

# Draco: Efficient Resource Management for Resource-Constrained Control Tasks

Pau Martí, *Member, IEEE*, Caixue Lin, Scott A. Brandt, *Senior Member, IEEE*, Manel Velasco, and Josep M. Fuertes, *Senior Member, IEEE*

**Abstract**—In many application areas, including control systems, careful management of system resources is key to providing the best application performance. Traditional control systems with multiple control loops statically allocate a fixed portion of the system resources to each controller based on their average or worst-case resource requirements. However, controllers' resource needs vary depending on the jobs they perform and the state of the systems they control. A controller of a plant operating close to its equilibrium requires fewer resources than a controller of a plant operating far from its equilibrium point. The Draco dynamic rate control system exploits this fact by dynamically allocating resources to control systems based on system state. Our research demonstrates that Draco provides significantly better overall control performance with much less resources than static controllers. Our experimental evaluation shows that in the control scenarios we examined Draco provides up to 25 percent better control performance with 30 percent less resources.

**Index Terms**—Resource management, real-time control, control performance.

## 1 INTRODUCTION

MODERN embedded systems are expected to provide both highly varied functionality and outstanding application performance within the available resources. As with most computing domains, fully exploiting system resources is crucial to maximizing embedded control system performance. However, traditional embedded control systems work *open-loop*, using a priori characterizations of the expected workload to determine appropriate static resource allocations using worst-case assumptions. This is done, for example, in the canonical work on uniprocessor real-time scheduling by Liu and Layland [1], and in more recent approaches to control and uniprocessor real-time systems [2].

Static open-loop resource allocation policies work well, guaranteeing that control systems can meet given control performance specifications by reserving a fixed portion of the system resources for each controller. But because this is done statically, open-loop systems cannot account for variations in either resource availability or the resource needs of the controllers. Dynamic closed-loop resource allocation would be beneficial in many situations, for example:

- In power-limited systems, controlling the voltage and clock frequency allows regulation of processor power consumption and potentially lower energy

• P. Martí, M. Velasco, and J.M. Fuertes are with the Department of Automatic Control, Technical University of Catalonia, Edifici U-UPC, Pau Gargallo 5, 08028 Barcelona, Spain. E-mail: {pau.marti, manel.velasco, josep.m.fuertes}@upc.edu.

• C. Lin and S.A. Brandt are with the Department of Computer Science, University of California Santa Cruz, 1156 High Street MS:SOE3, Santa Cruz, CA 95064. E-mail: {lxc, scott}@cs.ucsc.edu.

Manuscript received 15 May 2007; revised 6 May 2008; accepted 7 July 2008; published online 6 Aug. 2008.

Recommended for acceptance by M. Gokhale.

For information on obtaining reprints of this article, please send e-mail to: [tc@computer.org](mailto:tc@computer.org), and reference IEEECS Log Number TC-2007-05-0166. Digital Object Identifier no. 10.1109/TC.2008.136.

consumption. For digital controllers implemented as periodic tasks, a variation in processor speed must be accounted for in each control task execution [3]. Therefore, the system must support runtime adaptation in response to time-varying processor speeds.

- In open real-time systems, graceful degradation can be achieved in overload conditions by modifying task rates to bring the load to a desired value. For digital controllers, the variation applied to each task's rate of progress determines the specific controller that has to be executed [4]. Again, tracking variations on available resources is the key to providing robust implementations.
- In real-time control systems, an efficient way to quickly react to perturbations is to dynamically schedule controllers using flexible control timing constraints. In this way, scheduling decisions are taken based on control information rather than fixed timing constraints such as periods and deadlines [5]. Therefore, tracking the dynamics of the control applications becomes crucial for an efficient utilization of the system resources.

Therefore, modern real-time systems concurrently executing control tasks require dynamic adaptive resource management techniques capable of accommodating changes in the available resources and in application demands, that is, working in *closed-loop*. Achieving this requires

- monitoring mechanisms at both the system resource level and the application level aimed at providing *feedback* information to the real-time resource manager,
- effective resource management policies that make use of this feedback information, and
- digital controllers capable of executing at different rates.

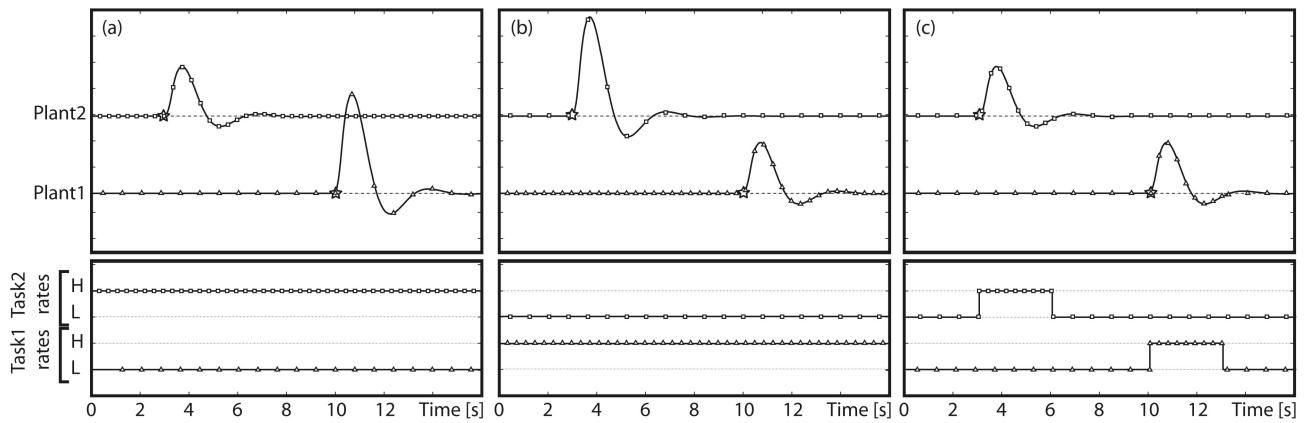


Fig. 1. (a) and (b) Controllers without Draco dynamic rate control. (c) Controllers with Draco.

Our system, the Draco dynamic rate control system, addresses all of these needs, providing

1. end-to-end monitoring of application state and system resource availability,
2. mechanisms allowing communication of this information between applications and the underlying operating system,
3. operating system mechanisms to control and dynamically adjust the resources provided to different tasks,
4. dynamic controllers capable of executing correctly at different rates, and
5. effective policies to dynamically adjust application resources based on knowledge of resource availability and application state.

### 1.1 A Motivating Example

A closer look at the behavior of control loops and at the relation between control performance and controller execution rate suggests that open-loop resource allocation may not be optimal for maximizing control performance when computing resources are limited. A controller whose controlled system or plant is in equilibrium (a *stabilized* plant) may not require the assigned execution rate while another experiencing a perturbation may stabilize more quickly and with less error if executed with a higher rate [5]. In a resource-constrained system running both controllers, redistributing system resources at runtime according to the state of the controlled plants is the key to maximizing overall control performance.

Fig. 1 illustrates three possible resource allocation strategies for this scenario, in which two controllers are executing concurrently in a resource-constrained system. Figs. 1a and 1b show two static open-loop resource allocation strategies, and Fig. 1c shows a simple closed-loop resource allocation policy. The lower portion of each subfigure shows the resources allocated to each of the two controllers, while the upper portion shows the resulting plant dynamics, i.e., the resulting control performance. Those transient responses in the figure that have a smaller maximum deviation from the origin exhibit better performance. The results shown are actual results from executions of the Draco system running real tasks controlling simulated

inverted pendulums, whose model is described in detail in Section 6.1, subject to perturbations at times indicated in the subfigures with stars. For simplicity, each task  $i$  can execute one of two controllers, one at a lower rate  $L$  and the other with a higher rate  $H$ . The lower rate controller is characterized by a long sampling interval and correspondingly lower resource consumption, while the higher rate controller is characterized by a short sampling interval and correspondingly higher resource consumption.

Due to resource limitations, suppose that the resources available at runtime allow only one task to run at a higher rate and the other at a lower rate.<sup>1</sup> Without further information about, for example, perturbation frequencies, the best static open-loop resource allocation policy is to choose the higher rate controller for Task 1 and the lower rate controller for Task 2 (as shown in Fig. 1a) or vice-versa (as shown in Fig. 1b). These two choices are equal in terms of control performance and resource utilization. However, if we know that during transients an increase in the rate of the controller can decrease system deviation and hasten system recovery, better control performance can be achieved by dynamically reallocating resources at runtime according to the plant dynamics (as shown in Fig. 1c). Not only does this reduce system error, but because during some time intervals neither of the two tasks is running at the higher rate, this scheme also reduces overall resource usage, which can, for example, reduce energy consumption in a system capable of dynamic voltage adjustment.

This simple (and ideal) example raises several additional questions:

- Provided that resources are available to run multiple controllers above their minimum effective rate, what is the best policy for allocating resources among multiple concurrently executing control tasks? We are concerned with how good the policy is in terms of both improving control performance and saving resources. At the same time, its implementation

1. This admittedly simple example is nonetheless realistic because digital controllers often run better—but consume more resources—with shorter sampling intervals. A simple way to leverage this is via two alternative intervals: one that uses few resources while awaiting perturbations and another that responds effectively when perturbations occur. We explore less trivial scenarios later in this paper.

must be simple in order to avoid introducing unacceptable overhead.

- Are existing proportional fair share algorithms [6] or water-level fair share algorithms [7] sufficient or do we need algorithms specific to this problem?<sup>2</sup>
- Should different resource allocation policies be applied depending on the frequency of the perturbations affecting the controlled plants? (Note that in our simple example, perturbations do not overlap.)

## 1.2 Contributions and Summary

The contributions of this paper are the answers to all of the questions raised above. Specifically, we have developed Draco, a dynamic resource allocation system for control tasks based on *feedback* information from the controlled plants. Draco demonstrates the feasibility of runtime dynamic resource allocation based on feedback about control task performance and provides efficient and effective algorithms for doing so. Our results demonstrate that Draco outperforms existing static and dynamic solutions.

In addressing the questions, we first define the resource allocation problem as an optimization problem maximizing control performance whose solution determines the best rates for control tasks according to the plant dynamics. The optimization problem has to be repeatedly solved at runtime. To do so with minimal computational overhead, Draco employs an efficient nearly optimal algorithm for the runtime redistribution of computational resources. Near-optimality comes from the fact that our algorithm computes the exact solution of the optimization problem which approximates the relation between tasks rates and control performance in order to yield a low-overhead feasible solution. Despite this approximation, our results show that our near-optimal policy outperforms previous policies for allocating resources to control tasks.

