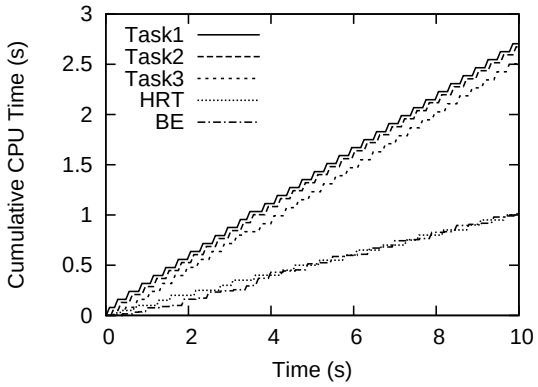
5. Results

To show the performance achieved using dynamic period adjustment, we simulated the control tasks, and ran them in the RBED scheduler. The simulated control tasks were configured with the QoS Levels described in Table 2. During the experiments, we also ran a hard real-time task and best-effort tasks. In RBED, the system always reserves a minimum of 5% of the CPU to the set of best-effort processes, enough to provide a functional interactive system for running command shells and other tools while real time applications are executing.

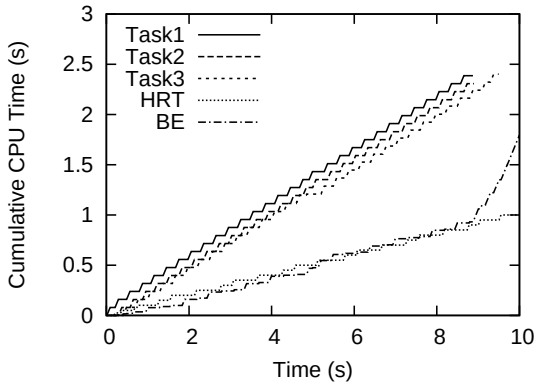
Table 2: Benefit tables for the control tasks

Number of QoS Levels: 4			
Level	Benefit	% CPU	Period
1	0.80	40%	0.2 s
2	0.56	27%	0.3 s
3	0.40	20%	0.4 s
4	0.32	16%	0.5 s

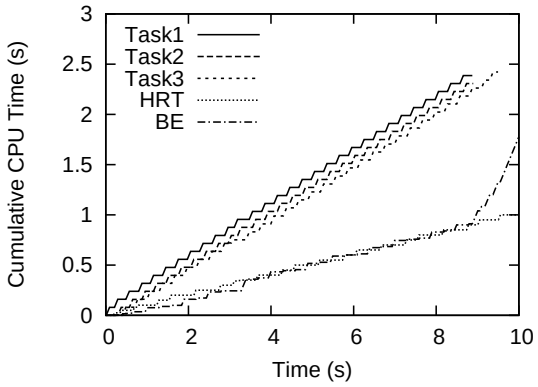
We evaluate the performance of the system by comparing the performance of the control tasks (i.e. pendulums) with and without dynamic scaled adaptation. In the non-adaptive case, benefits are static so the period chosen by the controller is determined statically when they enter the system, and does not change throughout the execution. With adaptation, the benefit of each QoS level is scaled with the dynamic control error.



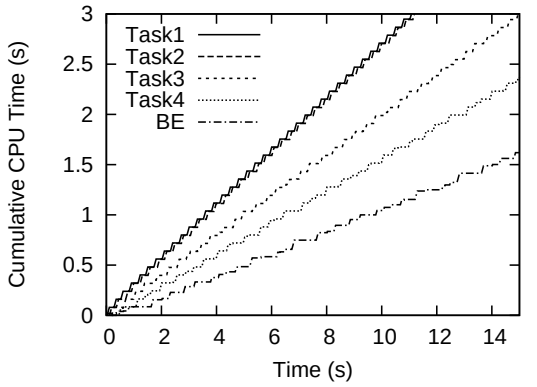
(a) Scenario 1 (three tasks not overlapped)



