Optimal State Feedback Based Resource Allocation for Resource-Constrained Control Tasks

Technical Report (UCSC-CRL-04-06)

Pau Marti, Caixue Lin and Scott A. Brandt
University of California
Santa Cruz, USA
pmarti,lcx,scott@cs.ucsc.edu

Manel Velasco and Josep M. Fuertes
Technical University of Catalonia
Barcelona, Spain
manel.velaso,josep.m.fuertes@upc.es

Abstract

In many application areas, including control systems, careful management of system resources is key to providing the best application performance. Most traditional resource management techniques for real-time systems with multiple control loops are based on *open-loop* strategies that statically allocate a constant CPU share to each controller, independent of their current resource needs. This provides average control performance with minimal overhead but in general fails to provide the best performance possible within the available resources. We show that by using feedback to dynamically allocate resources to controllers as a function of the current state of their controlled systems, control performance can be significantly improved. We present an optimal resource allocation policy that maximizes control performance within the available resources and provide experimental results showing that the optimal policy 1) significantly increases control performance compared to traditional control system implementations (by more than 20% in our experiments), 2) maximizes control performance over other feedback-based policies, 3) saves resources when perturbations occur infrequently, and 4) incurs negligible overhead.

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

1 Introduction

In many application areas, including control systems, fully exploiting the available system resources is crucial to maximizing application performance. Most traditional resource management techniques for real-time systems with multiple control loops are based on *a priori* characterizations of the expected workload. They use fixed parameters that are configured at system setup time. At run-time, resources are shared by all tasks according to the pre-established allocations regardless of the dynamics of the control applications, thus working *open loop*. This is done both in the early work on real-time scheduling [15] and in more recent approaches to control and real-time systems [19].

Open loop resource allocation policies work well because they guarantee a constant CPU share to each controller, allowing them to meet given control performance specifications. However, taking a closer look at the behavior of control loops and at the relation between control performance and controller execution rate, open loop policies may not be optimal for resource constrained systems. As suggested in [16], a controller may not require the assigned execution rate if the controlled system is in equilibrium. In this case, the contribution of each job can be considered

useless, i.e., resources are wasted. These underutilized resources could be more usefully employed by other tasks with higher processing demands. If a controlled system is affected by a perturbation and brought away from its equilibrium point, an increase in the rate of the controller will decrease system deviation and hasten system recovery, improving control performance.

Taking this observation as a baseline for our current work on control performance optimization in real-time systems, we have developed dynamic resource management policies for control tasks that allocate resources at run-time based on *feedback* information from the jobs that the controllers are performing. We show that a real-time system with multiple control loops can provide improved control performance by assigning resources to controllers based on whether control loops are affected by perturbations or not. Alternatively, with these techniques the same control performance could be achieved using less resources.

As a key contribution of this paper, we present an optimal resource allocation policy for control tasks based on feedback from the controlled systems dynamics, i.e., from their *state*, that maximizes control performance within the available resources. We argue that the optimization of control systems performance (for a wide class of control systems) when resources are limited and allocated as a function of the state of the controlled systems can be formulated as a linear constrained optimization problem [9]. Consequently, the solution to this problem is a feasible (in terms of computational complexity) algorithm for a run-time resource allocator, and provides the optimal resource allocation policy for control tasks.

To successfully implement the state feedback optimal resource allocation policy as well as two other state feedback based policies used as a basis for comparison, two key aspects must be considered. First, these policies require the implementation of controllers capable of running with different sampling frequencies given different resource allocations, providing better control performance when given more resources (as explicated in [17] and [16]). This feedback-based approach to resource management takes advantage of the fact that the performance of these classically designed controllers is improved by a run-time resource allocator that gives the controllers more resources exactly when they need it the most, i.e. when their controlled system is experiencing the greatest deviation from the equilibrium point.

Second, the implementation of such policies requires a flexible real-time system capable of dynamically changing the allocated resources (via e.g., the controllers' periods) while guaranteeing tasks timing constraints, as does the Rate-Based Earliest Deadline (RBED) integrated real-time system [4]. RBED facilitates the implementation of our feedback resource allocation policies because it is based on the Resource Allocation/Dispatching (RAD) integrated scheduling model, which explicitly separates the management of the amount of resources allocated to each task from the timing of the delivery of those resources. This separation allows the resource management to be precisely tailored to the needs of the individual control tasks.

To illustrate the benefits of our feedback approach to resource management, we implemented the policies in

RBED and performed extensive experiments on simulated inverted pendulums. Our results show that, on the same sequence of randomly generated perturbations, the optimal policy 1) significantly increases control performance compared to traditional control system implementations (by more than 20% in our experiments), 2) maximizes control performance over other feedback-based policies, 3) saves resources when perturbations occur infrequently, and 4) incurs negligible overhead.

2 Related work

Optimization of control systems performance subject to resource constraints has been examined before. Seto *et al.* [19] optimized task frequencies at the design stage in order to minimize a control performance index defined over the task set. Rehbinder and Sanfridson [18] proposed an off-line scheduling method based on optimal control theory. None of the previous resource-constrained control system optimization work has examined run-time adaptive resource management for optimization of control systems performance, as we do.

Different approaches to control systems and run-time resource allocation policies have also been examined before. Shin and Meissner [21] presented a resource allocation technique for multiprocessor systems where tasks are reallocated and periods are changed while taking into account control performance. Beccari *et al.* [2] present a scheduling technique for adaptation of soft real-time load to available computational capacity in the context of autonomous robot control architectures. Caccamo *et al.* [6] allowed tasks' computation times to range from average to worst case computation times and adjusted periods at runtime to optimize control performance and enhance schedulability using server approaches. Cervin and Eker [7] present a case study with hybrid controllers , where the sampling rates are adjusted to avoid CPU overloads. Cervin *et al.* [8], proposed a scheduling architecture for real-time control tasks where the scheduler uses feedback from execution time measurements and feed-forward from workload changes to adjust the sampling periods of the control tasks so that the combined performance of the controllers is optimized. Buttazzo *et al.* [5] present a method for promptly react to overload conditions, while still guaranteeing a given control performance. However, none of the previous work uses feedback from the controlled systems dynamics in order to reassign resources as we do.

