State Feedback Based Resource Allocation for Resource-Constrained Control Tasks

Pau Martí, 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.velasco,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 resource-constrained computing systems with multiple control loops are based on *open-loop* strategies, statically allocating a fixed portion of the system resources to each controller independent of its current resource needs. Our research demonstrates that these open-loop strategies fail to provide the best possible control performance within the available resources. We present 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 dynamically allocating resources to controllers as a function of the current state of their controlled systems both improves control performance and reduces resource utilization. We discuss the application of our state-feedback resource management to both centralized and distributed control systems and present results demonstrating its effectiveness in both domains. In the best case, our system improved control performance by 25% with 30% less resources.

I. Introduction

Modern embedded systems are expected to provide both highly varied functionality and outstanding application performance within the available processor and network computing resources. Fully exploiting system resources is crucial to maximizing overall performance. However, most traditional systems for managing multiple concurrently executing control loops work *open-loop*, using *a priori* characterizations of the expected workload to determine appropriate (fixed) resource allocations, regardless of the dynamics of the control applications. This is done, for example, in the early canonical work on uniprocessor real-time scheduling [1], in more recent approaches to control and uniprocessor real-time systems [2], and in co-design approaches to control and scheduling in networked control systems [3].

Open-loop resource allocation policies work well, guaranteeing that they can meet given control performance specifications by reserving a constant portion of the system resources for each controller. However, this is done statically and so cannot account for the potentially varying resource needs of the controllers. A closer look at the behavior of control loops and at the relation between control performance and controller execution rate suggests that such open-loop resource

allocation may be suboptimal. A controller whose system is in equilibrium may not require the assigned execution rate while one experiencing a perturbation may perform better with a higher rate [4]. In the former case, the contribution of each job is essentially nil and the resources used for it were wasted, while in the latter an increase in the rate of the controller—impossible with traditional control techniques—could decrease system deviation and hasten system recovery, thereby improving control performance.

Motivated by this observation, we have developed a dynamic resource management system that allocate resources to control tasks at run-time based on *feedback* information from the controlled systems. We show that dynamically allocating resources to controllers as a function of the state of their controlled systems improves control performance while reducing resource utilization.

In successfully implementing the state feedback resource allocation strategies two key technical challenges were overcome. First, we developed and implemented controllers capable of running with different sampling frequencies given different resource allocations, thus providing better control performance when given more resources [5], [4]. Second, we developed and implemented a flexible real-time platform capable of dynamically changing the allocated resources (via e.g., the controllers' periods) according to the application feedback while guaranteeing tasks' timing constraints [6]. Together, these two solutions provide the infrastructure needed to develop dynamic state-feedback based control systems.

After implementing and integrating these components, we performed extensive experiments including a comparative analysis with the static approach to resource management traditionally used in real-time control systems. The experiments we present 1) corroborate that the dynamics of the controlled systems are the key to better exploiting system resources and improving control systems performance, 2) confirm our theoretical results, and 3) show that our state feedback policies improve control performance and/or save system resources. In the best case, which occurs in a centralized system with relatively infrequent (i.e. generally non-overlapping) perturbations, our system was able to reduce controller error by more than 25% using 30% less resources.

The rest of this paper is organized as follows. In Sections II- IV we discuss the application of these techniques to the allocation of CPU resources to concurrently executing control tasks in a uniprocessor system. We formulate the problem and give an optimal solution, discuss our implementation, and present results showing the benefits of this approach over traditional static control. In Sections V- VII we discuss the application of these techniques to a distributed system

lacking centralized knowledge about the state of all controlled plants, making our centralized solution impossible to determine. We present a heuristic solution, discuss its implementation, and present results demonstrating the benefits of this approach over traditional static control and comparing its performance to the optimal solution. Section VIII discusses related research, placing our research into context and differentiating it from previous research in the area. Section IX concludes the paper.

II. A Centralized Solution

A. System architecture

Consider a real-time system with n controllers (see Figure 1), each one controlling a plant (or controlled system), and competing for the computing resources. Each plant i can be described by the ordinary vector differential equations given in (1) and (2) (called *state* and *output* equations, respectively) that describe their linear dynamics[1], where functions f_i and d_i are linear, $u_i(t)$ is the *control input* to the dynamical system, and the vector $x_i(t) = [x_i^1(t), \ldots, x_i^n(t)]$ is the state of the system at time t whose elements are called *state-variables*.

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

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

If, without loss of generality, we consider the equilibrium point of all controlled systems to be zero, the norm of the state vector (3), also called *error*, is the distance of each controlled system from its equilibrium point at time $t > 0$[2]. This measure, $e_i(t)$, indicates how critical the situation is for each controlled system—the greater the error, the worse the situation. This is used by the system to re-allocate resources r_i to each controller in such a way that more resources are assigned exactly when they are needed the most, i.e. when their controlled systems are experiencing the greatest error.

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

In a uniprocessor architecture where processor time is the critical constraint, each control task with its controlled plant constitutes a control loop. The control task implements the three main

[1]Henceforth, the subscript i will specify a control loop, identifying either the controller or the controlled plant.

[2]If, for a given control loop, all states cannot be measured, they can be determined from the available measurements and a model [7].

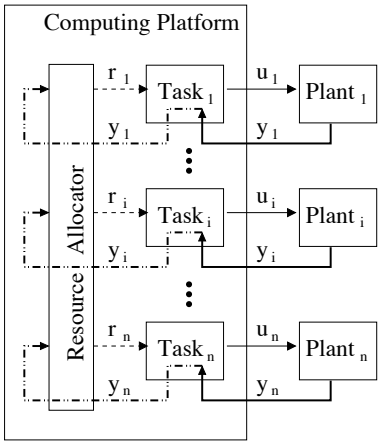


Fig. 1. Feedback architecture

actions to be performed sequentially in each closed-loop operation: sampling, control algorithm computation, and actuation. Each control task is characterized by its period h_i (corresponding to the sampling period) and its worst-case execution time c_i. The partial utilization factor of each task, also called *rate*, r_i (4), 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).

$$r_i = \frac{c_i}{h_i} \tag{4}$$

In section V, we extend the uniprocessor-based architecture to a network-based architecture where network bandwidth instead of CPU time is the constrained resource to be allocated.

B. Problem to be solved

The basic operation of the uniprocessor architecture, schematically illustrated in Figure 1, 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 equilibrium point). Each sample will show whether the controlled system is affected by a perturbation or not. The samples provide the magnitude of the controlled system deviations from the equilibrium points. This information is fed back to the operating system in order to allow the system to re-allocate resources accordingly.

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 in isolation. Given that the controllers will provide better control performance when given more resources, the problem to be solved is how to assign the scarce CPU resources to the set of controllers, according to the dynamics of the controlled plants, so that overall control performance is maximized (or resource usage minimized).

C. Problem formulation

Controller design attempts to minimize the system error produced for 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 Modern Control Systems, by Dorf and Bishop [8] for a review of classic control criteria or Shin *et. al* [9] for a review of performance measures to evaluate real-time computer control systems).

In addition to the characterization of each control loop given by equations (1) and (2), for each control task we specify a *performance criterion* $p_i(r_i)$ that specifies the relationship between these indices and the tasks' periods, and which will be used by the resource manager to allocate resources. The performance criterion represents the control performance under different task rates, r_i, specified for each control loop given the controlled system, the controller and a range of periods. As pointed out by Cervin *et al.* [10], a controller can normally give satisfactory performance within a range of sampling periods. Each performance criterion can be computed *a priori* and condensed in a function that maps controller period to control performance.

In fact, the relation between control performance (measured using a standard quadratic or linear performance index) and a range of allowed periods can be approximated by a linear relationship [10]. Therefore, for a given control task i, we approximate its performance criterion $p_i(r_i)$ by a linear increasing function (5), 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*. Parameters α_i and β_i in (5) are specific for each control loop and can be obtained prior to system run-time.

$$p_i(r_i) = \alpha_i r_i + \beta_i \tag{5}$$

Although this linear approximation is not an oversimplification and, in fact, covers a wide class of control systems (as we will discuss in Section III-A), the optimal resource allocation policy also admits performance criteria in the form of polynomials of degree less than five, as

we further discuss in Section III-A.