We overcame two key technical challenges in implementing the optimal policy and two alternative feedback-based resource allocation policies. First, we developed and implemented controllers capable of running with different sampling frequencies given different resource allocations. Second, we developed and implemented a flexible real-time platform capable of dynamically changing the allocated resources according to the application feedback while guaranteeing tasks' timing constraints [8]. Together, these two solutions provide the infrastructure needed to develop dynamic resource management for control tasks.

After implementing and integrating these components, we performed extensive experiments including a comparative analysis with the static resource management approach traditionally used in real-time control systems. The experiments we present 1) corroborate that the dynamics of the controlled plants are the key to better exploiting system resources and improving control systems performance, 2) confirm our theoretical approach, and 3) show that our state feedback policies improve control performance and/or save system resources while introducing negligible overhead. In the best case, our experimental evaluation

shows that Draco improves control performance by 25 percent using 30 percent less resources.

The rest of this paper is organized as follows: Section 2 discusses related research. In Section 3, we specify the system model and define the problem to be solved. Section 4 provides the theoretical solution to the resource management for control tasks. In Section 5, we discuss our implementation, and in Section 6, we present the experimental results. Section 7 presents simulation results that indicate that our approach outperforms previous work. Section 8 concludes this paper.

## 2 RELATED WORK

Optimization of control systems performance subject to resource constraints has been examined before for various resources including CPU, network, and batteries. Seto et al. [2] optimized for uniprocessor systems task frequencies at the design stage in order to minimize a control performance index defined over the task set. Similarly, Hong presented a scheduling algorithm to determine data sampling times, so that the control performance requirements are satisfied while increasing network utilization [9]. Rehlinger and Sanfridson [10] presented an offline scheduling method for uniprocessor systems based on optimal control theory. Branicky et al. presented a static scheduling optimization approach based on scheduling and control co-design in networked control systems [11]. Ling and Lemmon presented a method for obtaining specifications on real-time schedulers for networked control systems that assure overall feedback system performance [12]. None of them examined dynamic runtime resource management for optimization of control systems performance, as we do.

Different approaches to control systems and runtime resource allocation policies have also been examined before. Beccari et al. [13] presented a scheduling technique for adaptation of soft real-time load in the context of autonomous robot control architectures. Ramanathan [14] presented an overload management for control tasks based on the  $(m, k)$ -firm guarantee. Caccamo et al. [15] allowed varying tasks' computation times at runtime to optimize control performance using server approaches. Eker et al. [16] presented a method for distributing computing resources over a set of real-time control loops using feedback information on the workload changes. Cervin et al. [17] extended that solution by adding a feed-forward path based on tasks execution times measurements. Similarly, the scheduling method presented by Park et al. [18] allows for sampling period adjustment to allocate network bandwidth to other type of messages. Buttazzo et al. [4] presented a method for promptly react to overload conditions, while still guaranteeing a given control performance. Liu et al. [19] presented a method where control task periods are adjusted using a Markov chain formalism in order to achieve a quality-of-service (QoS) constraint expressed in terms of job dropouts. However, none of the previous work uses feedback from the controlled systems dynamics in order to reallocate resources as we do.

Our approach has some similarities to the various feedback scheduling architectures presented by Zhao and Zheng [20], Walsh and Ye [21], Henriksson et al. [22], Lee and Kim [3], Henriksson and Cervin [23] (which was

2. Conventional proportional fair share (also known as weighted fair share) algorithms allocate resources to applications in proportion to their relative weights or importance. The water-level fair share (aka max-min fair share) algorithm allocates resources to tasks in order of increasing demand.

further extended by Castañé et al. [24]), and Ben Gaid et al. [25]. Zhao and Zheng [20] discussed an event feedback scheduling strategy in which controllers are executed according to the dynamics of the controlled systems. They aimed to meet asymptotic and exponential stability criteria while our goal is to optimize aggregate control performance. Walsh and Ye [21] presented a dynamic arbitration technique to grant network access to the control loop with highest error. Our approach is similar, but focuses directly on resource allocation rather than on priority assignment. Henriksson et al. [22] presented a scheduler that allocates CPU time to model predictive controllers (MPC) in order to improve control performance by dynamically varying controllers execution times. Our approach is not restricted to MPC and the control optimization is achieved by dynamically adjusting the sampling interval of each controller. Lee and Kim [3] presented a dynamic solution for establishing the optimal processor speed for a set of control tasks based on the state of the controlled plants, an ad hoc solution that lacks theoretical foundation. Henriksson and Cervin [23] (which was further extended by Castañé et al. [24]) presented an algorithm to optimally assign sampling periods to control tasks. However, in the general case, their approach does not have an analytical solution and so a suboptimal solution has to be applied. None of these considers a static base allocation of resources to control tasks—guaranteeing a minimum level of responsiveness to perturbations—enhanced with the resource redistribution, as we do; they consider redistributing all of the resources at each resource allocation. Ben Gaid et al. [25] present a dual method in which resources are statically allocated to control tasks producing optimal offline cyclic sequences of jobs and then, at runtime, specific precomputed sequences are applied subject to the restriction that the current sequence has to complete before applying the next one. This, for a large number of applications, may result in less reactive systems. Removing this restriction would increase the computational overhead. Therefore, this paper does not analyze the feasibility of the online approach in a complex scenario as we do.

Unlike many previous approaches, the feasibility of our solution and its benefits have been verified through its implementation in a real operating system and under complex scenarios. This work consolidates our preliminary research on state feedback resource allocation for real-time control systems [26].

### 3 PROBLEM FORMULATION

In this section, we describe the system architecture and formalize the resource allocation problem to be solved.

#### 3.1 System Architecture

Traditional real-time CPU schedulers serve two roles in CPU resource management: *resource allocation* and *dispatching*. Resource allocation determines how much of the resource to give each task, while dispatching determines when to give the allocated resources to each task. Dynamically and independently managing resource allocation and dispatching (RAD—resource allocation/dispatching [8]) allows precise allocation of resources to the possibly dynamically changing needs of individual tasks.

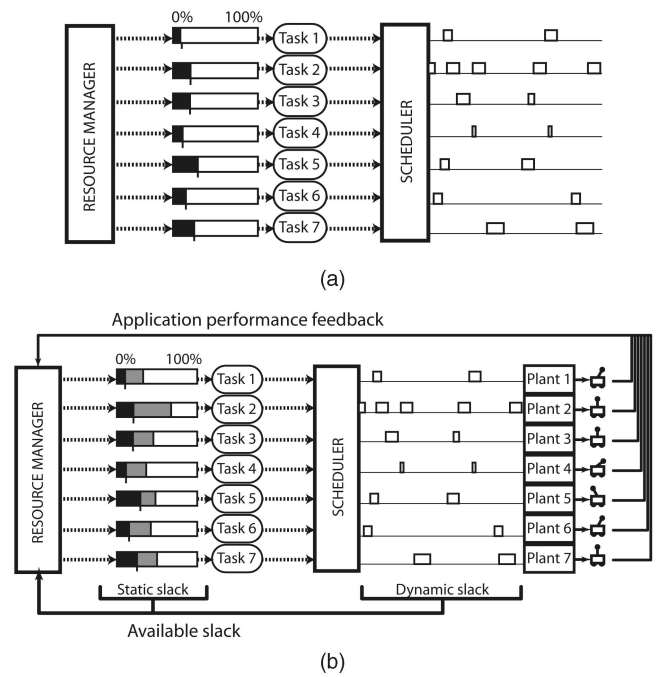


Fig. 2. RAD model. (a) Without slack management. (b) With slack management.

Draco employs the RAD model for its underlying resource management and is built on top of the Rate-Based Earliest Deadline (RBED) scheduler [8]. It allocates a baseline amount of resources to each control task and then dynamically reallocates *slack*, or excess resources. We are not concerned here with differentiating between *static slack*—unreserved resources—and *dynamic slack*—reserved but unused resources. Nor are we concerned with how to compute it or on when to allocate it. See [27] and the references therein for work on slack management addressing these and related questions.

Fig. 2a illustrates the feedback-based RAD resource management in the Draco system. Resource allocation allows us to tailor the resource demands (black/gray boxes) for each task at runtime based on feedback from the controlled plants, while dispatching addresses when the assigned resources (white boxes) can be used for each task in order to meet the required sampling intervals for each running control task.

Our approach to slack management focuses on the allocation resources to control tasks. As illustrated in Fig. 2b, we assume that each control task has a minimum resource share assigned, which is represented as a black box. In addition, each task can be assigned a portion of available slack, which is represented as a gray box. The dispatching can be performed by any existing policy, represented by white boxes (see Buttazzo [28] for an extensive survey of dispatching policies). Fig. 2b also shows the key components of the slack management framework. Information about each plant’s dynamics (*Application performance feedback*) is fed back to the resource allocator in order to provide the information required to make an appropriate slack redistribution decision. Finally, the current resource utilization (or alternatively, the current

static and dynamic slack in the system) is fed back to the resource manager.

The basic operation of the feedback architecture can be summarized as follows: at each control loop operation (which occurs once per period), the controlled system is sampled and this information is used by the control algorithm to compute and output the control signal. Each control signal affects the controlled system dynamics in such a way that the system is driven toward its specified *set point* (also called its *equilibrium point*). Each sample will show whether the controlled system is affected by a perturbation or not. This information is fed back to the operating system in order to allow the system to reallocate the available slack accordingly.

### 3.2 Problem to be Solved

The system we consider is a real-time system executing  $n$  control tasks, each one controlling a plant in a closed loop and competing for the computing resources (i.e., sharing a single CPU). The real-time system may also execute other hard, soft, and non-real-time tasks (as supported by RBED). Control tasks are implemented such that each execution sequentially performs sampling, control algorithm computation, and actuation. We assume a discrete time model [29]. An external observer counts the ticks of the globally synchronized clock (which is based on the metrics of the physical second, as a time unit) with granularity of length  $g$  and assigns natural numbers from 0 to  $\infty$  to all task timing constraints defined as multiples of  $g$ .

Each controlled plant  $i$  can be described by the linear time-invariant system equations of a continuous-time system in state space form [30]:

$$\dot{x}_i(t) = A_i x_i(t) + B_i u_i(t), \quad (1)$$

$$y_i(t) = C_i x_i(t), \quad (2)$$

where  $x_i(t)$  and  $y_i(t)$  are the state vector and the output of the system at time  $t$ ,  $u_i(t)$  is the control signal applied to the system at time  $t$ , and matrices  $A_i$ ,  $B_i$ , and  $C_i$  are the system, input, and output matrices of appropriate dimension. The state (1) and output (2) 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. The objective of the control is to keep the output  $y(t)$  as close to zero as possible, despite the presence of perturbations.