Our approach has some similarities to the feedback scheduling architectures presented by Zhao and Zheng [24], and by Henriksoon *et al.* [13]. Zhao and Zheng discussed an event feedback scheduling strategy in which controllers are executed according to the dynamics of the controlled systems to meet asymptotical and exponential stability performance criteria, without adapting sampling periods. The goal of their approach is the design of the control laws that meet those performance requirements. Our goal is to optimize a performance criterion based on the errors of the set of controlled systems, by appropriately varying the execution frequency of each controller. Henriksoon presented a scheduler that allocates CPU time to a specific class of controllers (model predictive controllers) based

on feedback information from the optimization algorithm carried out by each controller. In such a framework, performance optimization is achieved by dynamically varying and controlling the executions time of each controller. Our approach targets a wider class of control systems (linear systems), control performance optimization is based on feedback information from the controlled systems, and it is achieved by dynamically adjusting the sampling period of each controller (that is, the period of each control task), which is determined at each resource reallocation.

The optimal resource allocation policy for control tasks that we present in this paper optimally solves the Quality-of-Control (QoC) scheduling problem formulated by Marti *et al.* [16], but in terms of resource management. As suggested in [16] and further developed by Yepez *et al.* [23] and Velasco *et al.* [22], the dynamics of the controlled systems are the key to better exploiting system resources and improving control systems performance in resource constrained control systems. Preliminary results of the feedback approach to resource management, that are briefly included in this paper, were reported previously [14].

3 System model and feedback architecture

The system we consider is a real-time system with multiple control loops. That is, we have a set of controllers (or control tasks), each one controlling a physical system (or controlled system), sharing a single CPU. The real-time system may also integrate other hard real-time tasks as well as soft-real time and non-real-time tasks. Since the main goal of this work is control performance optimization via feedback resource allocation techniques, henceforth, we will focus on control tasks. Due to resource limitations, the controllers cannot all simultaneously run with their highest possible sampling frequency, providing the best possible control performance equivalent to what they would provide if they were running alone on the CPU. In order to optimize the performance delivered by this set of controllers, we apply our feedback resource allocation policies.
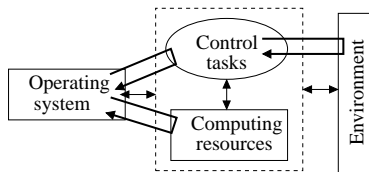
3.1 Basic operation of the feedback architecture

Figure 1: System Model

The basic operation of the system, schematically illustrated in Figure 1, can be summarized as follows: at each control task execution, the task samples its controlled system (the *environment* in Figure 1), capturing its current dynamics, in order to execute the control algorithm and output the control signal. Each control signal affects the

controlled system dynamics in such a way that the system is driven to the specified set point (also called equilibrium point). Each sample is an *instantaneous picture* of the dynamics of the controlled system. It indicates its *state*, that is, whether the controlled system is affected by a perturbation or not, providing the magnitude of the controlled system deviation with respect to the equilibrium point (deviation that is called controlled system *error*). This information is fed back to the operating system in order to allow the system to reallocate resources accordingly (see Section 3.2 for further explanation on the type of information that is fed back to the system).

At the system level, the state of each controlled system is used to re-scale the *static* relation that can be established between resources and control performance, which is specific for each control loop (given the controlled system, the controller and a range of periods). As pointed out in Cervin *et al.* [8], a controller can normally give satisfactory performance within a range of sampling periods. This relation, that is further explained in Section 3.2, can be computed *a priori* and condensed in a function (that we call the *performance criterion*) that maps controller period to control performance.

At the system level, we could only use this static information (all performance criteria of the set of control loops) to determine the allocation of resources such that all task are schedulable and the performance criteria are optimized (as is done by Cervin *et al.* [8]). However, by doing so, the maximum benefit (in terms of control performance) could possibly be achieved by several different resource allocations and in the absence of other information the system has no way of knowing which is the best choice. By taking into account the current dynamics of each controlled system, we can determine the optimal allocation.

To illustrate the problem, suppose there are two control tasks characterized by the same performance criterion, each of which may use one of two equal (in terms of control performance and rate) controllers, one with a higher sampling rate and one lower. Suppose also that because the amount of computing resources is limited, so the system cannot simultaneously run both controllers at their highest rate. 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 control benefit (provided by the performance criteria), both choices are equal. Nevertheless, it may happen that one controlled system is in equilibrium and the other is not. In that case, the best choice in terms of control performance optimization 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, as argued previously [16]. To make this possible, and to assign resources according to the dynamics of each controlled system, for each control loop we re-scale each performance criterion by the state (feedback measure) of the controlled system.

Finally, resource allocation also considers current system load (*computing resources* in Figure 1) to determine how much resources are available to be allocated and how many processes need them.

With all this information (performance criteria, controlled systems current states, and current system workload) at the system level, resources are reallocated to control tasks according to a particular resource allocation policy

(see Section 5.1) with the objective of optimizing the overall control performance of the set of control loops (or, alternatively, to save system resources), while guaranteeing task set schedulability.

3.2 Control systems and performance

We now examine the formal aspects of the optimization problem. Let (1) and (2) be the ordinary vector differential equations (called *state* and *output* equations, respectively) that describe the linear dynamics of each ith controlled system

$$\dot{x}_i(t) = f_i(x_i(t), u_i(t), t) \quad t \in \mathbb{R}^+ \tag{1}$$

$$y_i(t) = d_i(x_i(t)) \tag{2}$$

where the functions f_i and d_i are linear, $u_i(t)$ is the *control input* to the dynamical system, and the vector $x_i(t) = [x^1(t), \ldots, x^n(t)]$ is the state of the system at time t and its elements are called *state-variables*. The state and output equations defining a given system can be considered an abstract summary of the data obtained by subjecting the system to different inputs (control signals) and observing the corresponding outputs.

Without loss of generality, if we consider the equilibrium point of all controlled systems to be zero, the norm of the state vector, $|x_i(t)|$, is the distance that measures how far each controlled system is from its equilibrium point at any given time $t > 0$. This measure tells how *critical* the situation is for each controlled system; the higher its value, the worse the system. We define this measure (also called *error* (3)) as the feedback information that each controller, at each sample, will send to the system for the re-scaling of each performance criterion. If, for a given control loop, all states cannot be measured, they can be determined from the available measurements and a model [1].

$$e_i = |x_i(t)| \tag{3}$$

Therefore, more or less resources will be assigned at run-time to each controller taking into account the controlled system error: the higher the norm of the states (i.e., the higher the error), the more urgently a controller requires more resources.

For each control task, we specify a performance criterion $p_i(r_i)$ that relates control performance under different task rates, r_i. The rate (also called partial utilization factor) is the relation between each tasks' worst case execution time c_i and its period h_i, $r_i = \frac{c_i}{h_i}$. Since the worst case execution time is a constant value, any variation on the task rate implies a variation on the task period (and vice-versa).

Since controller design attempts to minimize the system error produced by certain anticipated inputs, traditional linear or quadratic performance criteria (also called performance indices or cost functions) are mainly based on

measures of the system error (see [11] for a review of classic control criteria or [20] for a review of performance measures to evaluate real-time computer control systems).