At the system level, the resource management could use only this static information (all performance criteria of the set of control loops) to determine the allocation of resources so that all tasks are schedulable and the performance criteria are optimized for a certain objective function (as is done by Cervin *et al*. [10]). However, by doing so, the maximum apparent 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 also taking into account the current dynamics of each controlled system, we can determine the optimal allocation for a more complete objective function.

To illustrate the problem, and the solution, 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 than the other. Suppose also that the amount of computing resources is limited enough so that the system cannot simultaneously run both controllers at their highest rates. 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. To make this possible, and to assign resources according to the dynamics of each controlled system, we re-scale each performance criterion of each control loop by the error (the feedback measure defined in (3)) of the controlled system. Finally, we also consider current system load and the number of processes executing.

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), as represented by (6)

$$\tau_i = \{r_i, p_i, e_i\} \tag{6}$$

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.

III. Centralized strategy to resource management

We show 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 [11]. The solution to this problem is a feasible algorithm (in terms of computational complexity) for a run-time resource manager, and provides the optimal resource allocation policy for control tasks [12].

A. Optimal state feedback based resource allocation policy

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

$$\text{maximize} \quad g(p_i(r_i), e_i) \tag{7}$$

$$\text{subject to} \quad \sum_{i=1}^{n} r_i \leq U_d \tag{8}$$

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 (7), restricted to the utilization feasibility constraint specified in (8), 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 (8). A generic algorithm to find the solution can be summarized in four steps (as detailed by Chong and Zak [11]):

Step 1: Search for local relative maxima in the interior of the feasibility set by solving the set of equations specified by (9), where $\frac{\partial g}{\partial r_i}$ are the partial derivatives of g with respect to each r_i.

$$\frac{\partial g}{\partial r_1} = 0, \; \frac{\partial g}{\partial r_2} = 0, \; \ldots, \; \frac{\partial g}{\partial r_n} = 0 \tag{9}$$

and keep those $\vec{r}$ that, being interior points of the feasibility set (conforming with the restriction (8)), maximize g.

Step 2: Search for local relative maxima in the boundary of the feasibility set by solving the

set of equations specified by (10),

$$\begin{aligned}
&\frac{\partial g([U_d - r_2 - r_3 - \ldots - r_n, r_2, r_3, \ldots, r_n])}{\partial r_i} = 0,\ i \leq n, i \neq 1 \\
&\frac{\partial g([r_1, U_d - r_1 - r_3 - \ldots - r_n, r_3, \ldots, r_n])}{\partial r_i} = 0,\ i \leq n, i \neq 2 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
&\frac{\partial g([r_1, r_2, \ldots, U_d - r_1 - r_2 - \ldots - r_{n-1}])}{\partial r_i} = 0,\ i \leq n, i \neq n
\end{aligned} \tag{10}$$

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

Step 3: Search for the g values of the feasibility set extremes as specified in (11),

$$g([U_d, 0, \ldots, 0]),\ g([0, U_d, \ldots, 0]),\ \ldots,\ g([0, 0, \ldots, U_d]) \tag{11}$$

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

Step 4: Choose among those obtained in *Step 1, 2* and *3* an $\vec{r}$ 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 II-A, the optimization problem can be simplified, the resulting problem is directly solvable, and the algorithm that obtains the solution can feasibly be executed at run-time, as we explain next.

If each controller is independent — controlling an independent plant (as we assumed in the system model in Section II-A) — the function $g(.)$ that links all of the control performance benefits (which are given by the performance criteria p_i defined in (5)) 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 by Seto *et al.* [2] or Cervin *et al.* [10]).

Each performance criterion can also have a weight, 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 re-scaling each performance criterion by e_i to account for the controlled system error, for the given set of n controllers we can rewrite the optimization

problem as follows:

$$\text{maximize} \quad \sum_{i=1}^{n} w_i e_i p_i(r_i) \tag{12}$$

$$\text{subject to} \quad \sum_{i=1}^{n} r_i \leq U_d \tag{13}$$

The complexity of the solution of the optimization problem stated in (12) and (13) depends on each function $p_i(r_i)$ due to the fact that equations (9) and (10) have been simplified to the equation set (14) and (15) (because g has turned into a sum), where $b_i = w_i e_i p_i(r_i)$:

$$\frac{\partial b_1}{\partial r_1} = 0, \; \frac{\partial b_2}{\partial r_2} = 0, \; \ldots, \; \frac{\partial b_n}{\partial r_n} = 0 \tag{14}$$

$$\begin{aligned}
\frac{\partial b_1(U_d - r_2 - r_3 - \ldots - r_n)}{\partial r_i} &= 0, \; i \leq n, i \neq 1 \\
\frac{\partial b_2(U_d - r_1 - r_3 - \ldots - r_n)}{\partial r_i} &= 0, \; i \leq n, i \neq 2 \\
&\vdots \\
\frac{\partial b_n(U_d - r_1 - r_2 - \ldots - r_{n-1})}{\partial r_i} &= 0, \; i \leq n, i \neq n
\end{aligned} \tag{15}$$

If the performance criteria p_i are linear (as we assumed in our system model in Section II-A), 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 (14) and (15) corresponding to *Step 1* and *2*, are not properly determined. That is, if p_i are linear ($p_i = \alpha_i r_i$) in (14), we end up with the following set of equations (16)

$$\begin{aligned}
\frac{\partial b_1}{\partial r_1} &= \frac{\partial w_1 e_1 \alpha_1 r_1}{\partial r_1} = w_1 e_1 \alpha_1 = 0 \\
\frac{\partial b_2}{\partial r_2} &= \frac{\partial w_2 e_2 \alpha_2 r_2}{\partial r_2} = w_2 e_2 \alpha_2 = 0 \\
&\vdots \\
\frac{\partial b_n}{\partial r_n} &= \frac{\partial w_n e_n \alpha_n r_n}{\partial r_n} = w_n e_n \alpha_n = 0
\end{aligned} \tag{16}$$

that are not determined. The same happens with the set of equations specified in (15). Therefore, by simply performing *Step 3* customized for the problem stated in (12) and (13), that is, by

evaluating

$$\begin{aligned} g([U_d, 0, \ldots, 0]) &= b_1(U_d) + b_2(0) + \ldots + b_n(0) = w_1 e_1 \alpha_1 U_d \\ g([0, U_d, \ldots, 0]) &= b_1(0) + b_2(U_d) + \ldots + b_n(0) = w_2 e_2 \alpha_2 U_d \\ &\vdots \\ g([0, 0, \ldots, U_d]) &= b_1(0) + b_2(0) + \ldots + b_n(U_d) = w_n e_n \alpha_n U_d \end{aligned} \tag{17}$$

we will find the optimal resource allocation. Finally we observe that (17) 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 (12) and (13) 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 of control tasks is described by (6).

Proof: Follows from the argument above. ■

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 degree less than five [13], an analytical solution can be found following *Steps 1, 2* and *3* by solving equations (14), (15) and (11), turning the solution into a feasible algorithm (in terms of computational complexity) 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 by (12) and (13) 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 (12) on the hyperplane defined by the objective function (13). 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 can be seen in the figure, the maximum benefit considering the schedulability constraints is

found at $\vec{r} = [0.8, 0.0]$ (extreme point).

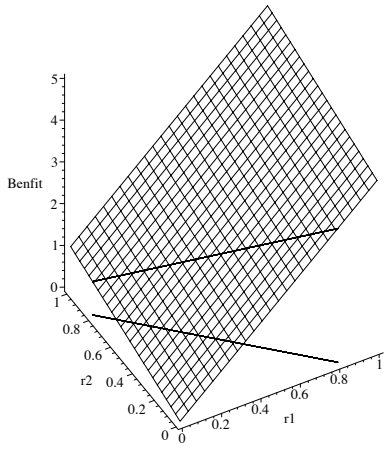


Fig. 2. Optimal Solution

B. Controller design

The application of the optimal policy requires the implementation of controllers capable of running with different sampling frequencies given different resource allocations, providing better control performance when given more resources (as explained by Marti *et al.* [5], [4]). Recall that for each controller, the execution frequency (i.e., the task period h) is obtained from the rate assigned by the optimal policy, as detailed in (4)).