Each control task is characterized by its period  $h_i$  and its exact (and constant) execution time  $c_i$ . The period corresponds to the sampling interval. The execution time includes the time required to sample the state, compute the control algorithm, and output the actuation command. Relative deadlines are assumed to be equal to the periods. The assumption that we know the exact execution time for control tasks is valid as control task execution times can be completely assessed. Control tasks implementing classically designed controllers execute a sequential code, with no conditional or loop sequences. For example, the code of a proportional-integral-derivative (PID) controller or the code of a state feedback controller is completely sequential (see code examples in [31]). In this case, an exact execution time

rather than a worst case can be calculated. Therefore, for control tasks, we can assume that the actual execution time is constant. Cache memories and pipelined processors are common features in modern computer architectures that can introduce significant variability into task execution time. Embedded real-time systems usually do not have such features so that task execution times in embedded systems would not exhibit large variations. In the unlikely case where control tasks execution time variations would occur, the presented approach would still hold if instead of rates the analysis were done with frequencies.

The partial utilization factor of each task, also called *rate*,  $r_i$  (3), is the CPU share (resource requirement) that each control task requires for a given period. Since the worst-case execution time of each control task is constant, any variation in task rate implies a corresponding variation in task period (and vice-versa):

$$r_i = \frac{c_i}{h_i}. \quad (3)$$

We assume that a minimum rate (4) is guaranteed to each control task, which is given by the longest task period and causes the slowest execution:

$$r_{i,min} = \frac{c_i}{h_{i,max}}. \quad (4)$$

If slack is available, the problem to be solved is to decide for each control task how the rate should be increased:

$$r_i = r_{i,min} + \Delta r_i \quad (5)$$

(i.e., how the period should be shortened) such that overall control performance is improved, considering the dynamics of the controlled plants. Since resources are limited, not all task rates can be increased and not all rate increases provide the same benefit. Therefore, solving the problem involves trade-offs and compromises.

### 3.3 Formulation of the Optimization Problem

Solving the problem requires establishing relations between control performance and task rates in order to allow direct comparison of control loops under different rate assignments. For a given plant (1) and (2), and for a given sampling period  $h$ , it is straightforward to obtain the relation between control performance and sampling period. The discrete-time representation of systems (1) and (2), obtained with a sampling period  $h$ , is<sup>3</sup>

$$x(kh + h) = \Phi(h)x(kh) + \Gamma(h)u(kh), \quad (6)$$

where

$$\Phi(h) = e^{Ah} \quad \text{and} \quad \Gamma(h) = \int_0^h e^{As} ds B. \quad (7)$$

If the control signal  $u(kh)$  is given by a state feedback control law  $L(h)$  designed using standard methods

$$u(kh) = -L(h)x(kh), \quad (8)$$

3. The  $i$ -subscript is omitted to simplify the notation.

then the closed-loop plant dynamics will have the form

$$x(kh + h) = (\Phi(h) - \Gamma(h)L(h))x(kh) = \Phi_{cl}(h)x(kh), \quad (9)$$

where  $\Phi_{cl}(h)$  is the closed-loop matrix.

Equation (9) indicates that the dynamics of each closed-loop system changes depending on the sampling period  $h$  and therefore on the assigned rate  $r$ . That is, for each  $i$ th control loop, we can define a performance function  $g_i(r_i, x_i(t))$  that establishes the relation between control performance and rate.

To evaluate control performance, many of the existing approaches to control systems and online resource allocation policies, for example those of Eker et al. [16], Cervin et al. [17], Henriksson and Cervin [23], Castañé et al. [24], and Ben Gaid et al. [25], use quadratic cost functions on the states and control signals,  $J(h)$ , such as

$$J(h) = \mathbb{E} \left( \sum_{k=0}^{N-1} \begin{bmatrix} x^T(kh) & u^T(kh) \end{bmatrix} Q \begin{bmatrix} x(kh) \\ u(kh) \end{bmatrix} + x^T(Nh)Q_{0c}x(Nh) \right), \quad (10)$$

where  $Q$  and  $Q_{0c}$  are weighing matrices (see [30] for further details). These works show that these cost functions, which are primarily used to design optimal controllers minimizing  $J$ , can be also used to address the resource allocation problem. In particular, their formulation is a schedulability problem in which sampling periods were selected to solve the optimization problem:

$$\begin{aligned} & \text{Minimize :} && \text{Penalty on control performance} \\ & \text{with respect to :} && \text{Sampling periods} \\ & \text{subject to :} && \text{Task set schedulability} \\ & && \text{(and Closed loop stability).} \end{aligned} \quad (11)$$

However, their common conclusion is that in the general case, the computational overhead of solving problems formulated in terms of cost functions like (10) cannot be handled by an online resource manager because no general analytical solutions can be found. To overcome this limitation, a different approach is required.

Alternatively, in traditional real-time (QoS) scheduling approaches, the goal is to allocate resources to tasks at runtime in order to maximize application benefit, for example those of Lee et al. [32], Burns et al. [33], Aydin et al. [34], Brandt and Nutt [35], Hegazy and Ravindran [36], and Chen et al. [37]. Some of these approaches define value or performance functions that approximate application benefit with respect to task rates and then formulate the schedulability or allocation problem as the optimization problem:

$$\begin{aligned} & \text{Maximize :} && \text{Overall application benefit} \\ & \text{with respect to :} && \text{Task rates} \\ & \text{subject to :} && \text{Task set schedulability} \\ & && \text{(and Quality-of-service constraints).} \end{aligned} \quad (12)$$

A primary limitation of these approaches is that the optimized performance functions are approximates of the

application utility. In addition, their applicability to control systems has not been explored.

Borrowing the formulation given by traditional QoS approaches, we define the slack redistribution problem for control tasks in the form of (12). The main goal in using this formulation is to obtain a specification of the optimization problem 1) allowing analytical and general closed form solutions with negligible computational overhead and 2) showing that its applicability to control systems is not compromised by using approximate performance functions.

If each controller is independent—controlling an independent plant (as we assumed in the system model in Section 3.2)—the objective function can be considered as the sum of all individual benefits obtained by each controller, i.e.,  $g_i(r_i, x_i(t))$ , possibly weighted by  $w_i$  to provide a mechanism allowing appropriate comparisons among the control loops in the system. Then, the slack allocation problem can be formulated as the following optimization problem:

$$\text{Maximize} \quad \sum_{i=1}^n w_i g_i(r_i, x_i(t)), \quad (13)$$

$$\begin{aligned} & \text{subject to} \quad \sum_{i=1}^n \Delta r_i \leq U_s(t) \\ & \quad \Delta r_i \geq 0, \end{aligned} \quad (14)$$

where the solution is a vector  $\Delta \vec{r} = [\Delta r_1, \Delta r_2, \dots, \Delta r_n]$  that maximizes the control performance delivered by the set of controllers running with rates specified by (5), restricted to the utilization feasibility constraint (14).  $U_s(t)$  is the feasible resource utilization due to the available (dynamic and static) slack. Note that  $\forall t U_s(t) \geq U_{min}$ , where  $U_{min} = \sum_{i=1}^n r_{i,min}$  is the minimum utilization guaranteed for all control tasks when executing with their longest period.

Note that the formulation of the optimization problem in the form of (12) does not ensure system stability as it does in (11). Consequently, closed-loop system stability should be checked in a separate stage.

## 4 RESOURCE MANAGEMENT

The set of objective functions  $g_i$  should 1) establish a relation between control performance and task rates and 2) enable determination of optimal task rates based on the feedback information received from the plants, as indicated by the motivating example. Depending on the objective function and the time-varying restrictions, solving the optimization problem may not be feasible for an online real-time resource manager.

### 4.1 Customization of the Optimization Problem

Let

$$g_i(r_i, x_i(t)) = e_i p(r_i) \quad (15)$$

be the structure of the performance functions relating control performance and task rates that we consider, where  $e_i$  is defined as

$$e_i = |x_i(t)|, \quad (16)$$

where  $|\cdot|$  is a suitable norm. Therefore, the benefit that can be achieved for a given rate is linearly rescaled by the controlled plant error,  $e_i$ , and the benefit function  $p_i(r_i)$  can be any function on the task rate.

**Observation 1.** *Without loss of generality, if we consider the equilibrium point of a controlled system to be zero, the norm of the state vector,  $|x_i(t)|$ , is the distance of each controlled system from its equilibrium point at any given time  $t > 0$ . If, for a given control loop, all states cannot be measured, they can be determined from the available measurements and a model [30]. This measure tells how critical the situation is for each controlled system; the higher its value, the worse the system. This measure is the feedback information that each controller, at each sample, will send to the operating system for the resource redistribution.*

Although control performance functions (15) have been defined without any control formalism, they can be directly mapped to functions like (10) (see Appendix A for further details). Moreover, their open structure gives a general framework which allows evaluation of other aspects, for example penalties on the use of resources.

Each benefit function  $g_i$  is based on an instantaneous measure of the plant dynamics,  $e_i$ , rather than a prediction. This simplifies complexity and its evaluation cost can be negligible. In addition, its application converts the time-varying restriction into a time invariant one because the only required information is the current available slack. See Castañe et al. [24] for a detailed discussion of the *pros* and *cons* of both types of objective functions.

Any type of function  $p_i(r_i) \in C^\infty$  can be expressed by its infinite Taylor series around the working point  $r_{i,min}$ :

$$p_i(r_i) = p_i(r_{i,min}) + \frac{p_i'(r_{i,min})}{1!}(r_i - r_{i,min}) + \frac{p_i''(r_{i,min})}{2!}(r_i - r_{i,min})^2 + \dots \quad (17)$$

Considering (5), (17) is given by

$$p_i(r_i) = p_i(r_{i,min}) + \frac{p_i'(r_{i,min})}{1!}(\Delta r_i) + \frac{p_i''(r_{i,min})}{2!}(\Delta r_i)^2 + \dots \quad (18)$$

Therefore, from (18), a simple linear function (19) on the task rate increments can be used to approximate the relation between different rates and control performance in the objective function  $g_i$ :

$$p_i(r_i) \approx \alpha_i \Delta r_i + \beta_i, \quad (19)$$

where parameters  $\alpha_i$  and  $\beta_i$  are specific for each control loop and performance function but known prior runtime.

The linear approximation (19) is a simplification that enables closed analytical solutions to the optimization problems. This in turn allows Draco to meet the bounded computational overhead requirement on policies implemented by an online real-time resource manager. The relation (19) covers a wide class of control systems, including the behavior of linear and nonlinear systems around the working point, and therefore, for a given range of sampling periods. Since the formulation of the optimization problem makes use of an approximation, the optimal

solution to the formulated problem is potentially suboptimal for some systems. Nevertheless, our results show that the performance achieved is better than other approaches for the scenarios we examined.