Therefore, our performance criteria, that will be rescaled by the errors (3), should capture the relation between these indices and the tasks' periods. In fact, the relation between control performance (measured using standard quadratic or linear performance index) and a range of allowed periods can be approximated by a linear relationship [8]. Therefore, for a given control task i, we approximate its performance criterion $p_i(r_i)$ by a linear increasing function (4), that establishes the following relation for each control loop: *the higher the rate (i.e., the shorter the sampling period) of the controller, the better the control performance*. The α_i parameter in (4) is specific for each control loop and can be obtained prior to system run-time.

$$p_i(r_i) = \alpha_i r_i \qquad (4)$$

Although this linear approximation is not an oversimplification and it covers a wide class of control systems, as we will discuss in Section 4, the optimal resource allocation policy also admits performance criteria in the form of polynomials of grade less than five.

We require controllers capable of running with different frequencies. To do so, we design controllers for the class of linear systems (that can be specified by (1) and (2)) using classic design procedures, either in the continuous-time domain followed by discretization, or directly in the discrete-time domain [1]. In the end, each control law is an algorithm that depends on the sampling period. We specify a range of sampling periods for which the closed loop requirements are met and allow the controller, implemented within a single task that sequentially executes sampling, control algorithm and actuation, to execute with a run-time period that belongs to the specified range (for full details, see Marti *et al.* [17]), adapting the gains accordingly. System stability is analyzed using the approach described by Dogruel and Özgüner [10].

4   Performance optimization of control systems

At the system level, each control task τ_i can be characterized by its rate r_i (which is a *system* characterization), its performance criterion p_i (which relates *system* resources and *control* performance), and its controlled system error e_i (which is a *control* characterization), represented by (5)

$$\tau_i = \{r_i, p_i, e_i\} \qquad (5)$$

With this information, for a given set of n control tasks, $\tau_1, \ldots, \tau_n$, the problem is to determine the task rates r_i, $i = 1, \ldots, n$, such that all the tasks are schedulable and the overall control system performance is maximized.

The resource allocation problem can be formulated as a generic constrained optimization problem as follows,

$$\text{maximize} \quad g(p_i(r_i), e_i) \tag{6}$$

$$\text{subject to} \quad \sum_{i=1}^{n} r_i \leq U_d \tag{7}$$

where the solution is a vector $\vec{r} = [r_1, r_2, \ldots, r_n]$ that maximizes the control performance delivered by the set of controllers, represented by the objective (vector) function g in (6), restricted to the utilization feasibility constraint specified in (7), where U_d is the desired global resource utilization factor for the set of control tasks.

The absolute maximum $\vec{r}$ may lie either in the interior, on the boundary, or at the extreme points of the feasibility set defined by (7). A generic algorithm to find the solution can be summarized in four steps (as detailed by Chong and Zak [9]):

Step 1: Search for local relative maxima in the interior of the feasibility set by solving the set of equations specified by (8), where $\frac{\partial g}{\partial r_i}$ are the partial derivatives of g with respect to each r_i.

$$\frac{\partial g}{\partial r_1} = 0, \frac{\partial g}{\partial r_2} = 0, \ldots, \frac{\partial g}{\partial r_n} = 0 \tag{8}$$

and keep those $\vec{r}$ that, being interior points of the feasibility set (conforming with the restriction (7)), maximize g.

Step 2: Search for local relative maxima in the boundary of the feasibility set by solving the set of equations specified by (9),

$$\frac{\partial g([U_d - r_2 - r_3 - \ldots - r_n, r_2, r_3, \ldots, r_n])}{\partial r_i} = 0, i \leq n, i \neq 1$$

$$\frac{\partial g([r_1, U_d - r_1 - r_3 - \ldots - r_n, r_3, \ldots, r_n])}{\partial r_i} = 0, i \leq n, i \neq 2 \tag{9}$$

$$\vdots$$

$$\frac{\partial g([r_1, r_2, \ldots, U_d - r_1 - r_2 - \ldots - r_{n-1}])}{\partial r_i} = 0, i \leq n, i \neq n$$

and keep those $\vec{r}$ that maximize g.

Step 3: Search for the g values of the feasibility set extremes as specified in (10),

$$g([U_d, 0, \ldots, 0]), g([0, U_d, \ldots, 0]), \ldots, g([0, 0, \ldots, U_d]) \tag{10}$$

and keep those $\vec{r}$ that maximize g.

Step 4: Choose a $\vec{r}$ among those obtained in *Step 1, 2* and *3* that maximize g.

Depending on the objective function g, solving the optimization problem may not be feasible for an on-line real-time resource manager. However, in the case of control tasks characterized as described in Section 3.2, the optimization problem can be simplified, it is directly solvable, and the algorithm that obtains the solution can feasibly be executed at run-time, as we explain next.

Assuming that each controller is independent in the sense of controlling an independent controlled system (as we assumed in the system model in Section 3), the function $g(.)$ that links all of the control performance benefits (which are given by the performance criteria p_i defined in (4)) can be considered as the sum (possibly weighted) of all individual benefits obtained by each controller (as was also done in the optimization procedures for control tasks presented in [19] or [8]).

Each performance criterion can be weighted, w_i, in order to provide a mechanism allowing appropriate comparisons among the control loops in the system. For example, a control loop in charge of the brake system of a car may be more critical than the one in charge of the air conditioning. In addition, by defining the re-scaling of each performance criterion to account for the each controlled system error (defined in (3)) as $e_i p_i$, for the given set of n controllers, we can rewrite the optimization problem as follows

$$\text{maximize} \quad \sum_{i=1}^{n} w_i e_i p_i(r_i) \tag{11}$$

$$\text{subject to} \quad \sum_{i=1}^{n} r_i \leq U_d \tag{12}$$

The complexity of the solution of the optimization problem stated in (11) and (12) depends on each function $p_i(r_i)$ due to the fact that equations (8) and (9) have been simplified to the set of equations specified by (13) and (14) (because g has turned into a sum), where $b_i = w_i e_i p_i(r_i)$:

$$\frac{\partial b_1}{\partial r_1} = 0, \frac{\partial b_2}{\partial r_2} = 0, \ldots, \frac{\partial b_n}{\partial r_n} = 0 \tag{13}$$

$$\frac{\partial b_1 (U_d - r_2 - r_3 - \ldots - r_n)}{\partial r_i} = 0, i \leq n, i \neq 1$$

$$\frac{\partial b_2 (U_d - r_1 - r_3 - \ldots - r_n)}{\partial r_i} = 0, i \leq n, i \neq 2 \tag{14}$$

$$\vdots$$

$$\frac{\partial b_n (U_d - r_1 - r_2 - \ldots - r_{n-1})}{\partial r_i} = 0, i \leq n, i \neq n$$