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 [7]. In the end, each control law is an algorithm that depends on the sampling period, $L(h)$. For each controller, we specify a range of sampling periods $h = [h_{min}, \ldots, h_{max}]$ for which the closed loop requirements are met and allow the controller, implemented within a task, to execute with a run-time period that belongs to the specified range, adapting the gains accordingly (following the techniques presented in [14] and [15]). If equation (18) is the discretization of the system equation (1) we consider for a given control loop,

$$\vec{x}_{n+1} = \Phi(h)\vec{x}_n + \Gamma(h)u_n \tag{18}$$

and if the input u_n is given by a state feedback control law designed according to (18), the new system dynamics in closed-loop are given by the following state-space representation

$$\vec{x}_{n+1} = \Phi(h)\vec{x}_n + \Gamma(h)L(h)\vec{x}_n \tag{19}$$

where (20) is the closed loop matrix that depends on the period h.

$$\Phi(h)_{cl} = (\Phi(h) + \Gamma(h)L(h)) \tag{20}$$

If at run-time, at each control loop execution, the period is allowed to change within h_{min} and h_{max} and $L(h)$ is adapted accordingly, the closed loop matrix will take different values, from a finite set of values. For this type of system evolution, system stability can be analyzed using the approach described by Dogruel and Özgüner [16].

IV. Implementation and results for centralized resource allocation

Feedback-based resource management requires that all of the controllers be capable of running with different sampling frequencies given different resource allocations. Each controller is considered a flexible real-time process which has 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), $h_i \in [h_{min} \dots h_{max}]$ (which for now we consider continuous), and relative hard deadlines equal to their periods. Given a set of n controllers, their resource utilizations are defined by $r_i = \frac{c_i}{h_i}$. To guarantee a feasible schedule (*i.e.*, no deadline is missed) for a set of controllers with any scheduling algorithm q, the following condition must be satisfied $\sum_{i=1}^{n} r_i \leq U_d^q$, where U_d^q is the schedulable utilization of algorithm q. For example, $U_d^{EDF} = 1$. Recall that such a schedulability constraint was considered in the optimal policy (in section III) when constraining the optimization problem with equation (8).

For the uniprocessor architecture, dynamic resource allocation for controllers can be achieved by any existing scheduling framework or scheduling algorithm which supports dynamic task period adjustment at run time and guarantees no deadline miss during the adjustment. Examples include RBED [6], the Elastic Model [17], and VRE [18].

A. Centralized feedback resource allocation

As a prototype and performance demonstration, our centralized control feedback architecture is implemented in the RBED integrated real-time system [6], 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% 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 are subject to the constraints described in [6].

We implemented the optimal feedback resource allocation policy (*optimal*, described in Section III-A) 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 among all control loops such that all tasks get a CPU share proportional to $w_i e_i \alpha_i$. The discrete policy, which is a mixture of optimal and proportional with a set of discrete values for the task periods, allocates resources in discrete amounts, reducing computational overhead. 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. Note that in the optimal and proportional policies the resource utilizations $\frac{c_i}{r_i}$ can have continuous values in the interval $[h_{min} \ldots h_{max}]$, while in the discrete policy they only have discrete values in the set $\{h_{min}, \ldots, h_{max}\}$.

1) Optimal: It implements the optimal solution described in Section III. Like all of the adaptive feedback based resource allocation policies implemented in this paper, it allows each control task to specify a performance criterion p_i (see Section II-C), which is a continuous function that relates the period (and thus resource usage) of the controller to the performance it provides. As we discussed in Section III-A, the optimal solution is to assign all the available resources to the controller whose controlled system has the largest error (if all the control loops are the same, that is, with equal w_i and p_i), after guaranteeing a minimum rate for the remaining controllers. With different p_i and weights w_i (meaning different type of controllers, different controlled processes, or controlled processes of different importance), the resources would be assigned to the controller having the highest $w_i e_i p_i$. This is summarized in (21), where n is the number of controllers and $r_{j,min}$ (corresponding to h_{max}) is the guaranteed minimum utilization of the controller j.

$$r_i = \begin{cases} U_d - \sum_{j \leq n}^{j \neq i} r_{j,min}, & \textit{if } w_i e_i p_i(r_i) \text{ is maximum} \\ r_{i,min}, & \textit{otherwise} \end{cases} \tag{21}$$

The optimal policy keeps track of which task has maximum $w_i e_i p_i$. This can be determined in $O(n)$ time by a linear scan of the list of n tasks. Reallocation of resources occurs when the task with the maximum $w_i e_i p_i$ changes.

2) Proportional: It was our first guess at an optimal allocation strategy and represents a "fair" allocation based on measured error. It assigns resources to each controller as a proportion of $w_i e_i \alpha_i$. In our implementation, the resource allocation algorithm is given by

$$r_i = \frac{w_i e_i \alpha_i}{\sum_{j \leq n} w_j e_j \alpha_j} \cdot U_d \tag{22}$$

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 (22) 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 (22). However, any change to the error of a controlled system will cause a dynamic resource allocation 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.

3) Discrete: It is based on the assumption of a discrete set of periods (which may be easier to implement using traditional control technologies) instead of the continuous range defined for optimal and proportional. Each control task has a discrete function corresponding to p_i evaluated for 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

$$benefit_i = w_i e_i p_i(r_i) = w_i e_i \alpha_i r_i \tag{23}$$

The optimal discrete allocation is NP-hard in general, based upon a straightforward reduction to the knapsack problem [19]. A heuristic algorithm [20], [21] 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 [20]. In this way, rate readjustment takes place only when the error moves from one range to another. This significantly reduces the overhead (see overhead analysis in Section IV-C).

4) Static: It statically allocates available resources to the controllers only based on their static information, which include w_i and α_i. The resource allocation algorithm is as follows

$$r_i = \frac{w_i \alpha_i}{\sum_{j \leq n} w_j \alpha_j} \cdot U_d \tag{24}$$

Among the four policies that we implemented, the static policy is the only one that does not use dynamic error e_i to allocate resources.

B. System interface implementation details

Figure 3 gives the pseudo-code for a controller and the dynamic rate adjustment in the centralized architecture. A controller requests a dynamic rate adjustment through the system call $\mathbf{rate\_adjust}()$. The controller (Figure 3(a)), with execution period h_i, does two things each period.

1) It 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) 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, $\mathbf{rate\_adjust}()$, in order to obtain the (possibly new) period that will start being used at the following controller execution.

The dynamic rate adjustment (Figure 3(b)) uses the specified resource adaptation policy (static, discrete, proportional, or optimal) to re-allocate the resources based on $w_i e_i \alpha_i$.

C. Performance evaluation

In order to evaluate the performance results, we have implemented our feedback based resource allocation 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

Controller
{
$h_i := h_i^{next}$
$x_i := \mathbf{read\_input}()$
$u_i := \mathbf{calculate\_output}(h_i)$; $\mathbf{send\_output}(u_i)$
$x_i^{next} := \mathbf{update\_state}(x_i, h_i)$; $e_i := |x_i^{next}|$
$r_i := \mathbf{rate\_adjust}(i, e_i)$
$h_i^{next} := \frac{c_i}{r_i}$
}

(a) Controller (Application Level)

rate_adjust(i, e_i)
{
if $(e_i != e_i^{old})$
{
$b_i := w_i e_i \alpha_i$
$r_i^{next} := \mathbf{resource\_allocation}(policy, b_i)$
$e_i^{old} := e_i$
return r_i^{next}
}
}

(b) Dynamic Rate Adjustment (System Level)

Fig. 3. Pseudo-code for Controller and Dynamic Rate Adjustment in Centralized Architecture

over long periods of time and performed a large number of experiments with randomly generated workloads. 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, 512MB RAM, and a 40GB hard drive.