(b) Scenario 2 (three tasks partially overlapped)



(a) Scenario 3 (three tasks fully overlapped)



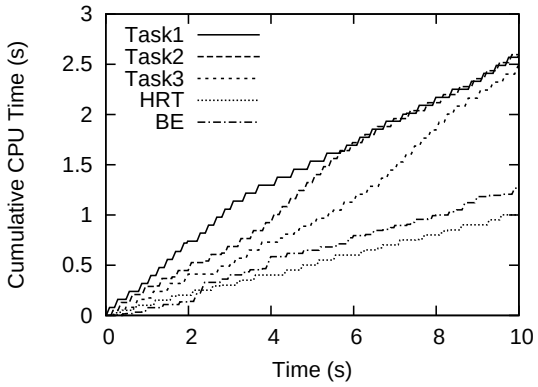
(b) Scenario 4 (four tasks not overlapped)

Figure 4: Cumulative CPU Time without Adaptation

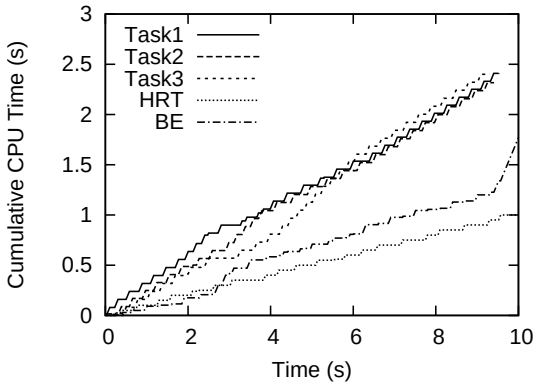
This means as the controller executes its benefit will be scaled, possibly triggering the QoS adaption mechanism to alter the sampling periods of the controller tasks.

To evaluate the performance of the system, we consider multiple (three or four) control tasks (inverted pendulums). A hard real-time task HRT with period of 0.5 seconds and execution time of 0.05 seconds (10% CPU bandwidth) and a CPU bound best-effort task BE are also running simultaneously with the control tasks in most of our experiments. The control tasks start running at the same time and initially are in stable states, which means their control errors are zero. We trigger error for each task at different times in the different scenarios. Four scenarios are evaluated in our experiments:

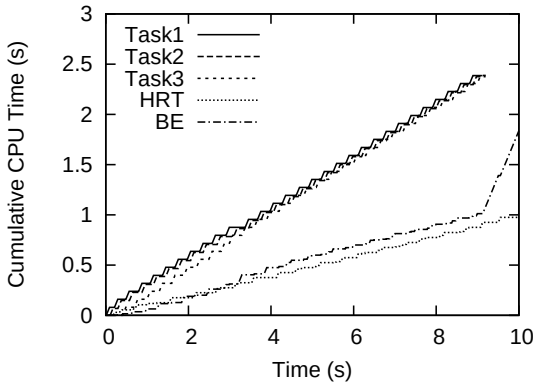
- Scenario 1 (3 control tasks + HRT + BE): An error (i.e. perterbation) is triggered at different times for each control task with large enough gaps that only one task has controller error at any one time.
- Scenario 2 (3 control tasks + HRT + BE): An error is triggered at different times for each control task with small enough gaps that multiple controllers have error at the same time.
- Scenario 3 (3 control tasks + HRT + BE): An error is triggered simultaneously for all control tasks.
- Scenario 4 (4 control tasks + BE): An error is triggered at different times for each control task with



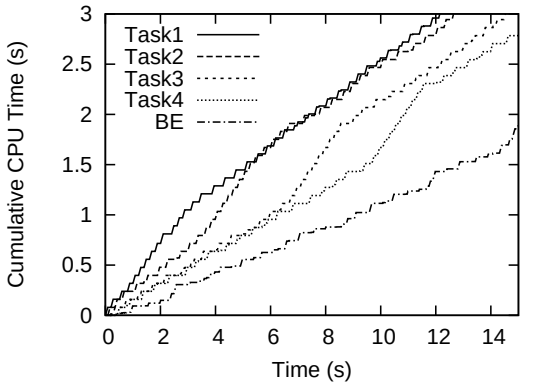
(a) Scenario 1 (three tasks not overlapped)