If the performance criteria p_i are linear (as we assumed in our system model in Section 3.2), the optimization

problem becomes linear, and the solution $\vec{r} = [r_1, r_2, \ldots, r_n]$ can be found by performing a simple search (i.e., performing *Step 3*) because equations (13) and (14) corresponding to *Step 1* and *2*, are not properly determined. That is, if p_i are linear ($p_i = \alpha_i r_i$) in (13), we end up with the following set of equations (15)

$$\frac{\partial b_1}{\partial r_1} = \frac{\partial w_1 e_1 \alpha_1 r_1}{\partial r_1} = w_1 e_1 \alpha_1 = 0, \quad \frac{\partial b_2}{\partial r_2} = \frac{\partial w_2 e_2 \alpha_2 r_2}{\partial r_2} = w_2 e_2 \alpha_2 = 0, \quad \ldots , \quad \frac{\partial b_n}{\partial r_n} = \frac{\partial w_n e_n \alpha_n r_n}{\partial r_n} = w_n e_n \alpha_n = 0 \ (15)$$

that are not determined. The same happens with the set of equations specified in (14). Therefore, by simply performing *Step 3* customized for the problem stated in (11) and (12), that is, by evaluating

$$g([U_d, 0, \ldots, 0]) = b_1(U_d) + b_2(0) + \ldots + b_n(0) = w_1 e_1 \alpha_1 U_d$$

$$g([0, U_d, \ldots, 0]) = b_1(o) + b_2(U_d) + \ldots + b_n(0) = w_2 e_2 \alpha_2 U_d \tag{16}$$

$$\vdots$$

$$g([0, 0, \ldots, U_d]) = b_1(0) + b_2(0) + \ldots + b_n(U_d) = w_n e_n \alpha_n U_d$$

we will find the optimal resource allocation. Note that (16) is equivalent to finding the maximum $w_i e_i \alpha_i, i = 1 \ldots n$.

Theorem 1 *The optimal solution $\vec{r} = [r_1, r_2, \ldots, r_n]$ of the optimization problem (11) and (12) is $\vec{r} = [0, 0, \ldots, 0, r_i = U_d, 0, \ldots, 0]$, $i \in [1, \ldots, n]$ such that $w_i e_i \alpha_i$ is maximum $\forall i \in [1 \ldots n]$, if the set on control tasks are described by (5).*

Proof.

Follows from the argument above. $\square$

Observation 1 In terms of resource allocation, the theorem states that we should assign all the available CPU (that is, U_d) to the control task with maximum $w_i e_i p_i$. If all of the functions p_i and all the weights w_i are the same, we should assign all of the available resources to the control task with the largest error e_i. In practice we need to assign a minimum rate to the rest of the control tasks so that stability tests can be performed and they can continue to monitor the state of their controlled systems. This result dictates that the control task with the largest error should receive all of the resources remaining after every task has received its minimum.

Observation 2 If the p_i are not linear but still polynomial functions on r_i of grade less than five [12], an analytical solution can be found following *Steps 1, 2* and *3* by solving equations (13), (14) and (10), turning the solution into a feasible (in terms of computational complexity) algorithm for a run-time resource manager.

Observation 3 For the case of linear p_i, the geometric explanation of the optimal solution of the problem formulated

10

by (11) and (12) is as follows. The optimal solution $\vec{r}$ is one of the extreme points (vertex) that is maximum in the projection of the hyperplane given by the constraints (11) on the hyperplane defined by the objective function (12). Figure 2 illustrates the case for two control tasks where $U_d = 0.8$, both controllers have same performance criterion and weights ($\alpha_1 = \alpha_2 = 1$ which implies that $p_1(r_1) = r_1$ and $p_2(r_2) = r_2$, and $w_1 = w_2 = 1$), but one controlled system at time t has a bigger error ($e_1 = |x_1(t)| = 4$) than the other ($e_2 = |x_2(t)| = 1$). As it can be seen in the figure, the maximum benefit considering the schedulability constraints is found at $\vec{r} = [0.8, 0.0]$ (extreme point).
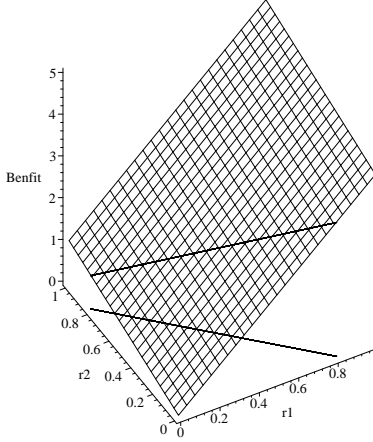
Figure 2: Optimal Solution

5 Flexible real-time processing in RBED

Our control feedback architecture is implemented in the RBED integrated real-time system [4], which supports hard real-time, soft real-time and best-effort processes. RBED allocates resources to processes as a percentage of the CPU such that the total allocated is less than or equal to 100% and then schedules all processes with the Earliest Deadline First (EDF) algorithm [15]. RBED dynamically changes allocated resources and application periods without violating EDF constraints, guaranteeing that tasks never misses their assigned deadlines.

In RBED, we have implemented control tasks as flexible real-time processes which have a known worst case execution time c_i, but flexible period choices, h_i, that take values from pre-defined ranges (which correspond to the sampling periods of the controllers), $h_i \in [h_{min} \ldots h_{max}]$ (which for now we consider continuous), and relative hard deadlines equal to their periods. Given a set of periodic control tasks $\tau_i (1 \leq i \leq n)$, their resource utilizations are defined as $r_i = \frac{c_i}{h_i}$. If $\sum_{i=1}^{n} r_i \leq 1$, the EDF algorithm in RBED guarantees that all of the deadlines of the tasks' jobs will be met. At any time t, upon a dynamic resource re-allocation by adjusting the periods of the tasks, the utilizations of the tasks are updated as $r_i' = \frac{c_i}{h_i'}$ such that $\sum_{i=1}^{n} r_i' \leq 1$. Note that any change to a task's period is subject to the constraints described in [4]. Therefore, RBED still guarantees all of the new deadlines of the tasks when they are assigned new periods.