1) Controlled processes: We simulated two kinds of controlled plants: an inverted pendulum and a ball & beam. 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. 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^2.

The linear time-invariant state space model for each ball & 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 θ is the beam angle and x is the ball position on the beam.

2) Workload generation: In the following experiments, we used two sets of workloads: three controllers, each controlling a simulated inverted pendulum, and three controllers, each controlling a simulated ball & 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. However, using different

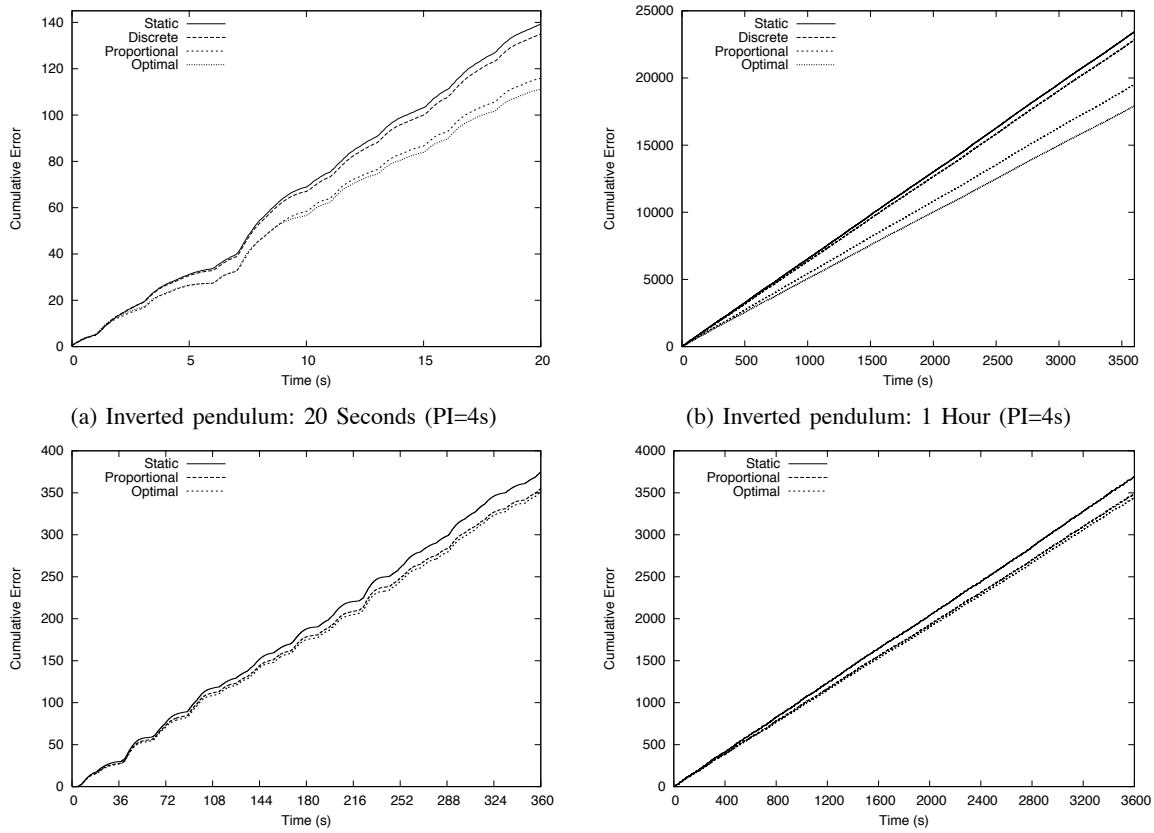
controlled processes (p_i) or weights (w_i) would not materially affect the validity of our solution or results. Both in the first and the second set each controller implements the same parametric control law obtained by standard pole placement, which is parametrized on the sampling period.

For each of the resource allocation policies, we ran the three controllers (in both sets) for one hour and randomly generated perturbations for each controller with different average perturbation intervals. The desired global resource utilization factor (U_d) for the three controllers is $U_d = 97\%$ because we reserve 3% of the resource capacity to allow the execution of other general-purpose processes. 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.

In the first set of workloads, each controller (in charge of a simulated inverted pendulum) 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. In this case, 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. However, in the second set of workloads, all controllers (each in charge of a simulated ball & beam) are slower than those in the first set: the worst-case execution time is 0.135s and the sampling period (h_i) is within 0.3s and 0.5s.

3) Performance analysis: For both sets of workloads, we evaluated the control performance of the four different policies by looking at 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 each experiment lasts).

Figure 4(a) shows the results with the first set of workloads, three inverted pendulums, running for 20s with perturbation interval $= 4$s. Figure 4(b) shows the performance of the same experiment over the course of one hour. In Figure 4(a), we see large and increasing gaps between the cumulative error of the static/discrete policies and the proportional/optimal policies. These results show both that our feedback based resource allocation policies improve overall control



(a) Inverted pendulum: 20 Seconds (PI=4s)

(b) Inverted pendulum: 1 Hour (PI=4s)

(c) Ball & beam: 360 Seconds (PI=20s)

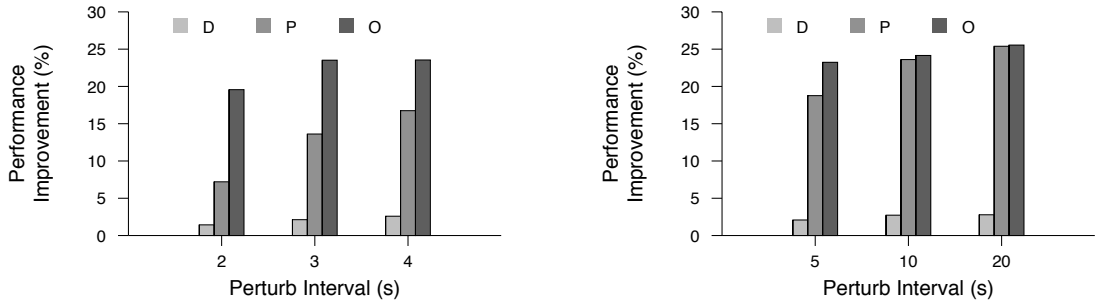
(d) Ball & beam: 1 Hour (PI=20s)

Fig. 4. Performance in terms of Cumulative Error (PI: perturbation interval)

systems performance and that the optimal policy achieves the highest overall control performance improvement. Figure 4(c) and 4(d) show the results with the second set of workloads (three ball & beams) running for 360s and one hour with perturbation interval $= 20$s. The results are very similar to those shown in Figure 4(a) and (b) though the gaps between the cumulative error of the static policies and the proportional/optimal policies are smaller. This is because a ball & beam system is much slower than an inverted pendulum system. For clarity we left out the performance results for the discrete policy; its performance was nearly identical to that of the static policy.

The main contribution of the paper is summarized in Figures 5 and 6, which show the performance improvement for the three policies (D–discrete, P–proportional, and O–optimal) against the static policy. Figures 4 (a-d) were for a specific perturbation interval. Figures 5 and 6 summarize the results for different perturbation intervals in one hour experiments.

Figure 5(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



(a) Without Idle CPU Usage

(b) With Idle CPU Usage

Fig. 5. Inverted Pendulum: Performance Improvement Relative to Static

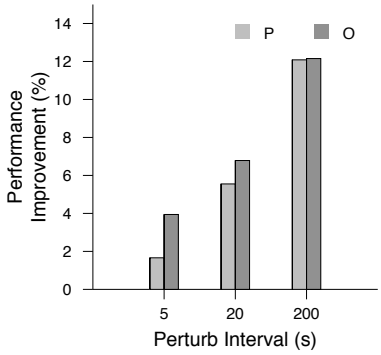


Fig. 6. Ball & Beam: Performance Improvement Relative to Static

one of the pendulums). Figure 5(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 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 20–25% compared to traditional static control; 3) discrete reduces error by only about 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 5(b)), the difference between optimal and proportional decreases. This is because, with perturbation intervals long enough (e.g., 20s), 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. Similarly, as shown in Figure 6, the optimal and proportional polices used in the second set of workloads (ball & beams) also improve the control performance, but by lower percentages (2–12%) with the perturbation intervals ranging from 5s to 200s. This is again because ball & beam is a slower

system.