(b) Scenario 2 (three tasks partially overlapped)



(a) Scenario 3 (three tasks fully overlapped)



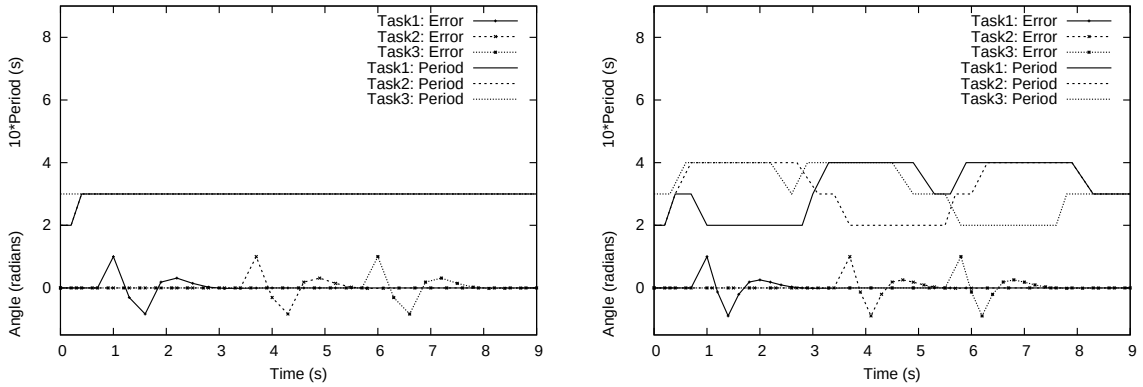
(b) Scenario 4 (four tasks not overlapped)

Figure 5: Cumulative CPU Time with Adaptation

large enough gaps that only one task has controller error at any one time.

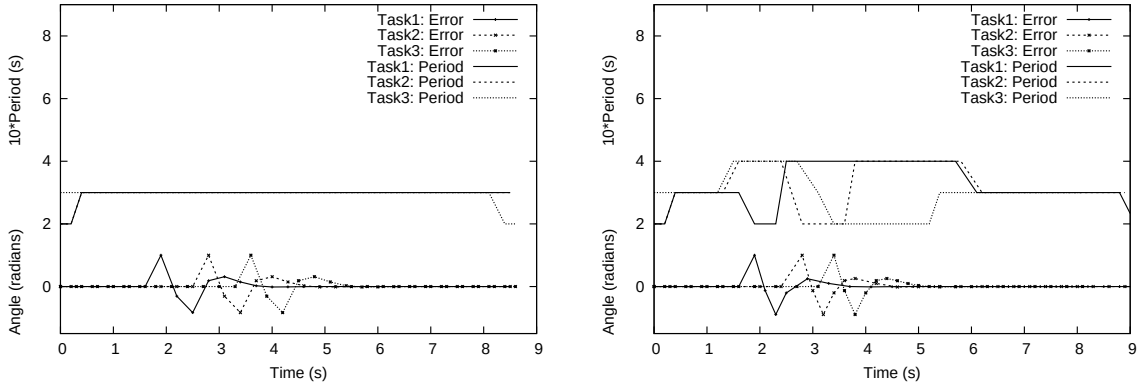
Figure 4 shows the cumulative CPU time received by each task in each of the four scenarios without adaptation. The hard real-time process receives its required resources, unaffected by the presence of the soft real-time or best-effort processes. The best-effort process receives at least the minimum reserved resources (5%) of the CPU, or 10% in Scenarios 1, 2 and 3 because it also uses the slack time left by the hard real-time and the three soft real-time processes. Without adaptation, the resource usage of the three soft real-time processes is constant and is based on a benefit optimization using each process's static relative benefit. The rough slope of each of the SRT (control task) lines is the same in Scenarios 1 through 3, reflecting the fact that they each received the same amount of resources throughout each experiment. In Scenario 4 the resource usage of the four SRT processes are different, reflecting the fact that no all could run at the same level given the available resources. Finally, the slope of the BE line increases rapidly at the end of each experiment (showed in Scenarios 2 and 3) as the SRT processes exit the system and their resources are given to it.