5.1 Feedback resource allocation policies

We implemented the optimal feedback resource allocation policy (*optimal*, described in Section 4), for control tasks in RBED. In order to better demonstrate the benefits of the optimal policy and to evaluate its performance, we also implemented two adaptive feedback based resource allocation techniques, called *proportional* and *discrete*. Unlike optimal, the proportional policy fairly distributes resources based on $w_i e_i p_i$ for all controlled systems such that all tasks get a CPU share proportional to their error. The discrete policy, which is a mixture of optimal and proportional with a set of discrete values for the task periods, reduces computational overhead due to its simplicity. To provide a direct comparison with traditional control system implementations, we also implemented a baseline policy, *static*, in which all controllers always share the available resources equally and no dynamic resource allocation is used. The static policy implements the "traditional" controller and is used for examining the overall performance benefit of adaptive feedback based resource allocation for control systems.

5.1.1 The optimal policy

The optimal policy, like all of the adaptive feedback based resource allocation policies implemented in this paper, allows each control task to specify a performance criterion p_i (see Section 3.2), which is a continuous function that relates its period (and thus resource usage) to the control performance it provides. As we discussed in Section 4, the optimal solution is to assign all the available resources to the controller whose controlled system is currently facing the largest error (if all the control loops are the same, that is, with equal w_i and p_i), taking into account that the system must also guarantee a minimum rate for the remaining controllers. With different p_i and weights w_i (meaning different type of controllers and/or different controlled processes), the resources would be assigned to the controller having the highest $w_i e_i p_i$. This is summarized in (17)

$$r_i = \begin{cases} U_d - \displaystyle\sum_{1 \leq j \leq n, j \neq i} r_{j,min}, & \text{if } w_i e_i p_i(r_i) \text{ is maximum} \\ r_{i,min}, & \text{otherwise} \end{cases} \quad , \quad \frac{c_i}{r_i} \in [h_{min} \dots h_{max}] \tag{17}$$

where n is the number of controllers and $r_{j,min}$ (corresponds to h_{max}) is the guaranteed minimum utilization of the controller j.

The optimal policy keeps track of which task has maximum $w_i e_i p_i$. Therefore, the dynamic resource allocation can be completed by a linear scan of the list of n tasks in $O(n)$ time. Note that dynamic resource allocation only occurs when the task with the maximum $w_i e_i p_i$ changes, which greatly reduces the overhead.

5.1.2 The proportional policy

The proportional policy assigns resources to each controller as a proportion of $w_i e_i p_i$. In our implementation, the resource allocation algorithm is given by

$$r_i = \frac{w_i e_i p_i(r_i)}{\displaystyle\sum_{1 \le j \le n} w_j e_j p_j(r_j)} \cdot U_d \;, \;\; \frac{c_i}{r_i} \in [h_{min} \ldots h_{max}] \tag{18}$$

Fairness comes from the fact that any controller whose controlled process is subject to a perturbation increases its utilization according to its relative degree of error. If (18) gives a utilization that results in a longer sampling period than h_{max}, then the controller will run at h_{max}. This mechanism allows us to guarantee a minimum sampling rate for all the controllers.

The proportional policy has the same time complexity $(O(n))$ for the resource allocation as the optimal policy though it may scan the task list a second time to distribute the leftover resources after the first round distribution based on (18). However, any change to the error of a controlled system will cause a dynamic resource allocation so the actual number of dynamic resource allocations and thus the introduced overhead of the proportional policy is always more than that of the optimal policy.

5.1.3 The discrete policy

In the discrete policy, based on the assumption of a discrete set of periods instead of the continuous range defined for optimal and proportional, each control task has a discrete function corresponding to p_i evaluated on only a few values. It defines discrete resource levels in terms of the different periods, which are mapped into benefits that will be used in the discrete optimization procedure. The benefit is given by

$$benefit_i = w_i e_i p_i(r_i) = w_i e_i \alpha_i r_i \;, \;\; \frac{c_i}{r_i} \in \{h_{min}, \ldots, h_{max}\} \tag{19}$$

A heuristic algorithm [14, 3] is employed to iteratively increase the level of the control tasks until no more increases are possible within the available resources, providing high average overall system benefit.

The worst-case time complexity of the discrete policy is $O(Ln)$, where L is maximum resource levels in the system (which is equal to the maximum number of sampling periods for all control tasks), because the worst case for a dynamic resource allocation may go through all of the resource levels of every control task. In order to simplify the operation and reduce the overhead, we map the benefit of each resource level with a range of possible error values [14]. In this way, rate readjustment takes place only when the error moves from one range to another. As a result the overhead is significantly reduced (see overhead analysis in Section 6).

13

5.2 Implementation details of the system interfaces

The operation of the feedback based resource allocation model was given in Section 3. In this section we describe the implementation details that allow the communication between control tasks at the application level and the resource manager at the system level.

```
Controller
{
    h_i := h_i^{next}
    x_i := read_input()
    u_i := calculate_output(h_i); send_output(u_i)
    x_i^{next} := update_state(x_i, h_i); e_i := |x_i^{next}|
    r_i := rate_adjust(i, e_i)
    h_i^{next} := c_i / r_i
}

        (a) Controller (Application Level)
```

```
rate_adjust(i, e_i)
{
    if (e_i! = e_i^{old})
    {
        b_i := w_i e_i p_i
        r_i^{next} := resource_allocation(policy, b_i)
        e_i^{old} := e_i
        return r_i^{next}
    }
}

        (b) Dynamic Rate Adjustment (System Level)
```

Figure 3: Pseudo-code for Controller and Dynamic Rate Adjustment

Figure 3 gives the pseudo-code for a controller and the dynamic rate adjustment, linked through the system call **rate_adjust**(). The controller (Figure 3(a)), with execution period h_i,

1) does its control job: it samples the system, calculates the control signal u_i (based on h_i), and sends it to the controlled system (as any traditional controller would do when keeping constant h_i); and

2) triggers the rate adjustment: it computes the next controlled system state vector (*update_state*), whose norm e_i is passed in by the system call, **rate_adjust**(), in order to obtain the new period that will start being used at the following controller execution.

The dynamic rate adjustment (Figure 3(b)) picks the specified resource adaptation policies, static, discrete, proportional, or optimal, to re-allocate the resources based on $w_i e_i p_i$. It does this by assigning different control tasks with appropriate sampling periods, which are finally returned back to the controllers for next-step control.

6 Results

We implemented our feedback based resource allocation policies for control tasks in RBED in the Linux 2.4.20 kernel (for the sake of simplicity and easy prototyping). We have run the system over long periods of time and performed a large number of experiments with randomly generated workloads so that the experiments are general, and not limited to some special cases. All experiments were performed on a standard Intel-based PC equipped with a 1 GHz Pentium III processor, 512MB RAM, and a 40GB hard drive.