Generally, from both Figures 5 and 6, we conclude that in the centralized architecture, feedback based resource allocation policies significantly outperform the traditional static resource allocation strategy and the optimal policy is the best.

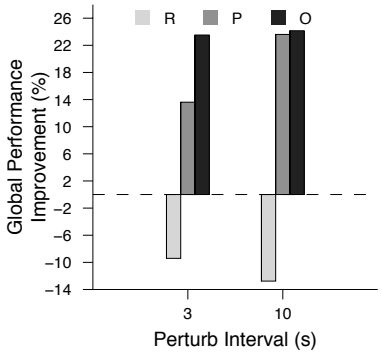


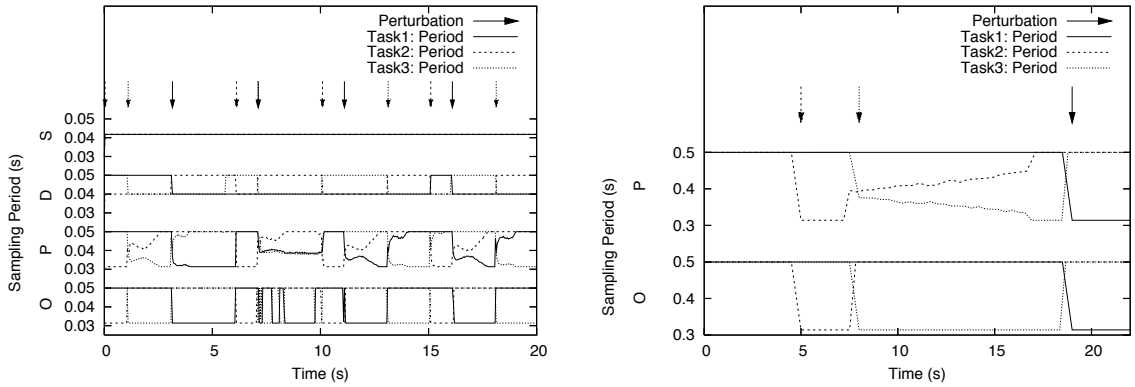
Fig. 7. Performance Improvement Compared to Random Case (Relative to Static)

In order to demonstrate that the benefits associated with our dynamic feedback-based resource allocation policies are due to the resource allocation decisions they make and not simply from some effect inherent in varying resource allocations, we also compared them to a random dynamic resource allocation policy. The random policy allocates the CPU 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 resource allocation policies. Figure 7 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.*, 10s), the relative performance of the random case gets worse while those of proportional and optimal improve.

4) CPU usage and overhead analysis: In order to further demonstrate the feasibility of this approach, we also thoroughly investigated its resource utilization and analyzed the overhead incurred by the four different policies. We first detail the effects on the execution rate (CPU usage) of each controller when using the different resource management policies we have presented. We then present the resource utilization and the overhead they incur.

Figure 8(a) shows the variation on the control tasks periods when using the four different policies (S–static, D–discrete, P–proportional, and O–optimal) with the first set of workloads

with perturbation interval=4s. Figure 8(b) shows a more detailed view of the rate adaptation for the proportional and optimal policies using the second set of workloads with perturbation interval=20s. In order to look into the progress of the three control tasks, we only show the first 20s of the one hour run. In both figures, the *x*-axis represents the time and the *y*-axis represents both the perturbations and the corresponding controller period for each of the policies. Perturbations are shown as arrows with different line styles indicating which control loop they are affecting. Controller periods vary from 0.03s to 0.05s in Figure 8(a) and from 0.3s to 0.5s in Figure 8(b)).



(a) Inverted pendulum (PI=4s)

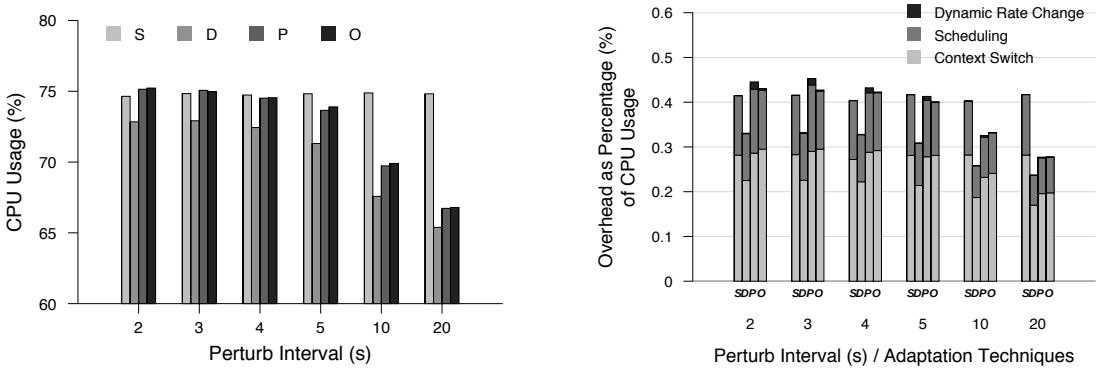
(b) Ball & beam (PI=20s)

Fig. 8. Perturbations vs. CPU Allocations (PI: perturbation interval)

In Figure 8(a), with the static policy, every control task has the same fixed sampling period, $h_i \approx 0.042$s, which comes from $h_i = c_i/r_i = \frac{0.0135}{U_d/3}$ and which does not change regardless of when the perturbations occur. With discrete, proportional, and optimal, 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.

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 IV-A. 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, $h_i \approx 0.032$, as long as the other two can run at or above their lowest rate, $h_i = 0.05$s. With proportional, any variation in the errors causes a proportional rate adjustment among the three controllers, resulting in a larger number of adjustments. The ball & beam results for the optimal and proportional policies shown in Figure 8(b) are similar to although slower than the those shown for these two policies on the inverted pendulums in Figure 8(a).



(a) CPU Usage in 1 Hour

(b) Overhead in 1 Hour

Fig. 9. CPU Usage vs. Overhead

Figure 9(a) shows the measured total CPU usage of all control tasks with the four policies using the first set of workloads. 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 9(b) shows the overhead introduced by the four different policies (measured from our implementation) using the first set of workloads. 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. As expected, and although still negligible, proportional exhibits the largest overhead due to its

more frequent changing of rates, as explained in Section IV-A.

V. A Distributed Solution

A. System architecture

The uniprocessor real-time control system architecture illustrated in Figure 1 can be extended to a network architecture, shown in Figure 10. In the networked architecture, each plant (described by equations (1) and (2)) and controller constitute a closed loop and each controller is assumed to be implemented in separate nodes. The nodes perform sampling, control algorithm computation and/or actuation and exchange sample and/or control signal information through a network shared by all nodes of all control loops.

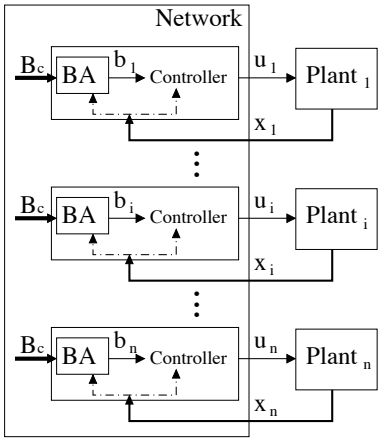


Fig. 10. Distributed Feedback architecture

In the networked architecture each control loop, sampling, control algorithm computation, and actuation are (or may be) performed in separate nodes and network bandwidth is the scarce shared resource that must be allocated among the control tasks. For each control loop, at each sampling period h_i, the time spent on the messaging required to perform each closed loop operation—which may include data exchange from sensor to controller, and from controller to actuator— is given by m_i (messaging time). m_i is assumed to be constant, as we further explain in Section VI, and should occur within each period. Analogous to CPU utilization, the partial utilization factor of each control loop configuration on the network, also called *bandwidth*, b_i (25), is the resource requirement that each control loop configuration needs for a given period. Any change in b_i will imply a change in h_i (and vice-versa).

$$b_i = \frac{m_i}{h_i} \tag{25}$$

B. Problem to be solved

Taking into account the bandwidth requirements of each control loop given in (25), the distributed allocation problem is analogous to the centralized problem: how to assign the scarce network bandwidth to the set of control loops such that all messages are schedulable and overall control performance is maximized, knowing that the controllers will provide better performance given more network bandwidth.