## 4.2 Optimal State Feedback-Based Slack Redistribution Policy

Formally, at the system level, each control task  $\tau_i$  can be characterized by its rate  $r_i$  (which is a *system* characterization), its benefit function  $p_i$  (which relates *system* resources and *control* performance), and its controlled system error  $e_i$  (which is a *control* characterization), as represented by

$$\tau_i = \{r_i, p_i, e_i\}. \quad (20)$$

By appropriately rewriting the optimization specified in (13) and (14) considering (15), for a given set of  $n$  control tasks,  $\tau_1, \dots, \tau_n$ , the problem to be solved is

$$\text{maximize} \quad \sum_{i=1}^n w_i e_i p_i(r_i), \quad (21)$$

$$\text{subject to} \quad \sum_{i=1}^n \Delta r_i \leq U_s \\ \Delta r_i \geq U_s, \quad (22)$$

where the solution will be the rate increments  $\Delta r_i$ ,  $i = 1, \dots, n$ , such that all the tasks are schedulable and the overall control system performance is maximized. The solution to the optimization problem (21) and (22) depends on each benefit function  $p_i(r_i)$ .

Taking into account the approximation for  $p_i(r_i)$  given in (19), the optimization problem (21) and (22) is linear and convex. Therefore, it can be solved by any available optimization technique such as the interior point methods or the classical Karush-Kuhn-Tucker conditions [38]. The solution is given next:

**Theorem 1.** *The optimal solution  $\Delta \vec{r} = [\Delta r_1, \Delta r_2, \dots, \Delta r_n]$  of the optimization problem (21) and (22), where  $p_i(r_i)$  are given by (19), is  $\Delta \vec{r} = [0, 0, \dots, 0, \Delta r_i = U_s, 0, \dots, 0]$ ,  $i \in [1, \dots, n]$  such that  $w_i e_i \alpha_i$  is maximum  $\forall i \in [1, \dots, n]$ , if the set of control tasks is described by (20).*

**Proof.** The proof is provided in our previous work [26].  $\square$

**Observation 2.** *In terms of slack redistribution, the theorem states that we should assign all available slack (that is,  $U_s$ ) to the control task with maximum  $w_i e_i \alpha_i$ . In the specific case where all functions  $p_i$  and all weights  $w_i$  are the same (i.e., the controlled plants are equal and of equal importance), we should assign all of the available slack to the control task with the largest error  $e_i$ .*

**Observation 3.** *The optimal solution holds if the approximation given by (19) is valid around the working point. Whenever the approximation is not good enough, a higher order approximation should be considered.*

**Observation 4.** *If the  $p_i$  approximations are polynomial functions on  $r_i$  of degree higher than one but still less than five [39], an analytical solution to the optimization problem (21) and (22) can still be found turning the solution into a feasible algorithm (in terms of computational complexity) for a*

runtime resource manager. In this case, the theorem should be restated taking into account the new  $p_i$  form.

**Observation 5.** It is possible to generalize (16) so that  $e_i = \eta|x_i|$ , where  $\eta: \mathbb{R} \rightarrow \mathbb{R}$  is a class  $\mathcal{K}$  function of the normed state. However, the optimal policy (as identified in Theorem 1) will no longer have an all-or-nothing quality.

The optimal policy determines control task rates to maximize aggregated control performance. However, stability analysis is not addressed. Therefore, our approach requires an additional step to check system stability.

### 4.3 Controller Design and Stability

The application of the optimal policy requires the implementation of controllers capable of running with different sampling frequencies given different resource allocations (as explained by Martí et al. [31], [5]). In particular, all of the controllers will run at  $h_{i,max}$  (given by the guaranteed minimum rate,  $r_{i,min}$ ), and they will also run at  $h_{i,min}$  (given by  $r_{i,min} + \Delta r_i$ ) if mandated by the optimal policy, where

$$h_{i,min} = \frac{c_i}{r_{i,min} + \Delta r_i}, \quad (23)$$

with  $\Delta r_i = U_s$ , which is a constant value in the optimization problem. But, because the available slack  $U_s$  may vary at each optimization problem resolution,  $h_{i,min}$  will vary over time. As a result, the sampling intervals that can apply for a given controller under this approach belong to a range of discrete values:

$$h_i \in [h_{i,min}^*, \dots, h_{i,max}], \quad (24)$$

where the highest value  $h_{i,max}$  is given by the minimum guaranteed rate, the lowest value  $h_{i,min}^*$  corresponds to the minimum sampling period due to the highest slack availability, which is a bounded value that can be obtained before runtime. The intermediate values are given by the clock granularity length,  $g$ .

Therefore, each controller implemented within a task will execute with a runtime period that belongs to the specified range. We design stable 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 [30]. At runtime, controller gains are switched according to the current period that applies (following the techniques presented by Wittenmark and Åström [40] and Albertos and Salt [41]).

In the end, each control signal (8) is given by a control law that is an algorithm on the sampling period,  $L_i(h_i)$ . Looking at the closed-loop dynamics of each control loop (9), having different  $h_i$  means to have a system evolution determined by different  $\Phi_{i,cl}(h_i)$ . However, since we have a discrete range of sampling intervals, we will have a set  $\Sigma_i$  with a finite number of closed-loop matrices:

$$\Sigma_i = \left\{ \Phi_{i,cl}(h_i) \mid h_i \in [h_{i,min}^*, \dots, h_{i,max}] \right\}. \quad (25)$$

Therefore, each closed-loop system evolution will be characterized by an infinite product of closed-loop matrices

taken from  $\Sigma_i$ . In switched systems notation (e.g., that of Liberzon [44]), each closed-loop evolution will be given by

$$x_i(k+1) = \Phi_{i,cl}(h_i)x_i(k), \quad \Phi_{i,cl} \in \Sigma_i, \quad (26)$$

where  $\Phi_{i,cl}(h_i)$  are stable matrices.

Stability for such systems can be studied using multiple existing stability results [42], [43], [44], [45]. For example, Corollary 2 by Dogrueel and Özgüner [42] gives the necessary and sufficient stability condition (27) for such systems in terms of linear matrix inequalities (LMIs):

$$\begin{aligned} \Sigma_i \text{ asymptotically stable} &\leftrightarrow \exists P > 0 : \\ \Phi_{i,cl}^T(h_i) \cdot P \cdot \Phi_{i,cl}(h_i) - P < 0, &\forall \Phi_{i,cl}(h_i) \in \Sigma_i^r, r \geq 1. \end{aligned} \quad (27)$$

The baseline idea of the stability analysis is to calculate all closed-loop matrices that can apply at runtime for all periods. This set of matrices is said to be stable if it fulfills a necessary and sufficient stability test (27).

In Appendix B, a numerical example of stability analysis based on common Lyapunov functions [44] is given.

## 5 IMPLEMENTATION

At the operating system level, dynamic slack redistribution for controllers can be achieved by any existing scheduling framework or scheduling algorithm which supports dynamic task period adjustment at runtime and guarantees no deadline miss during the adjustment. Examples include the RBED scheduler [8], the Elastic Model [46], and the Variable Rate Execution (VRE) model [47], all of which provide hard resource guarantees, isolation between processes, and correct operation during changes in resource allocations (unlike other operating systems such as RTLinux [48], which are based on the traditional static real-time task model with fixed worst-case execution times).

As a prototype and performance demonstration, our feedback architecture is implemented in the RBED-integrated real-time system [8], which supports hard real-time, soft real-time, and best-effort processes. RBED dynamically allocates resources to processes as a percentage of the CPU such that the total allocated is less than or equal to 100 percent and then schedules all processes with the Earliest Deadline First (EDF) algorithm [1]. RBED dynamically changes allocated resources and application periods without violating EDF constraints, guaranteeing that tasks never miss their assigned deadlines. Changes to tasks' periods and resource allocations may be arbitrarily adjusted at task deadlines (i.e., between sampling intervals) while maintaining all real-time guarantees [8].

We implemented the optimal feedback slack redistribution policy (*optimal*, described in Section 4.2) for control tasks in RBED. In order to better demonstrate its benefits, we also implemented two adaptive feedback-based slack management techniques, called *proportional* and *discrete*. Finally, to provide a direct comparison with traditional control system implementations, we also implemented a baseline policy, *static*.

### 5.1 Optimal

The optimal policy implements the optimal solution described in Section 4: it assigns all available slack to the

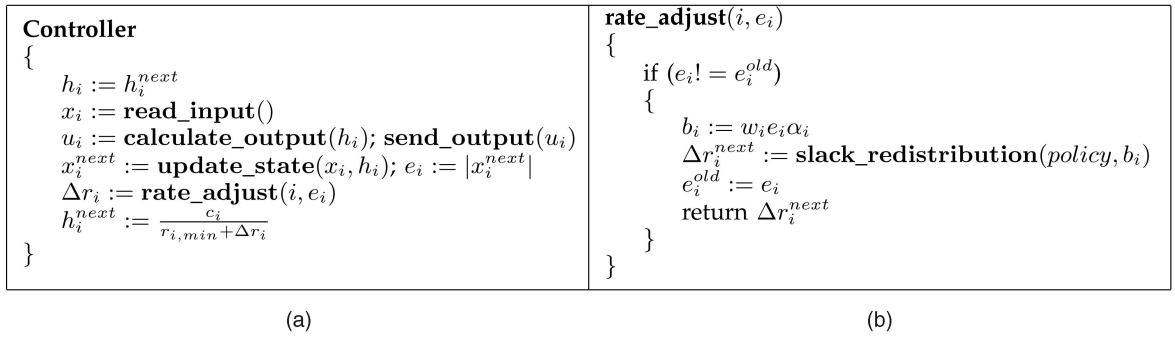


Fig. 3. Pseudocode for controller and dynamic rate adjustment.

controller with the highest  $w_i e_i \alpha_i$ , where  $U_s$  is the available utilization due to slack at the time the optimization takes place:

$$\Delta r_i = \begin{cases} U_s, & \text{if } w_i e_i \alpha_i \text{ is maximum,} \\ 0, & \text{otherwise.} \end{cases} \quad (28)$$

Recall that task rates will be updated according to  $r_i = r_{i,min} + \Delta r_i$ .

The optimal policy needs only to keep track of which task has maximum  $w_i e_i \alpha_i$ . This can be determined in  $O(n)$  time by a linear scan of the list of  $n$  tasks. Reallocation of slack occurs when the task with the maximum  $w_i e_i \alpha_i$  changes or when the amount of available slack changes (for instance because a control task has been started or stopped).

## 5.2 Proportional

The proportional policy represents a “fair” allocation based on measured error and reflects our initial guess at an optimal allocation prior to formally deriving that algorithm. It distributes slack among all control loops such that all tasks get a slack increment proportional to  $w_i e_i p_i$  (similar to fair share approaches):

$$\Delta r_i = \frac{w_i e_i \alpha_i}{\sum_{j \leq n} w_j e_j \alpha_j} \cdot U_s. \quad (29)$$

As before,  $\Delta r_i$  will be added to  $r_{i,min}$  for each control task.