6.1 Controlled processes

In our experiments we ran a set of control tasks, each controlling a simulated inverted pendulum. The linear time-invariant state space model we used for each inverted pendulum mounted on a cart is given by

$$
\begin{bmatrix} \dot{\theta} \\ \dot{\omega} \\ \dot{x} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{(M+m)\cdot g}{M\cdot l} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{-m\cdot g}{M} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \omega \\ x \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{-1}{M\cdot l} \\ 0 \\ \frac{1}{M} \end{bmatrix} u(t)
$$

where θ is the pendulum angle, ω is the angular velocity, x is the cart position and v its velocity. For the simulation, we customized all pendulums as follows: mass of the cart $M = 2kg$, mass of the pendulum $m = 0.1kg$, length of the pendulum stick $l = 0.5m$ and gravity $g = 9.81\,m/s^2$.

Each control task implements the same parametric control law obtained by standard pole placement [1], which is parametrized on the sampling period. We have defined all controlled tasks to be the same because it simplifies the performance analysis. Recall that in this case, w_i and p_i are equal for all tasks, which means that the error e_i is the main driving factor in each policy. However, using different controlled processes (p_i) or weights (w_i) would not materially affect the results.

6.2 Workload generation

For each of the four policies we performed experiments in which we ran three control tasks implementing the same control law and reserved a percentage of the CPU capacity (3% in our experiments) to allow the execution of general-purpose tasks. Thus the desired global resource utilization factor (U_d) for the three control tasks is $U_d = 97\%$. Each controller is in charge of a simulated inverted pendulum as described above, and has a fixed worst-case execution time (c_i) of 0.0135s.

With either the optimal or proportional policy, each controller can run at any sampling period (h_i) within 0.03s and 0.05s. With the discrete policy, we defined three resource levels corresponding to three different sampling periods $h_i \in \{0.03s, 0.04s, 0.05s\}$ for each controller. With this configuration, if there are three control tasks in the system, none of them will execute at their highest level ($h_i = 0.03s$) since $(\frac{0.0135}{0.03} + 2 \cdot \frac{0.0135}{0.05}) = 0.99 > 97\% = U_d$ (that is, the required CPU load would exceed what is available). For the static policy, the three controllers share the available CPU ($U_d = 97\%$) equally and thus each of them is given $\frac{97}{3}\%$ of the CPU for the duration of the experiments.

For each of the resource adaptation policies, we ran the three controllers for 1 hour and randomly generated perturbations for each inverted pendulum with different average perturbation intervals. The distance between two consecutive perturbations on the same system varies in such a way that a system may be continuously perturbed

or almost never perturbed (capturing any scenario). That is, with different perturbation intervals, a system may be perturbed fewer than one hundred times or many thousands of times.

6.3 Performance results

We first detail the effects on the execution rate of each controller when using the different resource management policies we have presented. Afterwards, we look at the control performance achieved by these techniques and examine the overhead they incur.

6.3.1 Execution rate patterns

Figures 4(a)–4(d) show the variation on the control tasks periods when using the four different policies: static, discrete, proportional, and optimal, with perturbation interval=4s . In order to look into the detailed progress of the three control tasks, we only show the first 20s of the 1 hour run. In each figure, the *x*-axis represents the time and the *y*-axis represents both the angle of each inverted pendulum (which allows us to see when each perturbation takes place) and the corresponding controller period multiplied by 40 to normalize it to the same range.
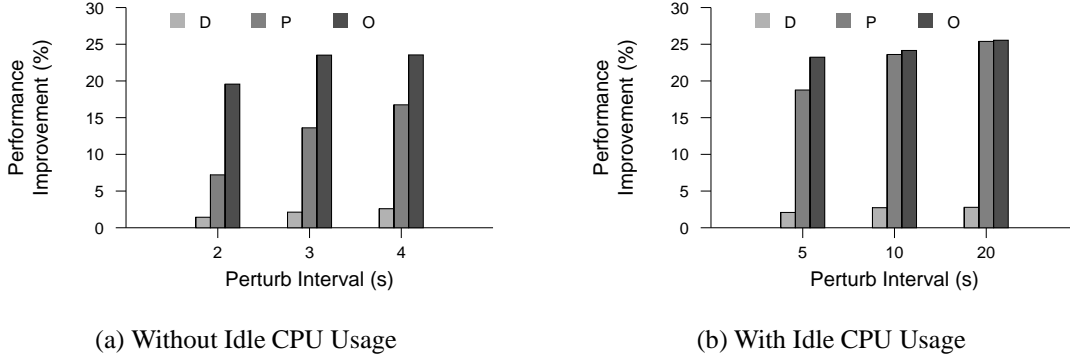

(a) Static

(b) Discrete

(c) Proportional

(d) Optimal
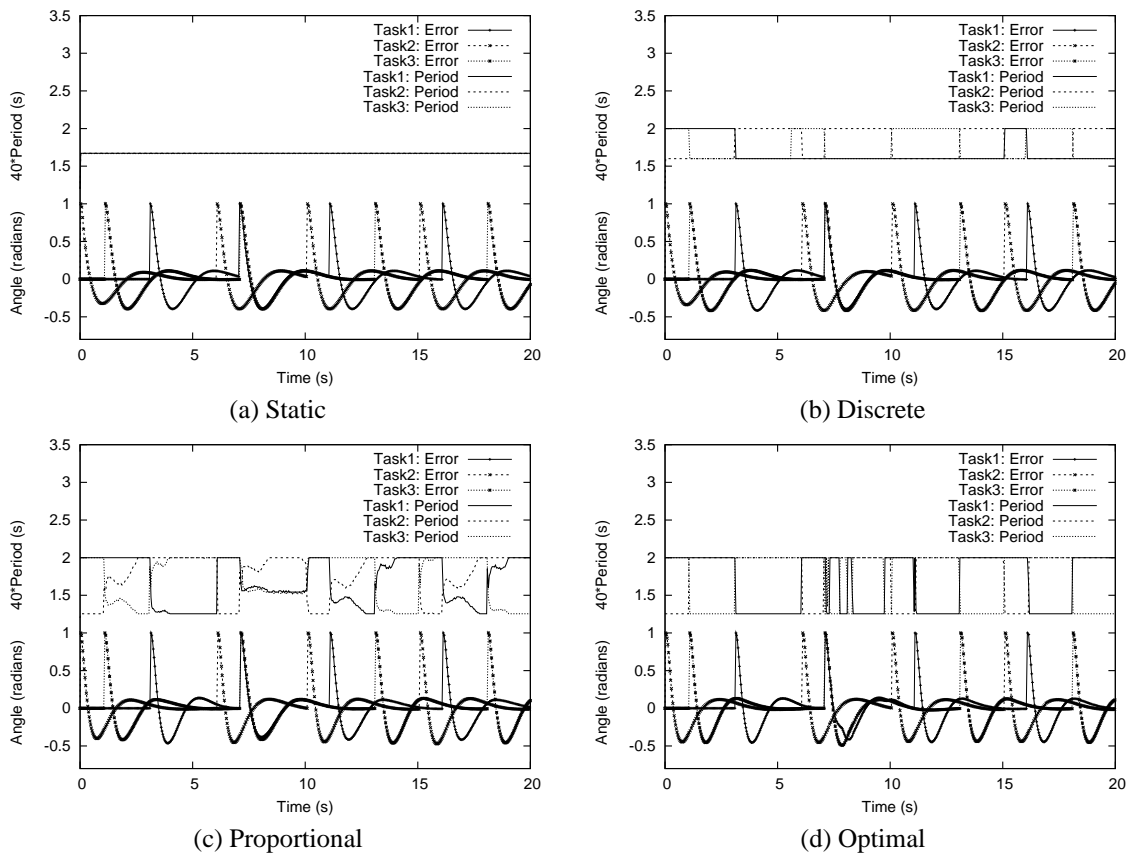
Figure 4: Sampling Period vs. Error in terms of Angle (perturbation interval=4s)