In the networked architecture, we can distinguish two scenarios. Given a master/slave architecture where the master node is responsible for the bandwidth allocation and coordinates the communication activities of all nodes[3], the problem can be solve using the optimal centralized strategy presented in Section III-A. The FTT paradigm applied to networks by Pedreiras and Almeida [22] is an existing scheduling framework that would support the application of the optimal policy in a master/slave architecture. In such a configuration, the sampler node of each control loop can send the state of the controlled plant to the master node. The master node, knowing the state of all controlled plants and all control loop message requirements, can act as a resource manager in the same way as in the uniprocessor architecture, where network bandwidth b_i in (25) is equivalent to CPU resource r_i in (4).

The primary limitation of this centralized approach to networked control is that it assumes the existence of a master node that knows the current state of all controlled plants and all control loop message requirements. Maintaining a global state can be problematic in practice, introducing additional latency and a central point of failure and increasing overall system messaging traffic. Without assuming that all the information required to solve the allocation problem is localized in a single node (master), a different solution is required.

C. Distributed Problem Formulation

To solve the bandwidth allocation problem for the case of a networked architecture lacking a master unit, we present a bandwidth allocation policy that is distributively and cooperatively performed by all control loops. We assume only two pieces of *global* knowledge: global bandwidth utilization factor, or network capacity, B_g, which can be either the total capacity of the network or a desired fraction of the total bandwidth determined by a system parameter setting, and current network bandwidth utilization factor B_c, which we assume can be obtained at each

[3]In a master/slave configuration, slave nodes send messages as instructed by the master. Therefore, when a message is transmitted, all the bandwidth is guaranteed to the transmitter, which assures a constant message transmission time.

node via continuous monitoring of the traffic on the network. Using B_c, B_g and the plant error ($|\vec{x}_i|$ as defined in (3)), each control loop must determine the appropriate period h_i (or bandwidth to be consumed, b_i) within $h_{i,min}$ and $h_{i,max}$ to improve the overall control performance (which is represented in Figure 10 by *BA*, bandwidth allocation).

At each execution, each control loop will cooperatively perform the bandwidth allocation, setting its period according to the following specifications:

- For each control loop, the bigger the error (3), the higher the bandwidth b_i (or the shorter the period h_i) to be allocated.
- For the set of n networked control loops, the bandwidth utilization factor must not exceed the global bandwidth utilization factor (26).

$$\sum_{i=1}^{n} b_i \leq B_g \tag{26}$$

These two specifications may conflict; the first indicates that nodes with higher error should have higher bandwidth allocations while the second restricts overall bandwidth.

A first step to the local bandwidth allocation is to enable each control loop to obtain the available (unused) bandwidth B_a. Given the current and global bandwidth utilization factors B_c and B_g, each control loop can calculate the available bandwidth B_a by taking the difference between B_g and B_c (minus the node's current bandwidth allocation b_i), as specified in (27), where m_i is the worst case transmission time required for each closed loop operation and h_i is the current period (recall (25)).

$$B_a = B_g - (B_c - b_i) = B_g - (B_c - \frac{m_i}{h_i}) \tag{27}$$

By taking into account the available bandwidth (27), the shortest possible period that a control loop suffering error in its controlled plant can infer is given by (28).

$$h_{i,min} = \frac{m_i}{B_a} \tag{28}$$

Because it depends upon other node's allocations, $h_{i,min}$ will vary over time, depending on the current network load. In the optimal policy for the centralized architecture, $h_{i,min}$ and $h_{i,max}$ are design specifications and each control task, at any given time, is allowed to execute with either one of them. In the distributed architecture, $h_{i,max}$ is still a design specification. However, $h_{i,min}$ varies, and no assurances can be given as to whether its value will significantly differ from $h_{i,max}$, as it does in the optimal policy.

VI. Distributed strategy to resource management

We present a fully distributed heuristic technique for optimization of control systems performance when resources are limited and allocated locally as a function of the state of the controlled systems. See Velasco *et al*. [23] for an extended explanation of this technique.

A. Heuristic state feedback based resource allocation policy

Each control loop is assumed to execute with $h_{i,max}$ if $e_i = 0$. If a perturbation affects a controlled system, its error will increase and thus the period of the control loop should decrease, down to the shortest value ($h_{i,min}$), if possible. Similarly, when its error decreases, its period should increase, up to the longest value $h_{i,max}$.

A heuristic technique that assures this behaviour while meeting the total bandwidth utilization constraint (26) can be defined as follows. We call the period to be inferred by each control loop h_i^{next}. The default period for each control loop in the absence of error is $h_i^{next} = h_{i,max}$. For each control loop experiencing error, we must ensure the possibility of the period being equal to the shortest value, that is, $h_i^{next} = h_{i,min}$ (taking into account the available bandwidth obtained through (27) and (28)), which will provide the best control performance. If we define $\Delta h_i = (h_{i,max} - h_{i,min})$, then $h_i^{next} = h_{i,min} + \Delta h_i * p_i$, where p_i=0 or 1. When $p_i = 0$, $h_i^{next} = h_{i,min}$, and when $p_i = 1$, $h_i^{next} = h_{i,max}$.

The inferred period h_i^{next} for each control loop should also take values within $h_{i,min}$ and $h_{i,max}$, depending upon the magnitude of the error, in such a way that not all bandwidth is consumed all the time when the controlled system experiences error. This can be achieved by increasing or decreasing the amount added to $h_{i,min}$ to obtain h_i^{next}. This, in turn, can be accomplished by replacing p_i with a function f defined within 0 and 1 inversely proportional to the error (see (29) for a summary of the heuristic). Such behaviour for the f function can be achieved by a decreasing exponential function of the error[4], that is, $f(e_i) = e^{-e_i}$.

$$
\begin{aligned}
& h_i^{next} = h_{i,min} + \Delta h_i * f(e_i),\ 0 < f(e_i) \leq 1 \\
& if\ e_i\ increases\ \rightarrow\ f(e_i)\ decreases\ \rightarrow\ h_i^{next}\ tends\ towards\ h_{i,min} \\
& if\ e_i\ decreases\ \rightarrow\ f(e_i)\ increases\ \rightarrow\ h_i^{next}\ tends\ towards\ h_{i,max}
\end{aligned}
\tag{29}
$$

[4]A similar behaviour could be obtained by simply using the inverse of the error, $f(e_i) = \frac{1}{e_i}$. However, this may introduce computation problems when $e_i = 0$

Finally, similar to the weight w_i used in the optimal policy to allow appropriate comparisons among the control loops in the system, we also define a parameter c_i, called *criticalness*, to be specified offline that will allow the system to regulate how fast each networked control loop will change from $h_{i,min}$ to $h_{i,max}$ and vice-versa. That is, it will specify how quickly each control loop will increase or decrease its period according to error. Higher values for c_i will imply more abrupt changes on the period. Because each control loop is *self-assigning* bandwidth in isolation, the nodes must not be allowed to instantly assign themselves all of the available bandwidth at once because bandwidth may be also demanded by others control loops. The criticalness parameter is used to control this problem.