Figure 5 shows the cumulative CPU time received by each task in each of the four scenarios with adap-



(a) Without Adaptation (b) With Adaptation

Figure 6: Scenario 1 (three tasks not overlapped)

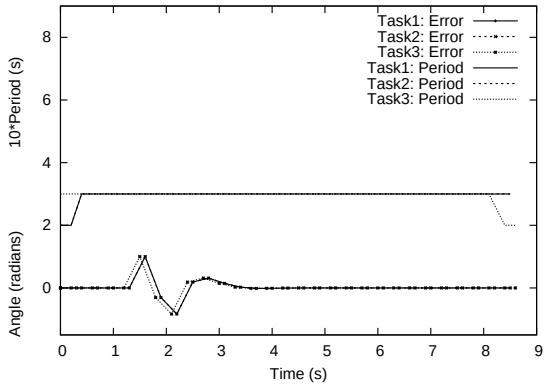


(a) Without Adaptation (b) With Adaptation

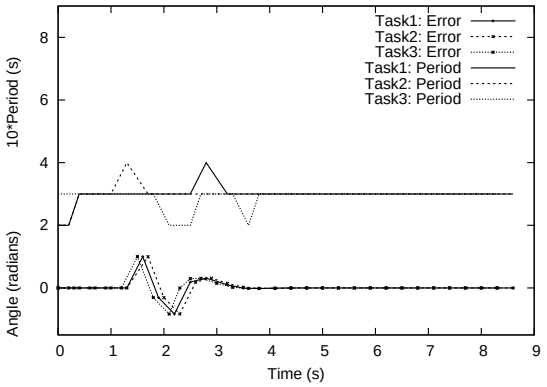
Figure 7: Scenario 2 (three tasks partially overlapped)

tation (in which benefit is scaled based on the current error of the controller, as described above). In contrast to the non-adaptive case, we see that the progress of each of the tasks in Scenarios 1, 2 and 4 varies at different times as the system adjusts the resource allocations to reflect the current state of the applications. The progress of each task in Scenario 3 remain relatively constant. This is due to the fact that the perturbations and resulting error are almost completely overlapping. As a result, no additional benefit can be obtained by shifting resources from one application to another, and so all are left at roughly the same level throughout the execution. In addition, the best-effort process receives more than 12% of the CPU in all the Scenarios. This is because the control tasks consume less resources campared to the non-adaptation case.

Figures 6 through 9 show the resulting performance of the control tasks in these scenarios. Figure 6(a) and Figure 6(b) show the performance of the control tasks in Scenario 1 without and with adaptation. In Figure 6(a) the system executes each controller throughout the experiment at a level determined by its static benefit specification, mirroring the stable CPU allocations seen in Figure 4(a). This results in an overall controller error