16

(a) 20 Seconds   (b) 1 Hour
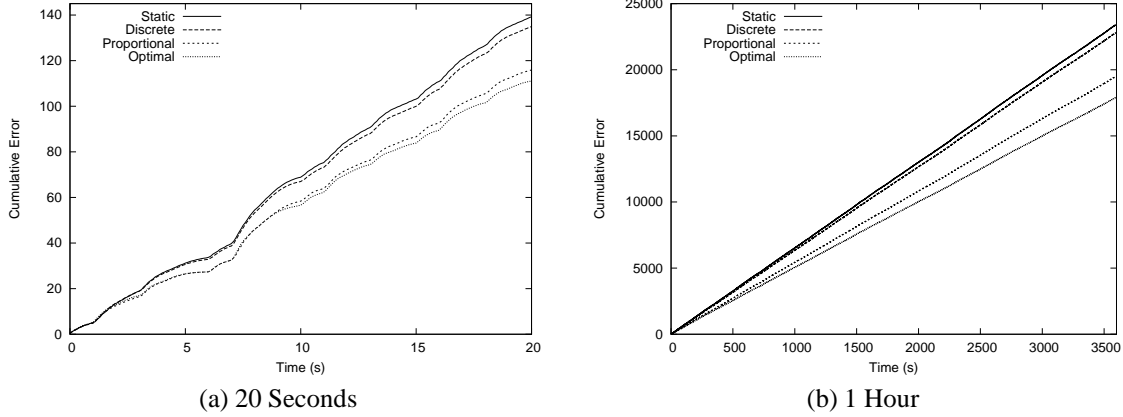
Figure 5: Performance in terms of Cumulative Error (perturbation interval=4s)

(a) Without Idle CPU Usage   (b) With Idle CPU Usage

Figure 6: Performance Improvement Relative to Static

With the static policy (figure 4(a)), every control task has the same sampling period ($h_i \approx 0.042$s, which comes from $h_i = c_i/r_i = \frac{0.0135}{U_d/3}$) and does not change regardless of when the perturbations occur. With discrete, proportional, and optimal (Figures 4(b)-(d)), perturbations force the controllers to dynamically adapt their rates. The specific periods for each controller are determined by each policy. With the discrete policy, the three tasks cannot simultaneously run at their second resource level (corresponding to $h_i = 0.04$) because of the limited available resources. Thus, the only scenario that can increase global benefit is the following: the two control tasks with the larger error run at their second resource level and the control task with the smallest error runs at its lowest resource level (Figure 4(b)). With the proportional and optimal policies, the periods can be any value in the predefined range.

The dynamic period change with the discrete policy is less frequent than with both optimal and proportional (Figures 4(c) and (d) and as explained in Section 5). The sampling rate adaptation of the optimal and proportional policies also occurs with different frequencies. With optimal, the magnitude of the errors determines the period assignment: the one with largest error always runs at the highest rate ($h_i \approx 0.032$) provided the other two at least

17

(a) CPU Usage in 1 Hour   (b) Overhead in 1 Hour
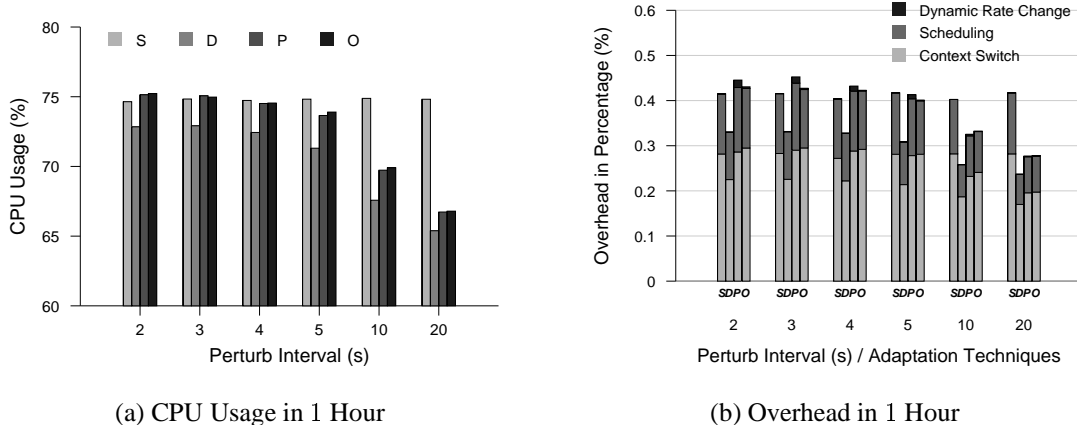
Figure 7: CPU Usage vs. Overhead

can run at their lowest rate ($h_i = 0.05$s). With proportional, any variation in the errors causes a period adjustment among the three controllers, resulting in a larger number of adjustments.

6.3.2 Control performance

We evaluated the control performance of the four different policies by looking at the total cumulative error of the three inverted pendulums (i.e., $\int_0^{t_e} \sum_{i=1}^{3} |x_i(t)| dt$, where t_e is the time each experiment lasts).

The main contribution of the paper is summarized in Figures 5 and 6. Figure 5(a) shows the performance results (in terms of total cumulative error) of the experiments running for 20s with perturbation interval = 4s and Figure 5(b) shows the performance over the course of 1 hour. By looking at both Figure 4 and Figure 5(a), we can see how each perturbation results in an error increase. In Figure 5(a), we see large gaps between the cumulative error of the static/discrete policies and the proportional/optimal policies. Furthermore, from a general view, the gaps are getting larger as time progresses, which shows both that our policies improve overall control systems performance and that the optimal policy achieves the highest overall control performance improvement.