In summary, the heuristic function used for the local bandwidth management that gives, at each closed loop operation, the period h_i^{next} (or bandwidth allocation) within $h_{i,min}$ and $h_{i,max}$ as a function of the error ($e_i = |\vec{x}_i|$) and taking into account the global bandwidth constraints is given by (30)

$$h_i^{next} = (h_{i,max} - h_{i,min})\, e^{-c_i|\vec{x}_i|} + h_{i,min} \tag{30}$$

Taking into account (27) and (28), equation (30) can be rewritten as

$$\begin{aligned} h_i^{next} &= (h_{i,max} - h_{i,min})\, e^{-c_i|\vec{x}_i|} + h_{i,min} = \\ &= (h_{i,max} - \frac{m_i}{B_a})\, e^{-c_i|\vec{x}_i|} + \frac{m_i}{B_a} = \\ &= (h_{i,max} - \frac{m_i}{B_g - (B_c - \frac{m_i}{h_i})})\, e^{-c_i|\vec{x}_i|} + \frac{m_i}{B_g - (B_c - \frac{m_i}{h_i})} \end{aligned} \tag{31}$$

where, for each control loop, $h_{i,max}$ and c_i are parameters specified at system setup, B_g and B_c are assumed to be known, m_i is constant, and h_i and $|\vec{x}_i|$ are the current control loop period and error (and thus locally known).

B. Controller design

The application of the heuristic policy requires the implementation of controllers capable of running with different sampling frequencies, as also happens in the optimal policy (section III-B).

The derivation of the heuristic distributed allocation policy in the form of a simple equation (31) enables the integration of the bandwidth dynamics that this equation expresses into the discrete state space representation of each closed loop system. This allows the direct design of controllers using classical techniques that assure stability and meet control performance requirements given different sampling periods.

That is, augmenting the discrete state space representation of each control loop (18) by the period (given by (30)) as a new state variable, we obtain the system described by (32) (where $h_{n+1} = h^{next}$).

$$\begin{bmatrix} \vec{x}_{n+1} \\ h_{n+1} \end{bmatrix} = \begin{bmatrix} \Phi(h) \cdot \vec{x}_n \\ (h_{max} - h_{min})\, e^{-c|\vec{x}_n|} + h_{min} \end{bmatrix} + \begin{bmatrix} \Gamma(h) \\ 0 \end{bmatrix} u_n \tag{32}$$

In this case, the nonlinear characteristics of the extended model (32) require the excitation input u_n to be given by a control law designed using nonlinear techniques [24]. By doing so, we obtain controllers that accept different execution rates according to the allocation given by the distributed bandwidth policy specified in (30). For further explanation of the extended model and the application of nonlinear techniques to the design of the control law, see Velasco *et al.* [23].

VII. Performance evaluation for distributed resource allocation

A. Distributed feedback resource allocation

To evaluate the performance of the distributed architecture, we instrumented RBED in such a way that real-time tasks emulate nodes and the CPU is treated as a network. Compared to the centralized mechanism, the distributed feedback resource allocation policy (presented by equation (30)) is simpler. In this case, it does not require centralized dynamic rate adjustment; Instead each emulated distributed node adjusts its execution frequency locally according to its error, adjusting the bandwidth allocated to it accordingly.

Controller
{
$x_i := \mathbf{read\_input}()$
$e_i := |x_i|$
$B_a := B_g - (B_c - \frac{m_i}{h_i})$
$h_{i,min} := \frac{m_i}{B_a}$
$h_i^{next} := (h_{i,max} - h_{i,min})\, e^{-c_i e_i} + h_{i,min}$
$u_i := \mathbf{calculate\_output}(h_i^{next})$; $\mathbf{send\_output}(u_i)$
}

Fig. 11. Pseudo-code for Controller in Distributed Architecture

Figure 11 shows the pseudo-code for a controller in the distributed architecture. First, the code implements the bandwidth management, computing h_i^{next} after obtaining the state of the controlled system x_i and the available bandwidth B_a. Second, the code computes the control signal and sends it to the controlled system.

We have also emulated the static resource allocation policy, as in the centralized case, to use as a baseline for analyzing the performance of our distributed solution. We did not emulate the

proportional or discrete policies because they would require the same centralized information as the optimal policy.

B. Workload generation

In the experiments evaluating the performance of distributed resource allocation policies we used the second set of workloads which consists of three control loops, each of which controls a simulated ball&beam (see Section IV-C.2 for more details). All the tasks' parameters and system configuration are the same as those given previously, except that a task's processing time is now its message transmission time m_i instead of its CPU execution time c_i.

C. Results for distributed resource allocation

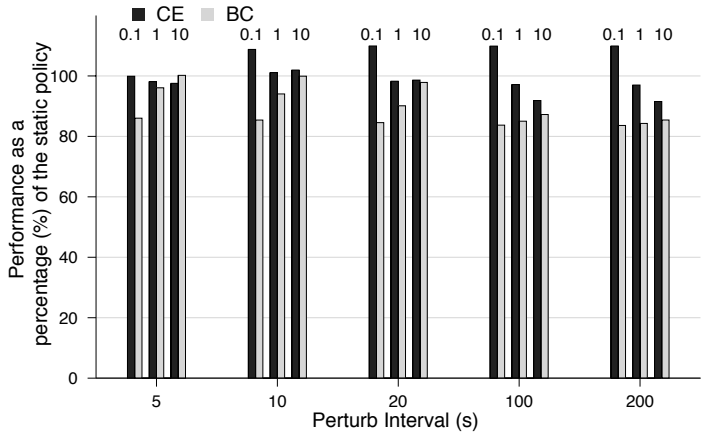


Fig. 12. Analysis of control error (CE) and bandwidth consumption (BC) of the dynamic allocation technique compared to the static, for different perturbation intervals and for different criticalness values.

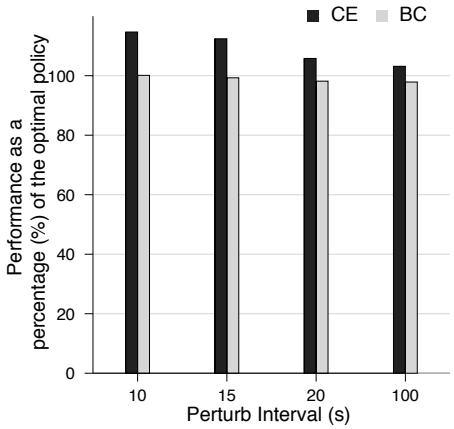


Fig. 13. Analysis of control error (CE) and bandwidth consumption (BC) of the dynamic allocation technique (with criticalness=10) compared to the optimal, for different perturbation intervals

Figure 12 shows the control performance in terms of control error (CE) and bandwidth consumption (BC) of our dynamic distributed bandwidth allocation technique relative to the static policy for different criticalness values and perturbation intervals. The data of Figure 12 is in percentage of the static policy; a higher bar means either more control error (worse control performance relative to the static policy) or more bandwidth consumption compared to the static approach. The numbers above the bars represent the different criticalness values; $c = 10$ implies aggressive bandwidth management while $c = 0.1$ implies timid management. For example, with perturbation interval of 100s and c of 10, the distributed policy has only 92% of the control error using 87% of the bandwidth compared to the static policy.

Regardless of perturbation interval, more aggressive bandwidth management (with c from 0.1 to 10) leads to less control error (better control performance relative to the static policy) but greater bandwidth consumption. Up to perturbation intervals of 20s the intervals are short enough so that all of the available network bandwidth can be allocated to the control loops with a big value of c (e.g., 10). At perturbation intervals greater than 20s the intervals are long enough so that, even with a big value of c, a portion of bandwidth is saved by the control loops when there is no error. Figure 12 shows control error is more reduced in this second scenario. Although there is no control performance improvement (control error reducing) with $c = 0.1$—there is, in fact, a slight degradation—the bandwidth savings are important, and adequately trading off control performance and bandwidth consumption is crucial. From Figure 12 we see that whatever the perturbation interval and criticalness parameter, the dynamic approach to bandwidth management always saves bandwidth (less than 100%) compared to the static approach.