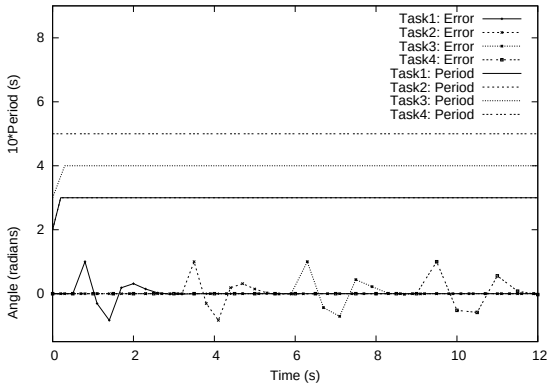


(a) Without Adaptation

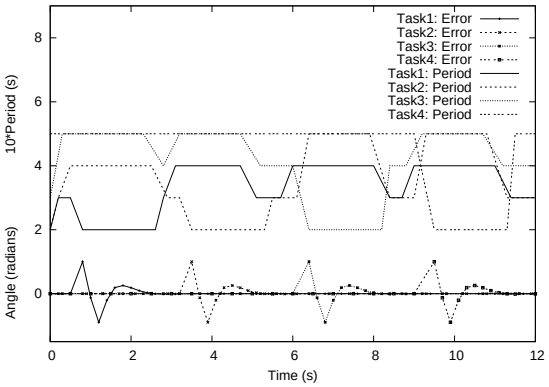


(b) With Adaptation

Figure 8: Scenario 3 (three tasks fully overlapped)



(a) Without Adaptation



(b) With Adaptation

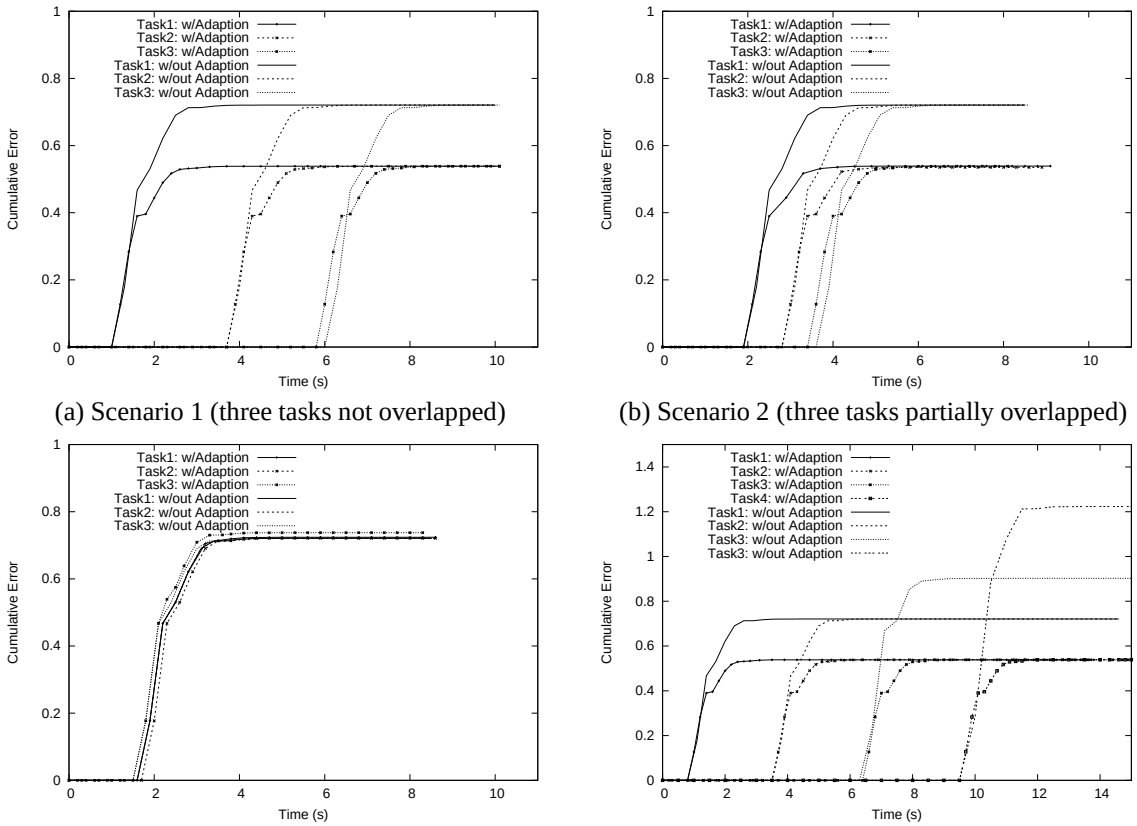
Figure 9: Scenario 4 (four tasks not overlapped)

of 2.16 (calculated as the integral of the error curves as showed in 3).