Figure 6 shows the performance improvement in terms of accumulated error (1 hour experiments) against the baseline, static policy, with different perturbation intervals. Figures 5 (a) and (b) were for a specific perturbation interval. Now we show the same results for different perturbation intervals. Figure 6(a) shows the results when the perturbation intervals are short enough so that all of the available CPU is allocated to the control tasks (i.e., there is always error on at least one of the pendulums). Figure 6(b) shows the results when the perturbation intervals are long enough so that at least a portion of the CPU usage is saved by the control tasks when there is no error. From both figures we see that: 1) discrete, optimal, and proportional achieve better performance than static; 2) optimal outperforms all other policies and reduces accumulated error by 20–25%; 3) discrete only reduces error by about

18

3%, due to the limitations imposed by the available number and predefined values of the discrete periods; and 4) as the perturbation interval increases (Figure 6(b)), the difference between optimal and proportional decreases. This is because, with enough long (e.g., 20s) perturbation intervals, most or all perturbations are non-overlapped and the proportional policy makes essentially the same allocations as the optimal policy, i.e. the available resources are allocated to the only task with error.

6.4 CPU usage and overhead

Besides the evaluation on the control performance, we also thoroughly investigated the resource utilization and evaluated the overhead incurred by the four different policies.

Figure 7(a) shows the measured total CPU usage of all control tasks with the four policies. With a 4s perturbation interval, the three adaptive feedback based resource allocation policies use almost exactly the same amount of CPU as the static policy. As the perturbation intervals increase, beginning at about 5s, the dynamic policies begin to consume less CPU due to the fact that when all the controlled systems are in equilibrium, their execution frequency is set to the minimum. This unused CPU can be allocated to other less time-critical tasks in the system.

Figure 7(b) shows the overhead introduced by the four different policies (measured from our implementation). Context switches are responsible for the majority of the overhead, followed by actual scheduling overhead. The overhead introduced by the dynamic rate change is negligible compared to the control tasks' actual CPU usage and these other sources of overhead. As a result, the overhead is comparable for all four policies. The dynamic policies incur almost no extra overhead relative to the static policy and as the perturbation interval increases the overheads of the dynamic policies are seen to be even less than that of the static policy because they incur fewer context switches. Although negligible, proportional is seen to have the largest overhead due to dynamic rate change (as explained in Section 5).

7 Conclusions

Careful resource management is the key to providing the best possible performance in resource-constrained computing systems. We have presented a feedback-based resource management model for concurrently executing control tasks in a resource constrained environment that allows the system to allocate resources as a function of the state of the controlled systems. We have presented several resource allocation policies based on this model and shown that they provide better overall control performance while using the same amount of (or less) resources than traditional resource allocation practice for control tasks.

We have also presented an optimal state feedback resource allocation policy for control tasks that optimizes control performance within the available resources. We have implemented the policies and presented results showing

that the optimal policy outperforms (in terms of control performance improvement measured as accumulated error) all other dynamic policies that we examined and traditional static policies. The optimal policy also saves system resources when they are not needed, and the overhead incurred by this policy has been shown to be negligible.

References

[1] K. J. Astrom and B. Wittenmark. *Computer-Controlled Systems. Third Edition*. Prentice–Hall, 1997.

[2] G. Beccari, S. Caselli, M. Reggiani, and F. Zanichelli. Rate modulation of soft real-time tasks in autonomous robot control systems. In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, June 1999.

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

[4] S. A. Brandt, S. Banachowski, C. Lin, and T. 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, Dec. 2003.

[5] G. Buttazzo, M. Velasco, P. Marti, and G. Fohler. Managing quality-of-control performance under overload conditionsvalue. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, July 2004.

[6] M. Caccamo, G. Buttazzo, and L. Sha. Elastic feedback control. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, pages 121–128, June 2000.

[7] A. Cervin and J. Ecker. Feedback scheduling of control tasks. In *39th IEEE Conference on Decision and Control*, June 2000.

[8] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén. Feedback-feedforward scheduling of control tasks. *Real-Time Systems*, 23:25–53, 2002.

[9] E. K. Chong and S. H. Zak. *An Introduction to Optimization*. John Villey and Sons, Inc, 1996.

[10] M. Dogruel and U. zgner. Stability of a set of matrices: A control theoretic approach. In *Proceedings of the 34th IEEE Conference of Decision and Control*, Sept. 1995.

[11] R. Dorf and R. Bishop. *Modern Control Sytems. Seventh Edition*. John Villey and Sons, Inc, 1995.

[12] W. Gellert, S. Gottwald, and M. Hellwich. *The VNR Concise Encyclopedia of Mathematics*. Van Nostrand Reinhold Company, 1988.

[13] D. Henriksson, A. Cervin, J. kesson, and K.-E. rzn. Feedback scheduling of model predictive controllers. In *Proceedings of the Real-Time Technology and Applications Symposium (RTAS02)*, Sept. 2002.

[14] C. Lin, P. Marti, S. A. Brandt, S. Banachowski, M. Velasco, and J. M. Fuertes. Improving control performance using adaptive quality of service in a real-time system. In *Work in Progress Proceedings of the Real-Time Technology and Applications Symposium (RTAS04)*, May 2004.

[15] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, Jan. 1973.

[16] P. Marti, G. Fohler, K. Ramamritham, and J. 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)*, Dec. 2002.

[17] P. Marti, J. M. Fuertes, G. Fohler, and K. Ramamritham. Jitter compensation for real-time control systems. In *Proceedings of the 23nd IEEE Real-Time Systems Symposium (RTSS 2002)*, Dec. 2001.

[18] H. Rehbinder and M. Sanfridson. Integration of off-line scheduling and optimal control. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, pages 137–143, Stockholm, Sweden, June 2000.

[19] D. Seto, J. Lehoczky, L. Sha, and K. Shin. On task schedulability in real-time control systems. In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS 1996)*, Dec. 1996.

[20] K. G. Shin, C. Krishna, and Y.-H. Lee. A unified method for evaluating real-time computer controllers and its application. *IEEE Transactions on Automatic Control*, 30(4):357–366, 1985.

[21] K. G. Shin and C. Meissner. Adaptation adn graceful degradation of control system performance by task reallocation and period adjustment. In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pages 29–37, 1999.

[22] M. Velasco, J. Fuertes, and P. 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, Dec. 2003.

[23] J. Yepez, J. Fuertes, and P. 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, Dec. 2003.

[24] Q. Zhao and D.-Z. Zheng. Stable and real-time scheduling of a class of perturbed hybrid dynamic systems. In *IFAC World Congress*, pages 91–96, 1999.