The proportional policy has the same time complexity ( $O(n)$ ) for the slack redistribution as the optimal policy. However, any change to the error of a controlled plant will cause a dynamic slack redistribution so the actual number of dynamic resource reallocations, and thus the introduced overhead of the proportional policy, is greater than that of the optimal policy.

## 5.3 Discrete

The discrete policy is similar to the proportional but for a limited number of discrete task rates. It predefines for each task a set of rate increments which are mapped into benefits, for example, using (19). At runtime, these benefits are rescaled by the norm of the plant state, for example, using (15). With these new benefits, a heuristic algorithm [35], [49] is employed to iteratively increase the rate level of the control tasks until no more increases are possible within the available slack, providing high average overall system benefit. Note that an optimal discrete allocation is NP-hard

in general, based upon a straightforward reduction to the knapsack problem [50].

The worst-case time complexity of the discrete policy is  $O(Ln)$ , where  $L$  is maximum slack increments in the system (which is equal to the maximum number of sampling periods for all control tasks), because the worst case for a dynamic slack redistribution may go through all of the slack increments 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 [49]. In this way, slack redistribution takes place only when the error moves from one range to another. This significantly reduces the overhead (see overhead analysis in Section 6).

## 5.4 Static

The static policy implements the “traditional” approach. It statically allocates the available resources to the controllers only based on static information, which includes  $w_i$  and  $p_i(r_i)$ . The resource allocation algorithm is as follows:

$$r_i = \frac{w_i p_i(r_i)}{\sum_{j \leq n} w_j p_j(r_j)} \cdot U_d. \quad (30)$$

Among the four policies that we implemented, the static policy is the only one that does not use dynamic error  $e_i$  to redistribute slack. Therefore, (30) is computed prior to system runtime, where  $U_d$  is the overall available resource utilization (see further details in Section 6.2), resulting in zero runtime reallocation overhead.

## 5.5 System Interface Implementation Details

Fig. 3 shows the pseudocode for a controller and the dynamic rate adjustment. A controller requests a dynamic rate adjustment through the system call **rate\_adjust()**. The controller (Fig. 3a), with execution period  $h_i$ , does two things each period. First, 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  $h_i$ ). Then, it 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 (possibly new) rate increment that will be used at the following controller execution.

The dynamic rate adjustment (Fig. 3b) uses the specified slack redistribution policy (discrete, proportional, or optimal) to reallocate slack based on  $w_i e_i \alpha_i$ . The static policy is not considered because the complete resource allocation is

performed offline. Therefore, the parameter *policy* determines to execute (28) for the optimal, (29) for the proportional, or the approach described by Lin et al. [49] for the discrete policy.

The rate adjustment requires  $e_i$  from all tasks. Since tasks run at different rates, synchronization is an issue. Two approaches can be considered. First, a pseudosynchronized approach, in which the reallocation is performed considering the new  $e_i$  passed in by the task that triggered the rate adjustment, plus all remaining  $e_j$  (with  $j \neq i$ ) that were updated in previous `rate_adjust()` calls. Second, a fully synchronized approach, in which we assume that the system level procedure that reallocates resources can access all tasks  $e_i$ . The first approach is simple but the rate adjustment is performed using nonfresh data. The second approach uses fresh data, which will translate into a more accurate resource allocation. This can be achieved by accessing the data buffered in the data acquisition board (DAB). Usually, DABs are constantly (and very frequently) sampling all plants, independent of the rate of the control tasks. Therefore, DABs always maintain fresh data. In our experiments, we adopted the second approach.

Although not shown, it is important to point out that in Fig. 3 the procedure `rate_adjust()` adjusts the rate of all controllers, not just controller  $i$ .

## 6 PERFORMANCE EVALUATION

In order to evaluate the performance results, we implemented our four slack redistribution policies (static, discrete, proportional, and optimal) 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. Although we present a few specific cases, our experiments are general and our results are not limited to these or any other special cases. All experiments were performed on a standard Intel-based PC equipped with a 1-GHz Pentium III processor, 512-Mbyte RAM, and a 40-Gbyte hard drive.

### 6.1 Controlled Processes

We simulated two kinds of controlled plants: an inverted pendulum and a ball and beam, both standard benchmarks in the area of control systems. Their relevance is due to fact that they are unstable plants, and feedback control is essential to make them operate safely. Many modern industrial processes and technological systems are intrinsically unstable (e.g., exothermic chemical reactions or control of a rocket or aircraft during vertical takeoff). Since real unstable systems are usually dangerous and cannot be brought into the laboratory, these two systems were developed to resolve this paradox. They are simple, safe and yet they have the important dynamic features of unstable systems.

The linear time-invariant state space model we used for each inverted pendulum mounted on a cart is given by

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

where  $\theta$  is the pendulum angle,  $\omega$  is the angular velocity,  $x$  is the cart position, and  $v$  its velocity. We customized all pendulums as follows: mass of the cart  $M = 2$  kg, mass of the pendulum  $m = 0.1$  kg, length of the pendulum stick  $l = 0.5$  m, and gravity  $g = 9.81$  m/s<sup>2</sup>.

The linear time-invariant state space model for each ball and beam is given by

$$\begin{bmatrix} \dot{\theta} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ x \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t),$$

where  $\theta$  is the beam angle, and  $x$  is the ball position on the beam.

For both plants, we assume ideal operating conditions, and therefore, the models used for controller design are assumed to be free of errors. This assumption allows us to focus our performance analysis on the evaluation of the resource allocation policies, avoiding the interference on performance that modeling errors would have introduced into the analysis.

### 6.2 Workload Generation

In the following experiments, we used two workloads: three controllers, each controlling a simulated inverted pendulum, and three controllers, each controlling a simulated ball and beam. We use uniform workloads in the experiments to simplify the performance analysis and comparison. In this case,  $w_i$  and  $p_i$  are equal for all controllers, which means that the error  $e_i$  is the main driving factor in each policy. Although the numbers would differ, the results would be essentially the same, just harder to interpret, with nonuniform  $w_i$  and  $p_i$ . For the same reasons, we assumed a constant slack availability ( $U_s$ ). This allows a fair evaluation of the static (traditional) policy with respect to the dynamic slack policies.

In both workloads, each controller implements the same parametric control law obtained by standard pole placement, which is parameterized on the sampling interval.

For each of the policies, we ran the three controllers (in each set) for 1 hour and randomly generated perturbations for each controller with different average perturbation intervals. The available slack is defined such that the global resource utilization factor ( $U_d$ ) for the three controllers is  $U_d = 97$  percent; we reserve 3 percent of the resource capacity for the execution of other general-purpose processes. Recall that  $U_s = U_d - U_{min}$  (Section 3.2), where  $U_{min}$  is the minimum utilization factor guaranteed to all control tasks. 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 100 times or many thousands of times over the course of 1-hour-long experiment.

In the first workload, each controller in charge of a simulated inverted pendulum has a fixed worst-case execution time  $c_i$  of 0.0135 second. With either the optimal or proportional policy, each controller can run at any sampling period ( $h_i$ ) within 0.03 and 0.05 second. With the discrete policy, we defined three resource levels corresponding to three different sampling periods  $h_i \in \{0.03 \text{ second}, 0.04 \text{ second}, 0.05 \text{ second}\}$  for each controller. In this case, if there are three control tasks in the system, none of them will

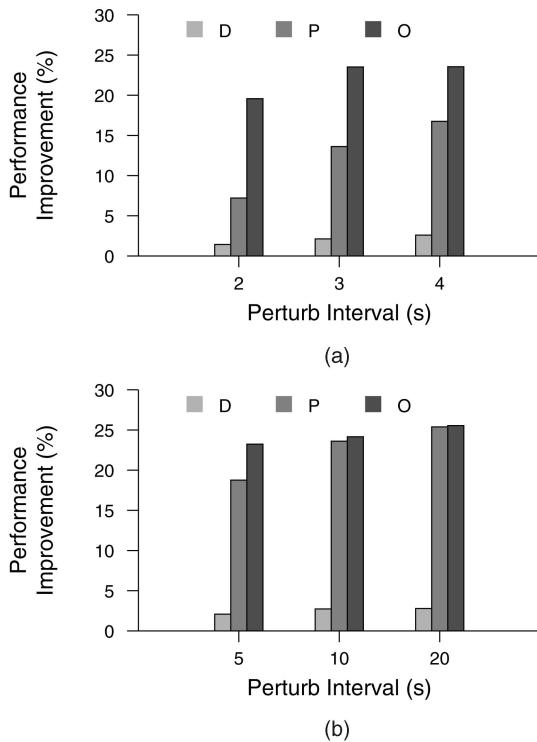


Fig. 4. Inverted pendulum: Performance improvement relative to static. (a) Without idle CPU usage. (b) With idle CPU usage.

execute at their highest level ( $h_i = 0.03$  second) since  $(0.0135/0.03 + 2 \cdot 0.0135/0.05) = 0.99 > 97$  percent  $= U_d$  (that is, the required CPU load would exceed what is available). For the static policy, the three controllers share the available CPU ( $U_d = 97$  percent) equally, and thus, each of them is given  $\frac{97}{3}$  percent of the CPU for the duration of the experiments.<sup>4</sup> In the second workload, all controllers (each in charge of a simulated ball & beam) are slower than those in the first set: the worst-case execution time is 0.135 second and the sampling period ( $h_i$ ) is within 0.3 and 0.5 second.

In our experiments, we assume a noise-free model for our plants. Incorporating noise would require the definition of appropriate thresholds for filtering biased measurements. However, the performance evaluation would not change. For both workloads, we evaluated the control performance of the four different policies by examining the total cumulative error of the three control tasks (i.e.,  $\int_0^{t_e} \sum_{i=1}^3 |x_i(t)| dt$ , where  $t_e$  is the time interval of the experiment).

### 6.3 Performance Analysis

The main contribution of this paper is summarized in Figs. 4 and 5, which show the performance improvement for the three policies (discrete (D), proportional (P), and optimal (O)) relative to the performance of the static policy for different perturbation intervals in 1-hour experiments. For the ball and beam system (Fig. 5), we left out the performance results for the discrete policy; its performance was nearly identical to that of the static policy.

Fig. 4a shows the results when the perturbation intervals are short enough that all of the available CPU is always

4. Note that having constant slack allows us to evaluate the performance obtained by static controllers with a higher rate than if slack were dynamic, thus providing a more challenging performance evaluation scenario.