In Figure 6(b) the system dynamically reallocates the system resources to allow the controller with the greatest error to run at a higher sampling frequency while lowering the sampling frequency of the controller with the least error. As the perturbations and resulting error for the three applications are sufficiently far apart, there is only one application with any error at one time and this application is always allowed to run at it's highest level while it is responding to the perturbation. This reflects the varying cumulative CPU allocations seen in Figure 5(a). This results in an overall controller error of 1.61, 25.3% less than without adaptation. The three processes with adaptation decrease their error faster than those without adaptation and thus incur less total error.

The results for Scenario 2 (Figure 7) are similar to those for Scenario 1 except that the partial overlap of the errors requires greater sharing of the resources. Nevertheless, the controller error numbers are almost identical to those of Scenario 1, with total error of 2.16 without adaptation and 1.61 with adaptation for a 25.3% reduction overall.

In Scenario 3 (Figure 8), the overlap of the errors is



(a) Scenario 1 (three tasks not overlapped)

(b) Scenario 2 (three tasks partially overlapped)

(c) Scenario 3 (three tasks fully overlapped)

(d) Scenario 4 (four tasks not overlapped)

Figure 10: Performance Evaluation in terms of Cumulative Error

nearly complete and no benefit can be achieved by reallocating the resources. Thus each controller continues to run mostly at its original level and there is little difference between the two cases, with and without adaptation. In fact, the total error rises very slightly (by 1%) with adaptation, from 2.16 to 2.18. This slight reduction in performance is the result of some adaptations occurring at the wrong times (i.e. just before a perturbation). Specifically, the level of SRT2 (task2) is lowered just before a perturbation. This slightly longer sampling period increases the response time of the system slightly and increases the error for this task, but it is quickly corrected as soon as the increased error is detected, minimizing the effect of this slight delay.

In Scenario 4 (Figure 9), the results are similar to those of Scenario 1, but with a corresponding increase in overall performance resulting from the improved performance of a fourth controller as controller error goes down from 3.25 to 2.16 for a 33.6% reduction in total error.

The reduced error for each process in the four scenarios are difficult to discern and compare in the previous figures. The accumulated error for each task, shown in Figure 10, provides a good way to compare the performance of the system in the various Scenarios, without and with adaptation. In Scenario 1 and Scenario 2, the tasks with adaptation have much less cumulative error than those without adaptation. In Scenario 3, there

is almost no gap between the accumulated error of the two cases, again reflecting the lack of room for adaptation due to the overlapping errors. In Scenario 4 there is once again a large gap reflecting the performance gains achieved by the system.

Table 3 summarizes the total error for the various scenarios. It shows that in all of the scenarios where the error is even partially non-overlapping, significant reductions in error are achieved. In the one case where the perturbations are nearly simultaneous and the controllers and the perturbations are identical, a very slight performance loss is experienced. However, this loss is more than made up for by the significant gains achieved in the other scenarios. Overall in these four scenarios the controllers achieved 22.3% less error. While the actual benefits in real system will vary, the artificially constructed worst-case parameters of Scenario 3 are unlikely to occur in practice and we expect that the system will therefore provide significant performance improvements in most real situations.

6. Related Work

This work represents a synergistic combination of two technologies: adaptive soft real-time scheduling algorithms and adaptive control applications. Control applications are not considered soft in the traditional sense, in that we do not allow a control application to miss a deadline. Nevertheless, adaptive control applications may use the same scheduling approaches developed to support soft real-time applications that adjust their resource consumption in order to consume limited available resources that change dynamically.

We have adapted QoS Levels to work with a new class of real-time control applications which we call adaptive control. The control application may choose among a discrete set of controllers, each of a different sampling rate, and hence consume a different amount of CPU resource at each level [10]. In a similar work, Cervin et al. [6] present a system in which feedback from control tasks is used to adjust the workload by rescaling the task periods. However, in this work all periods are rescaled each time, and there is no provision to trade off resources among tasks that need them more urgently.

Tokuda and Kitayama [14] developed QoS Levels as a mechanism to be used with RT Mach [15] and processor capacity reserves [11]. Their implementation of levels was limited only to differences in either temporal or spatial processing, as opposed to DQM where QoS levels may be determined by the application developer. Rajkumar et al. have developed a theoretical QoS model called Q-RAM [12, 13] that is similar to the one presented here. Q-RAM uses continuous benefit functions to specify application benefit as a function of resource allocations. Lee extended Q-RAM to address applications with discrete benefit functions [8], a model similar to QoS Levels. Unlike DQM, which was implemented in a soft-real time system, these systems were based on theoretical models. Abdelzaher et al. use a similar notion to QoS Levels to support automated flight control processes distributed over a pool of processors [1]. Their system was built upon the real-time support of RT Mach, but the work did not indicate how to select appropriate QoS Levels. In this work, we implemented QoS Levels

Table 3: Accumulated Error for the Control Tasks in the Four Scenarios

Scenario 1: Three Tasks w/Non-Overlapping Error			
Task	Error (w/out Adaptation)	Error (w/Adaption)	Change
SRT-1	0.72	0.54	-25.3%
SRT-2	0.72	0.54	-25.3%
SRT-3	0.72	0.54	-25.3%
Total	2.16	1.61	-25.3%

Scenario 2: Three Tasks w/Partially Overlapping Error			
Task	Error (w/out Adaptation)	Error (w/Adaption)	Change
SRT-1	0.72	0.54	-25.2%
SRT-2	0.72	0.54	-25.7%
SRT-3	0.72	0.54	-25.4%
Total	2.16	1.61	-25.4%

Scenario 3: Three Tasks w/Fully Overlapping Error			
Task	Error (w/out Adaptation)	Error (w/Adaption)	Change
SRT-1	0.72	0.72	+0.0%
SRT-2	0.72	0.72	+0.0%
SRT-3	0.72	0.74	+2.3%
Total	2.16	2.18	+0.9%

Scenario 4: Four Tasks w/non-Overlapping Error			
Task	Error (w/out Adaptation)	Error (w/Adaption)	Change
SRT-1	0.72	0.54	-25.3%
SRT-2	0.72	0.54	-25.7%
SRT-3	0.90	0.54	-40.3%
SRT-4	0.90	0.54	-40.3%
Total	3.25	2.16	-33.6%

into the RBED real-time system.

The RBED scheduler handles dynamic mixed workloads, flexibly adjusting the rates and deadlines assigned to applications as they run such that constraints are never violated and all hard deadlines are met. For adaptive tasks that may change their resource utilization, such as the adaptive control applications, Buttazzo et al. formulated an algorithm in which rate changes are modeled using spring coefficients [5]. This novel approach incorporates constraints for dynamically changing resource assignments. Liu and Goddard have implemented an enhancement to the rate-based execution model, which also supports task rate variability [9]. Our goal is similar, but the approach used by RBED differs as all resource assignments are changed within an EDF framework.

7. Conclusion

Traditional control and hard real-time systems have evolved hand-in-hand. Our work continues this evolution by merging adaptive control with adaptive soft real-time processing. We have extended the RBED integrated real-time scheduler to include dynamic QoS Level soft real-time processing, and implemented an adaptive control system on this platform. We show significant performance improvement when using our QoS level framework to execute adaptive control applications with up to 40% improvement in some controllers and an average of 22% improvement over all of our experiments, with no corresponding increase in resource usage.

These improvements comes from two standpoints: from the control perspective, the adaptive system accumulates less error than the non-adaptive system, providing better control even when computing resources shared by the controllers are limited. From the system perspective, the adaptive approach is also an improvement because, even while achieving better control, the control tasks consume less resources.