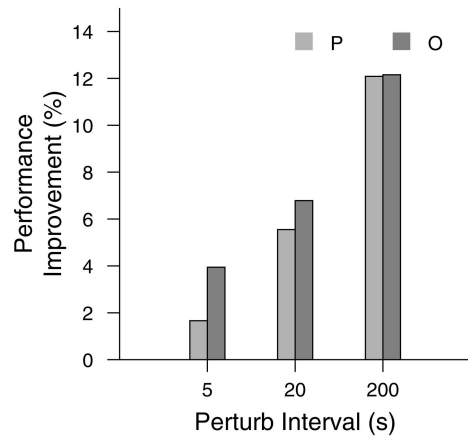


Fig. 5. Ball and beam: Performance improvement relative to static.

allocated to the control tasks (i.e., there is always error on at least one of the pendulums). Fig. 4b 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 these two figures, we see that:

1. the dynamic policies—discrete, optimal, and proportional—achieve better performance than the traditional static control policy,
2. optimal outperforms all other policies and reduces accumulated error by 20 percent-25 percent compared to traditional static control,
3. discrete reduces error by only about 3 percent, due to the limitations imposed by the available number and predefined values of the discrete periods, and
4. as the perturbation interval increases (Fig. 4b), the difference between optimal and proportional decreases.

This is because, with long-enough perturbation intervals (e.g., 20 seconds), most or all of the perturbations are nonoverlapped, and the proportional policy makes essentially the same slack allocations as the optimal policy, i.e., the available slack is allocated to the only task with error. It can be also seen that as the perturbation interval increases, the performance increases. This is due to the fact that the perturbations are less overlapped and the slack redistribution policies can perform their job more effectively. In addition, the absolute benefit of the baseline static policy decreases as the perturbation interval increases.

As shown in Fig. 5, the optimal and proportional policies also improve the control performance with the second workload (ball and beams), but by lower percentages (2 percent-12 percent) with perturbation intervals ranging from 5 to 200 seconds. The longer perturbation intervals correspond to the relatively longer computation times of the ball and beam controller. The somewhat lower performance improvement may reflect the differing dynamics of the underlying (simulated) system. Nevertheless, the performance improvements at all perturbation intervals are significant.

In order to demonstrate that the benefits associated with our dynamic feedback-based slack redistribution policies are due to the resource allocation decisions they make and not simply from some effect inherent in varying resource

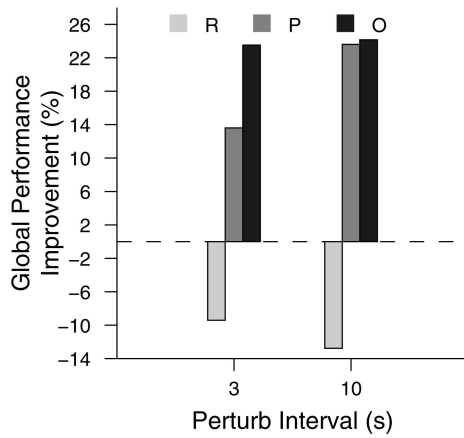


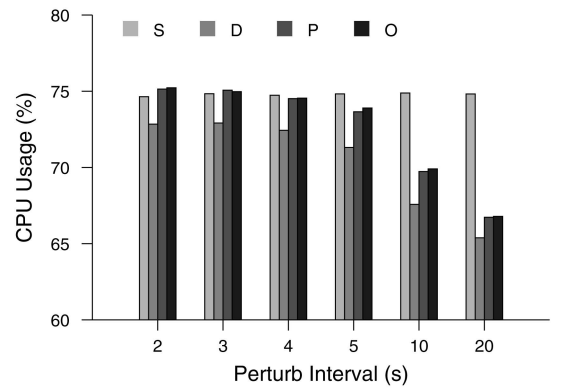
Fig. 6. Performance improvement compared to random case (relative to static).

allocations, we also compared them to a random dynamic slack allocation policy. The random policy allocates the slack randomly among the processes as often as the frequency of the perturbations. We performed a set of experiments that simulated the random policy with perturbation intervals equivalent to those used in the feedback slack redistribution policies. Fig. 6 shows the performance of random, proportional, and optimal policies relative to the static policy on the first set of workloads. The figure shows that the performance of the random policy is even worse (as indicated by the negative values) than that of the static policy. Furthermore, with larger perturbation intervals (e.g., 10 seconds), the relative performance of the random case gets worse while those of proportional and optimal improve. This is due to the fact that with short perturbation intervals, controllers frequently experience concurrent perturbations and so the random policy is relatively more likely to allocate the slack to a controller experiencing a perturbation; while with longer perturbation intervals, only one controller is likely to be experiencing a perturbation at any one time, and so the random policy is statistically less likely to give the resources to the (single) controller experiencing a perturbation at any given time.

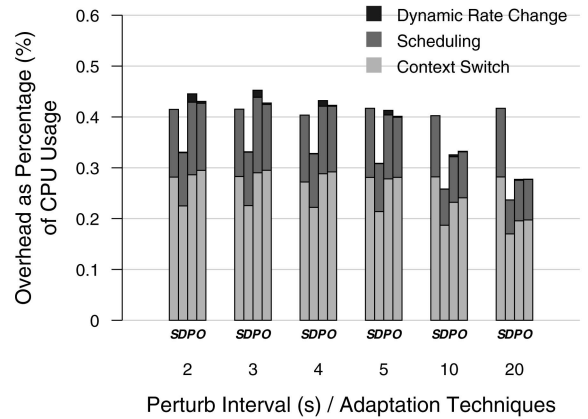
#### 6.4 CPU Usage and Overhead Analysis

In order to further demonstrate the feasibility of this approach, we examined its resource utilization and analyzed the overhead incurred by the four different policies. 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, as explained in Section 5.3. 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 as long as the other two can run at or above their lowest rate. With proportional, any variation in the errors causes a proportional rate adjustment among the three controllers, resulting in a larger number of adjustments.

Fig. 7a shows the measured total CPU usage of all control tasks with the four policies using the first workload. With a 4-second perturbation interval, the three adaptive feedback-based slack redistribution policies use almost



(a)



(b)

Fig. 7. CPU usage versus overhead. (a) CPU usage in 1 hour. (b) Overhead in 1 hour.

exactly the same amount of CPU as the static policy. As the perturbation intervals increase, beginning at about 5 seconds, the dynamic policies begin to consume less CPU due to the fact that when all the controlled systems are in equilibrium, their sampling interval is set to the minimum. This unused CPU can be allocated to other less time-critical tasks in the system or, in systems with dynamic voltage scaling, the system clock rate can be lowered, saving power.

Fig. 7b shows the overhead introduced by the four different policies (measured from our implementation) on the first workload. 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 (scheduling and context switches). 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 those of the static policy because they incur fewer context switches. This is due to the fact that they execute less frequently with longer periods. As expected, and although still negligible, proportional exhibits the largest dynamic rate change overhead due to its more frequent changing of rates, as explained in Section 5.

TABLE 1  
Key Features and Performance of Evaluated Methods

Approach	When	Dynamics	PI	solution	Performance
Seto et al. [2]	Off			periods	2.13
Eker et al. [16]	On	kernel		periods	2.13
Ben Gaid et al. [25]	On	kernel/plant	FH	job sequences	1.40
Castañé et al. [24]	On	kernel/plant	FH	periods	0.86
Our optimal policy	On	kernel/plant	Inst.	periods	0.81

## 7 COMPARATIVE ANALYSIS

In this section, we present a performance analysis of our approach compared to representative results in the area: those of Seto et al. [2], Eker et al. [16], Castañé et al. [24], and Ben Gaid et al. [25].

Table 1 presents a summary of the key features as well as the achieved performance for a given scenario. For each approach, the four first columns refer to

1. when the optimization is carried out (offline or online),
2. what dynamics (kernel workload and/or plant dynamics) are included in the optimization when it is performed online,
3. which type of performance index (instantaneous, which refers to our approach (19) or finite horizon, which is a prediction based on costs such as (10)) is included in the objective function of the optimization procedure when plant dynamics are accounted for, and
4. what type of solution is given after solving the optimization approach (sampling periods—task rates—or sequences of jobs).

The last column gives the performance results, as discussed below.

For this experiment, three identical ball and beams (as described earlier) were used as controlled plants. Each plant is controlled by a single control task. For the simulation purposes, it is assumed that the three plants start in equilibrium. Each plant is perturbed up to 30 times by randomly generated pulses. In particular, each plant is affected by a single perturbation occurring at a random time during each simulation period of 12 seconds. This is repeated 30 times, and the average value is reported. Each controller is designed to place the continuous closed-loop poles at  $s_{1,2} = -2 \pm 5i$ . Therefore, depending on the sampling period that applies the continuous closed-loop poles are mapped into the discrete-time domain,  $z_{1,2} = e^{s_{1,2}h}$ , and then the discrete-time controller is obtained by standard pole placement. This is either performed offline or online, depending on the evaluated method.

Simulations have been carried out with the *Truetime* simulator [51] to implement the multitasking processor together with each strategy. The performance analysis measures the impact of each strategy on the controlled plant dynamics. The evaluation during each simulation period ( $t_{sim}$ ) is performed using a continuous standard quadratic cost function:

$$J = \int_0^{t_{sim}} [x^T(t)Qx(t) + u^T(t)Ru(t)] dt, \quad (31)$$

where the weighting matrices are specified as

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad R = 1.$$

For the simulation purposes, it is assumed that plants' states are available, and therefore, there is no need for observers.

The periods of the three control tasks are allowed to take values between 0.5 and 0.3 second. Since the focus is on evaluation of control performance, no changes in the kernel workload and utilization set point have been injected. Therefore, for the online methods, changes in tasks' periods are based only on the dynamics of the controlled plants, not on the kernel workload. The specific task periods for each method are obtained as indicated by each approach. In particular, in the optimal policy, each task switches between 0.5 and 0.3 second. The stability analysis for this case is reported in Appendix B.

The last column of Table 1 summarizes the performance of the evaluated methods. The lower the numbers, which were obtained by the cost function given in (31), the better the approach. As can be seen, the first two methods provide the same performance because no changes in the workload have been introduced, which is the advantage of the method of Eker et al. [16] over that of Seto et al. [2]. The last three methods clearly improve control performance corroborating the assumption that reassigning resources according to the controlled plant dynamics is the key to improving control performance. The last two methods are much better than the method of Ben Gaid et al. [25] because in the case where perturbations overlap, the method of Ben Gaid et al. is not reactive enough. Recall that Ben Gaid mandates finishing the execution of the job sequence before applying a new one. Finally, although the difference in performance between Castañé's method and our policy looks subtle in this experiment, the computational overhead of our method is negligible compared to the computational overhead of Castañé's method. In fact, Castañé's method proposes a lookup table strategy to overcome the overhead problem. Overall, our approach is the best in this simulated experiment, providing better control performance.