Future directions for this work include several avenues: for soft real-time processing we will continue to refine Adaptive QoS in RBED, and include provisions for continuous as well as discrete levels of quality. One novelty of the work described here is that we allowed the static benefit specifications of adaptive tasks to change

based on run-time performance. This was a natural by-product of using control applications, because error provides a reasonable metric for the critically of the application. There may be a similar analogous approach for soft real-time applications—using a metric of run-time performance to scale the user-provided benefits, we may get better results when trying to optimize the system over several dimensions. In this study, all the adaptive tasks running simultaneously were control tasks; we have not yet studied a system where some soft real-time tasks are control tasks and others are not.

References

[1] Tarek F. Abdelzaher, Ella M. Atkins, and Kang G. Shin. QoS negotiation in real-time systems and its application to automated flight control. In *Proceedings of the Real-Time Technology and Applications Symposium (RTAS97)*, June 1997.

[2] Scott Brandt and Gary Nutt. Flexible soft real-time processing in middleware. *Real-Time Systems*, 22:77–118, 2002.

[3] Scott A. Brandt. *Dynamic Soft Real-Time Processing with QoS Level Resource Management*. Ph.d. dissertation, University of Colorado at Boulder, June 1999.

[4] Scott A. Brandt, Scott Banachowski, Caixue Lin, and Timothy Bisson. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS 2003)*, pages 396–407, December 2003.

[5] Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3):289–302, March 2002.

[6] Anton Cervin, Johan Eker, Bo Bernhardsson, and Karl-Erik Årzén. Feedback-feedforward scheduling of control tasks. *Real-Time Systems*, 23:25–53, 2002.

[7] Michael R. Garey and David S. Johnson. *Computers and Intractability*. W.H. Freeman, 1979.

[8] Chen Lee, John Lehoczky, Raj Rajkumar, and Dan Siewiorek. On quality of service optimization with discrete QoS options. In *Proceedings of the Real-Time Technology and Applications Symposium (RTAS99)*, June 1999.

[9] Xin Liu and Steve Goddard. Scheduling legacy multimedia applications. *Journal of Systems and Software*, July 2004. Accepted for publication.

[10] Pau Marti, Gerhard Fohler, Krithi Ramamritham, and Josep M. Fuertes. Improving quality-of-control using flexible time constraints: Metric and scheduling issues. In *Proceedings of the 23nd IEEE Real-Time Systems Symposium (RTSS 2002)*, December 2002.

[11] Clifford W. Mercer, Stefan Savage, and Hideyuki Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *Proceedings of the 1994 IEEE International Conference on Multimedia Computing and Systems (ICMCS '94)*, pages 90–99, May 1994.

[12] Ragunathan Rajkumar, Chen Lee, John Lehoczky, and Dan Siewiorek. A resource allocation model for QoS management. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS 1997)*, December 1997.

[13] Ragunathan Rajkumar, Chen Lee, John Lehoczky, and Dan Siewiorek. Practical solutions for QoS-based resource allocations. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS 1998)*, December 1998.

[14] Hideyuki Tokuda and Takuro Kitayama. Dynamic QoS control based on real-time threads. In *Proceedings of the Fourth International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 114–123, 1993.

[15] Hideyuki Tokuda, Tatsuo Nakajimi, and Prithvi Rao. Real-time Mach: Towards a predictable real-time system. In *Proceedings of USENIX Mach Workshop*, October 1990.

[16] Manel Velasco, Josep Fuertes, and Pau Marti. The self triggered task model for real-time control systems. In *Work in Progress Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS WIP 2003)*, pages 67–70, December 2003.

[17] Jose Yepez, Josep Fuertes, and Pau Marti. The large error first (LEF) scheduling policy for real-time control systems. In *Work in Progress Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS WIP 2003)*, pages 63–66, December 2003.