## 8 CONCLUSIONS

Careful resource management is the key to providing the best possible performance in resource-constrained computing systems. We presented Draco, an efficient and effective system for managing resources allocated to concurrently executing controllers in such systems. A feedback-based slack management model for concurrently executing control tasks in resource-constrained environments allows Draco to allocate available slack as a function of the state of the controlled systems. Draco relies on adaptive control systems capable of operating at different (and varying)

sampling intervals. Based on the error reported for the plants controlled by these controllers, Draco dynamically adjusts the resources provided to the controllers, and thus the rate at which they operate. This approach has proven to be highly effective, outperforming both traditional static controllers and other dynamic controllers.

Draco's slack allocation model relies on complete information about the state of the controlled systems. We have shown that given this knowledge, the optimal solution is always to run the controller experiencing the greatest error at its maximum possible rate while running the rest at the minimum acceptable rate. Our experimental results from a real implementation of Draco (controlling simulated plants) demonstrate that, in practice, this approach outperforms the other algorithms we examined. In our experiments, the optimal solution outperforms the traditional static control solution by as much as 25 percent while consuming up to 30 percent less resources overall.

The feasibility of approaches aimed at improving control performance by adapting resource allocations at runtime require solutions with low computational overhead. Our experimental results have corroborated that our policies introduce negligible overhead relative to the context switch overhead, thus providing feasible solutions for runtime resource adaptation.

We anticipate that our adaptive control solutions will be useful in any resource-constrained environment in which multiple controllers are competing for a shared resource such as CPU cycles, network bandwidth, or battery power.

## APPENDIX A

### MAPPING FUNCTIONS

In this section, we show the benefit functions given by (15) and here reproduced omitting the  $i$ -subscript

$$g(r, x(t)) = ep(r)$$

can be mapped to standard cost functions used to design and evaluate optimal control systems, such as the cost (10) and here reproduced:

$$J(h) = \mathbb{E} \left( \sum_{k=0}^{N-1} \begin{bmatrix} x^T(kh) & u^T(kh) \end{bmatrix} Q \begin{bmatrix} x(kh) \\ u(kh) \end{bmatrix} + x^T(Nh)Q_{0c}x(Nh) \right).$$

The minimal value of (10) for a given optimal linear quadratic controller is [30]

$$J = x_0^T S x_0, \quad (32)$$

where  $S$  is the solution to the algebraic Riccati equation

$$S = \Phi^T S \Phi + Q_1 - (\Phi^T S \Gamma + Q_{12})(\Gamma^T S \Gamma + Q_2)^{-1}(\Gamma^T S \Phi + Q_{12}^T), \quad (33)$$

where the weighting matrices  $Q_1$ ,  $Q_{12}$ , and  $Q_2$  can be found in [30]. These matrices, as well as  $\Phi$ ,  $\Gamma$ , and  $S$ , all depend on the sampling interval  $h$ .

It is, however, also possible to evaluate the cost function (10) for an arbitrary (i.e., nonoptimal) state-feedback control law, as long as the closed-loop system is stable. In this case, the expression for  $S$  in (32) is replaced by the solution  $S$  to the Lyapunov equation

$$S = (\Phi - \Gamma L)^T S (\Phi - \Gamma L) + Q_1 - Q_{12} L - L^T Q_{12}^T + L^T Q_2 L. \quad (34)$$

In any case, cost (32) can be used to measure the performance when the controller (or a family of controllers) executes at different rates.

If we consider a first-order system, (32) reduces to a scalar equation

$$J(h) = x_0^T S(h) x_0 = x_0^2 S(h), \quad (35)$$

which has the structure given by (15). For second or higher order system, a similar result can be obtained. Noting that  $J(h) \in \mathbb{R}$ , then

$$J(h) = \text{vec}(J(h)), \quad (36)$$

where  $\text{vec}(\cdot)$  is the vector value function [52]. Therefore,

$$\begin{aligned} J(h) &= \text{vec}(J(h)) \\ &= \text{vec}(x_0^T S(h) x_0) \\ &= (x_0^T \otimes x_0^T) \text{vec}(S(h)), \end{aligned} \quad (37)$$

which has, again, the same structure as (15), in vector form.

Equations (35) and (37) are cost functions that depend on  $h$ . Therefore, they can exactly map benefit functions on  $r$  like (15) if  $J$  is expressed in terms of  $r$  (considering (3)) and we take  $g = -J$ .

## APPENDIX B

### STABILITY ANALYSIS: PARTICULAR CASE

The system described in Section 7 executes with two periods,  $h_1 = 0.5$  second and  $h_2 = 0.3$  second. To place the continuous closed-loop poles as specified, the discrete controller gains are

$$K_{0.3} = [13.595 \quad 4.3686], \quad K_{0.5} = [6.8991 \quad 3.4541].$$

The closed-loop matrices are then

$$\begin{aligned} \Phi_{cl}(0.3) &= \begin{bmatrix} 0.38822 & 0.10341 \\ -4.0785 & -0.31058 \end{bmatrix}, \\ \Phi_{cl}(0.5) &= \begin{bmatrix} 0.13761 & 0.068236 \\ -3.4496 & -0.72706 \end{bmatrix}. \end{aligned}$$

To check the stability of the system, we look for a Lyapunov function,  $V(x) > 0$ . Let  $V(x_k) = x_k^T P x_k$  be the quadratic Lyapunov function. If we show that  $V(x_{k+1}) - V(x_k) < 0$  for each subsystem characterized by each closed-loop matrix, the combined system will be stable [44]. This is the same as solving  $\Phi_{cl}(0.3)^T P \Phi_{cl}(0.3) - P \leq 0$  and  $\Phi_{cl}(0.5)^T P \Phi_{cl}(0.5) - P \leq 0$ .

Let

$$P = \begin{bmatrix} 1957.6881 & 152.6225 \\ 152.6225 & 122.9964 \end{bmatrix}.$$

It is easy to observe that  $V(x_k) > 0$  for any  $x_k$ . Therefore,  $V(x)$  is a Lyapunov function. Solving the previous inequalities, we check that

$$\Phi_{cl}(0.3)^T P \Phi_{cl}(0.3) - P = \begin{bmatrix} -100 & -1 \\ -1 & -100 \end{bmatrix} \leq 0,$$

$$\Phi_{cl}(0.5)^T P \Phi_{cl}(0.5) - P = \begin{bmatrix} -601.9163 & 123.0439 \\ 123.0439 & -64.0074 \end{bmatrix} \leq 0.$$

Therefore, the system is stable for any switching of periods, provided that the correct controller is applied, as we do.

## ACKNOWLEDGMENTS

This work has been partially supported by NoE ARTIST-DESIGN IST-FP7-2008-214373 and by Spanish Ministerio de Educación y Ciencia Project Ref. CICYT DPI2007-61527.

## REFERENCES

- [1] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multi-programming in a Hard-Real-Time Environment," *J. ACM*, vol. 20, no. 1, pp. 46-61, Jan. 1973.
- [2] D. Seto, J. Lehoczky, L. Sha, and K. Shin, "On Task Schedulability in Real-Time Control Systems," *Proc. 17th IEEE Real-Time Systems Symp. (RTSS '96)*, Dec. 1996.
- [3] H.S. Lee and B.K. Kim, "Dynamic Voltage Scaling for Digital Control System Implementation," *Real-Time Systems*, vol. 29, no. 2, pp. 263-280, Mar. 2005.
- [4] G. Buttazzo, M. Velasco, and P. Martí, "Quality-of-Control Management in Overloaded Real-Time Systems," *IEEE Trans. Computers*, vol. 56, no. 2, pp. 253-266, Feb. 2007.
- [5] P. Martí, G. Fohler, K. Ramamritham, and J.M. Fuertes, "Improving Quality-of-Control Using Flexible Time Constraints: Metric and Scheduling Issues," *Proc. 23rd IEEE Real-Time Systems Symp. (RTSS '02)*, Dec. 2002.
- [6] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *Proc. ACM SIGCOMM Symp.*, pp. 1-12, Sept. 1989.
- [7] S. Keshav, *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*. Addison Wesley, 1997.
- [8] 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," *Proc. 24th IEEE Real-Time Systems Symp. (RTSS '03)*, pp. 396-407, Dec. 2003.
- [9] S.H. Hong, "Scheduling Algorithm of Data Sampling Times in the Integrated Communication and Control Systems," *IEEE Trans. Control Systems Technology*, vol. 3, no. 2, pp. 225-230, 1995.
- [10] H. Rehbinder and M. Sanfridson, "Integration of Off-Line Scheduling and Optimal Control," *Proc. 12th Euromicro Conf. Real-Time Systems (ECRTS '00)*, pp. 137-143, June 2000.
- [11] M.S. Branicky, S.M. Phillips, and W. Zhang, "Scheduling and Feedback Co-Design for Networked Control Systems," *Proc. 41st IEEE Conf. Decision and Control (CDC '02)*, Dec. 2002.
- [12] Q. Ling and M. Lemmon, "Soft Real-Time Scheduling of Networked Control Systems with Dropouts Governed by a Markov Chain," *Proc. Am. Control Conf. (ACC '03)*, June 2003.
- [13] G. Beccari, S. Caselli, M. Reggiani, and F. Zanichelli, "Rate Modulation of Soft Real-Time Tasks in Autonomous Robot Control Systems," *Proc. 11th Euromicro Conf. Real-Time Systems (ECRTS '99)*, June 1999.
- [14] P. Ramanathan, "Overload Management in Real-Time Control Applications Using (m, k)-Firm Guarantee," *IEEE Trans. Parallel and Distributed Systems*, special issue on dependable real-time systems, vol. 10, no. 6, pp. 549-559, June 1999.
- [15] M. Caccamo, G. Buttazzo, and L. Sha, "Elastic Feedback Control," *Proc. 12th Euromicro Conf. Real-Time Systems (ECRTS '00)*, pp. 121-128, June 2000.
- [16] J. Eker, P. Hagander, and K.-E. Årzén, "A Feedback Scheduler for Real-Time Control Tasks," *Control Eng. Practice*, vol. 8, no. 12, pp. 1369-1378, Jan. 2000.
- [17] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén, "Feedback-Feedforward Scheduling of Control Tasks," *Real-Time Systems*, vol. 23, pp. 25-53, 2002.
- [18] H. Park, Y. Kim, D. Kim, and W. Kwon, "A Scheduling Method for Network-Based Control Systems," *IEEE Trans. Control Systems Technology*, vol. 10, no. 3, pp. 318-330, 2002.
- [19] D. Liu, X.S. Hu, M.D. Lemmon, and Q. Ling, "Firm Real-Time System Scheduling Based on a Novel QoS Constraint," *IEEE Trans. Computers*, vol. 55, no. 3, Mar. 2006.
- [20] Q. Zhao and D.-Z. Zheng, "Stable and Real-Time Scheduling of a Class of Perturbed Hybrid Dynamic Systems," *Proc. 14th IFAC World Congress (IFAC '99)*, vol. J, pp. 91-96, 1999.
- [21] G. Walsh and H. Ye, "Scheduling of Networked Control Systems," *IEEE Control Systems Magazine*, vol. 21, no. 1, pp. 57-65, 2001.
- [22] D. Henriksson, A. Cervin, J. Åkesson, and K.-E. Årzén, "Feedback Scheduling of Model Predictive Controllers," *Proc. Eighth IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS '02)*, Sept. 2002.
- [23] D. Henriksson and A. Cervin, "Optimal On-Line Sampling Period Assignment for Real-Time Control Tasks Based on Plant State Information," *Proc. 44th IEEE Conf. Decision and Control and European Control Conf. (CDC-ECC '05)*, Dec. 2005.
- [24] R. Castañé, P. Martí, M. Velasco, A. Cervin, and D. Henriksson, "Resource Management for Control Tasks Based on the Transient Dynamics of Closed-Loop Systems," *Proc. 18th Euromicro Conf. Real-Time Systems (ECRTS '06)*, July 2006.
- [25] M.-M. Ben Gaid, A. Çela, Y. Hamam, and C. Ionete, "Optimal Scheduling of Control Tasks with State Feedback Resource Allocation," *Proc. Am. Control Conf. (ACC '06)*, June 2006.
- [26] P. Martí, C. Lin, S.A. Brandt, M. Velasco, and J.M. Fuertes, "Optimal State Feedback Based Resource Allocation for Resource-Constrained Control Tasks," *Proc. 25th IEEE Real-Time Systems Symp. (RTSS '04)*, pp. 161-172, Dec. 2004.
- [27] C. Lin and S.A. Brandt, "Improving Soft Real-Time Performance through Better Slack Reclaiming," *Proc. 26th IEEE Real-Time Systems Symp. (RTSS '05)*, pp. 3-14, Dec. 2005.
- [28] G.C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.
- [29] H. Kopetz, "Sparse Time versus Dense Time in Distributed Real-Time Control Systems," *Proc. 12th Int'l Conf. Distributed Computing Systems (ICDCS '92)*, pp. 460-467, 1992.
- [30] K.J. Åström and B. Wittenmark, *Computer-Controlled Systems*, third ed. Prentice-Hall, 1997.
- [31] P. Martí, J.M. Fuertes, G. Fohler, and K. Ramamritham, "Jitter Compensation for Real-Time Control Systems," *Proc. 23rd IEEE Real-Time Systems Symp. (RTSS '01)*, Dec. 2001.
- [32] C. Lee, J. Lehoczky, R. Rajkumar, and D. Siewiorek, "On Quality of Service Optimization with Discrete QoS Options," *Proc. IEEE Real-Time Technology and Applications Symp. (RTAS '99)*, June 1999.
- [33] A. Burns, D. Prasad, A. Bondavalli, F.D. Giandomenico, K. Ramamritham, J. Stankovic, and L. Strigini, "The Meaning and Role of Value in Scheduling Flexible Real-Time Systems," *J. Systems Architecture*, vol. 46, no. 4, pp. 305-325, Jan. 2000.
- [34] H. Aydin, R. Melhem, D. Mosseé, and P. Mejia-Alvarez, "Optimal Reward-Based Scheduling for Periodic Real-Time Tasks," *IEEE Trans. Computers*, vol. 50, no. 2, pp. 111-130, Feb. 2001.
- [35] S. Brandt and G. Nutt, "Flexible Soft Real-Time Processing in Middleware," *Real-Time Systems*, vol. 22, pp. 77-118, 2002.
- [36] T. Hegazy and B. Ravindran, "Using Application Benefit for Proactive Resource Allocation in Asynchronous Real-Time Distributed Systems," *IEEE Trans. Computers*, vol. 51, no. 8, pp. 945-962, Aug. 2002.
- [37] Y. Chen, C. Lu, and X. Koutsoukos, "Optimal Discrete Rate Adaptation for Distributed Real-Time Systems," *Proc. 28th IEEE Real-Time Systems Symp. (RTSS '07)*, Dec. 2007.
- [38] E.K. Chong and S.H. Zak, *An Introduction to Optimization*. John Wiley & Sons, 1996.
- [39] W. Gellert, S. Gottwald, and M. Hellwich, *The VNR Concise Encyclopedia of Mathematics*. Van Nostrand Reinhold, 1988.
- [40] B. Wittenmark and K.J. Åström, "Simple Self-Tuning Controllers," *Methods and Applications in Adaptive Control*, vol. 24, H. Unbehauen, ed., pp. 21-29, 1980.

- [41] P. Albertos and J. Salt, "Digital Regulator Redesign with Irregular Sampling," *Proc. 11th IFAC World Congress (IFAC '90)*, preprints, vol. 8, pp. 157-161, 1990.
- [42] M. Dogrueel and U. Özgüner, "Stability of a Set of Matrices: A Control Theoretic Approach," *Proc. 34th IEEE Conf. Decision and Control (CDC '95)*, Sept. 1995.
- [43] M. Schinkel, W.-H. Chen, and A. Rantzer, "Optimal Control for Systems with Varying Sampling Rate," *Proc. Am. Control Conf. (ACC '02)*, May 2002.
- [44] D. Liberzon, *Switching in Systems and Control*. Birkhauser, 2003.
- [45] P. Naghshabrizi and J.P. Hespanha, "Stability of Network Control Systems with Variable Sampling and Delays," *Proc. 44th Ann. Allerton Conf. Comm., Control, and Computing*, Sept. 2006.
- [46] G.C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic Scheduling for Flexible Workload Management," *IEEE Trans. Computers*, vol. 51, no. 3, pp. 289-302, Mar. 2002.
- [47] S. Goddard and L. Xu, "A Variable Rate Execution Model," *Proc. 16th Euromicro Conf. Real-Time Systems (ECRTS '04)*, pp. 135-143, July 2004.
- [48] V. Yodaiken, "The RTLinux Manifesto," *Proc. Fifth Linux Expo*, Mar. 1999.
- [49] C. Lin, P. Martí, S.A. Brandt, S. Banachowski, M. Velasco, and J.M. Fuertes, "Improving Control Performance Using Adaptive Quality of Service in a Real-Time System," *Work in Progress Session of the IEEE Real-Time and Embedded Technology and Applications Symp.*, Technical Report UCSC-CRL-04-04, Univ. of California, Santa Cruz, May 2004.
- [50] M.R. Garey and D.S. Johnson, *Computers and Intractability*. W.H. Freeman, 1979.
- [51] D. Henriksson, A. Cervin, and K.-E. Årzén, "TrueTime: Simulation of Control Loops under Shared Computer Resources," *Proc. 15th IFAC World Congress on Automatic Control*, July 2002.
- [52] J.W. Brewer, "Kronecker Products and Matrix Calculus in System Theory," *IEEE Trans. Circuits and Systems*, vol. 25, no. 9, pp. 772-781, Sept. 1978.



**Pau Martí** received the degree in computer science and the PhD degree in automatic control from the Technical University of Catalonia, Barcelona, in 1996 and 2002, respectively. Since 1996, he has been an assistant professor in the Department of Automatic Control, Technical University of Catalonia. During his PhD, he was a visiting student at Malardalen University, Sweden. From 2003 to 2004, he held a research fellow appointment in the Department of Computer Science, University of California, Santa Cruz. His research interests include embedded systems, control systems, real-time systems, and communication systems. He is a member of the IEEE.



**Caiuxue Lin** received the BS and MS degrees in computer science and engineering from Zhejiang University in 1998 and 2001, respectively, the MS degree in computer science from the University of California, Santa Cruz (UCSC) in 2003, and the PhD degree in computer science from UCSC in 2006 for his work in the area of adaptive soft real-time systems. He is currently a member of the technical staff at VMware.



**Scott A. Brandt** received the BMath degree and the MS degree in computer science from the University of Minnesota in 1987 and 1993, respectively, and the PhD degree in computer science from the University of Colorado at Boulder in 1999. He is currently a professor of computer science in the Department of Computer Science, University of California, Santa Cruz (UCSC), the director of the UCSC/Los Alamos National Laboratory Institute for Scalable Scientific Data Management (ISSDM), and a founder of the UCSC Storage Systems Research Center (SSRC). His research interests are broadly in the area of computer systems, and his research ranges from scalable, high-performance distributed storage to real-time CPU scheduling. He is a senior member of the IEEE.



**Manel Velasco** received the degree in maritime engineering and the PhD degree in automatic control from the Technical University of Catalonia, Barcelona, in 1999 and 2006, respectively. Since 2002, he has been an assistant professor in the Department of Automatic Control, Technical University of Catalonia. He has been involved in research on artificial intelligence from 1999 to 2002 and, since 2000, on the impact of real-time systems on control systems. His research interests include artificial intelligence, real-time control systems, and collaborative control systems, especially on redundant controllers and multiple controllers with self-interacting systems.



**Josep M. Fuertes** received the degree in industrial engineering and the PhD degree from the Technical University of Catalonia (UPC), Barcelona, in 1976 and 1986, respectively. From 1975 to 1986, he was a researcher at the Institut de Cibernètica (Spanish Consejo Superior de Investigaciones Científicas). In 1987, he became a permanent professor in the Department of Automatic Control, UPC. In 1987, he got a position for a year at the Lawrence Berkeley Laboratory, Berkeley, California, where he was a visiting scientific fellow for the design of the Active Control System of the W.M. Keck 10-m segmented telescope (Hawaii). From 1992, he was responsible for the University Research Line in Advanced Control Systems and later of the Distributed Control Systems Group. From 1996 to 2001, he acted as a Spanish Representative at the Council of the European Union Control Association. His research interests include the areas of distributed, networked and real-time control systems and applications. Since 1989, he has been coordinating projects at the national and international levels related to the aforementioned areas of expertise. He is a member of the Administrative Committee of the IEEE Industrial Electronics Society, where he is chairing the Technical Committee of Networked Based Control Systems and Applications. He is also the president of the Catalan Society of Technology, at the Institut d'Estudis Catalans Academy. He has collaborated as the organizer (1997 IEEE International Workshop on Factory Communications Systems, 1999 IEEE International Conference on Emerging Technologies and Factory Automation), the chairman, a session organizer, and a member of program committees for several international conferences. He is a senior member of the IEEE.

► **For more information on this or any other computing topic, please visit our Digital Library at [www.computer.org/publications/dlib](http://www.computer.org/publications/dlib).**