Figure 13 shows the control performance in terms of control error (CE) and bandwidth consumption (BC) of our dynamic distributed bandwidth allocation technique (with criticalness=10) relative to the optimal policy for different perturbation intervals. The data of the optimal policy is the same as that in the experiments performed on the second sets of workload under the centralized approach. When the perturbation interval becomes bigger, the control performance of the distributed policy approaches that of the optimal policy; the control error of the distributed policy decreases from 114% to 103% when the perturbation interval ranges from 10s to 100s. Regardless of perturbation interval, the distributed policy always uses as much bandwidth as the optimal policy or less.

VIII. Related work

Optimization of control systems performance subject to resource constraints has been examined before, both for uniprocessor architectures and for networked architectures.

A. *Uniprocessor systems*

Seto *et al*. [2] optimized task frequencies at the design stage in order to minimize a control performance index defined over the task set. Rehbinder and Sanfridson [25] proposed an off-line scheduling method based on optimal control theory. Neither of them 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. Beccari *et al*. [26] presented a scheduling technique for adaptation of soft real-time load to available computational capacity in the context of autonomous robot control architectures. Ramanathan [27] presented an overload management for control tasks based on the (m,k)-firm guarantee. Caccamo *et al*. [28] 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 [29] present a case study with hybrid controllers , where the sampling rates are adjusted to avoid CPU overloads. Cervin *et al*. [10], 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*. [30] 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 [31], and by Henriksoon *et al*. [32]. 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.

B. Networked-based systems

While co-design of control and processor allocation/scheduling has received considerable attention in the literature, allocation/scheduling on networked control systems is still an area that need to be further explored. See Tipsuwan and Chow [33] for an overview on control methodologies for these systems.

Static scheduling strategies for networked-based systems are reviewed next. Hong presented a scheduling algorithm to determine data sampling times, so that the performance requirement of each control loop is satisfied as well as the utilization of network resources is considerably increased [34]. We focus on minimizing the network utilization while improving control performance. Branicky *et al.* presented a static scheduling optimization approach based on scheduling and control co-design [3]. Ling and Lemmon presented a method for obtaining specifications on real-time schedulers that assure overall feedback system performance [35]. The technique we present targets similar control performance optimization problems but for the case of dynamic bandwidth allocation.

Dynamic strategies to scheduling and control co-design in networked control systems have appeared lately. Walsh and Yes presented a dynamic arbitration technique to grant network access to the control loop with highest error [36]. Our approach is similar, but focuses directly on bandwidth allocation rather than on priority assignment. The approach presented by Liberatore [37] focuses on the middleware scheduling of network access through a polling scheme. Our resource allocation architecture does not require a master arbitrating the network access. The scheduling method presented by Park *et al.* [38] allows for sampling period adjustment to allocate bandwidth to other type of messages. Although the sampling adjustment is considered on the control analysis, it is not performed according to the dynamics of the control loop as in

our system. Otanez *et al.* [39] reduced traffic for networked control systems using deadbands, an approach that demands a tradeoff between traffic and control performance. We aim at improving control performance while reducing bandwidth consumption, following the scheduling strategy presented by Yepez *et al.* [40].

C. Preliminary work

Some preliminary results of the feedback approach to resource management on uniprocessor systems that are briefly included in this paper were reported previously [20]. The optimal resource allocation policy for control tasks that we describe in Section III-A and was presented previously [12] optimally solves the Quality-of-Control (QoC) scheduling problem formulated by Marti *et al.* [4] but in terms of resource management. The heuristic allocation policy for networked-based systems that we present in Section VI was previously presented in a short paper by the authors [23].

IX. 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 resource constrained environments that allows the system to allocate resources as a function of the state of the controlled systems. We rely 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, our system dynamically adjusts the resources provided to the controllers, and thus the rate at which they operate. This approach has proven to be highly effective in both centralized and distributed control system architectures, significantly outperforming traditional static controllers.

Our centralized resource allocation model relies on complete information about the state of the controlled systems. We have proven 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 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 20% while consuming less resources overall.

Our centralized solution can be applied to any distributed architecture in which global knowledge is available about the state of all controllers. This is a strong limitation, and may not be feasible or practical for all systems. Our distributed allocation policy is designed to avoid this limitation and does not assume centralized knowledge of the state of all of the controllers. It approximates the optimal solution by having each controller adjust its own resource usage according to its error. While theoretically suboptimal, this solution can also provide improved control performance with lower overall resource consumption.

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 or network bandwidth. Additional research is focused on improving the current heuristic-based distributed solution. Through the use of a distributed token-passing algorithm we expect to be able to implement a distributed approximation to the optimal algorithm with better performance than our current distributed solution.

REFERENCES

[1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp. 46–61, Jan. 1973. [Online]. Available: http://ssrc.cse.ucsc.edu/PaperArchive/liu-jacm73.pdf

[2] 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.

[3] M. S. Branicky, S. M. Phillips, and W. Zhang, "Scheduling and feedback co-design for networked control systems," in *Proceedings of the 41th IEEE Conference of Decision and Control*, Dec. 2002.

[4] P. Martí, 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. [Online]. Available: http://ssrc.cse.ucsc.edu/PaperArchive/marti-rtss02.pdf

[5] P. Martí, 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. [Online]. Available: http://ssrc.cse.ucsc.edu/PaperArchive/marti-rtss01.pdf

[6] 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)*, Dec. 2003, pp. 396–407. [Online]. Available: http://ssrc.cse.ucsc.edu/PaperArchive/brandt-rtss03.pdf

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

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

[9] 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*, vol. 30, no. 4, pp. 357–366, 1985.

[10] 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.

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

[12] P. Martí, C. Lin, S. A. Brandt, M. Velasco, and J. M. Fuertes, "Optimal state feedback based resource allocation for resource-constrained control tasks," in *Proceedings of the 25th IEEE Real-Time Systems Symposium (RTSS 2004)*, Dec. 2004, pp. 161–172. [Online]. Available: http://www.cse.ucsc.edu/ lcx/research/papers/rtss04.pdf

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

[14] B. Wittenmark and K. J. Åström, "Simple self-tuning controllers," *Unbehauen, Ed. Methods and Applications in Adaptive Control, Lecture Notes in Control and Information Sciences*, vol. 24, pp. 21–29, 1980.

[15] P. Albertos and J. Salt, "Digital regulators redesign with irregular sampling," in *11th IFAC World Congress (Preprints)*, vol. 8, 1990, pp. 157–161.

[16] 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.

[17] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic scheduling for flexible workload management," *IEEE Transactions on Computers*, vol. 51, no. 3, pp. 289–302, Mar. 2002. [Online]. Available: http://ssrc.cse.ucsc.edu/PaperArchive/buttazzo-ieeetc02.pdf

[18] S. Goddard and L. Xu, "A variable rate execution model," in *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, July 2004, pp. 135–143. [Online]. Available: http://ssrc.cse.ucsc.edu/PaperArchive/goddard-ecrts04.pdf

[19] M. R. Garey and D. S. Johnson, *Computers and Intractability*. W.H. Freeman, 1979.

[20] 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," in *Work in Progress Session of the IEEE Real-Time and Embedded Technology and Applications Symposium (UC Santa Cruz Technical Report UCSC-CRL-04-04)*, Toronto, Canada, May 2004. [Online]. Available: http://www.cs.virginia.edu/rtas04/wip/wip20.pdf

[21] S. Brandt and G. Nutt, "Flexible soft real-time processing in middleware," *Real-Time Systems*, vol. 22, pp. 77–118, 2002. [Online]. Available: http://ssrc.cse.ucsc.edu/PaperArchive/brandt-rts02.pdf

[22] P. Pedreiras and L. Almeida, "The flexible time-triggered (FTT) paradigm: An approach to qos management in distributed real-time systems," in *International Parallel and Distributed Processing Symposium (IPDPS'03)*, Apr. 2003.

[23] M. Velasco, J. Fuertes, C. Lin, P. Martí, and S. Brandt, "A control approach to bandwidth management in networked control systems," in *30th Annual Conference of the IEEE Industrial Electronics Society (IECON04)*, Nov. 2004.

[24] A. Isidori, *Nonlinear Control Systems*. Spring Verlag, 1989.

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

[26] 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.

[27] P. Ramanathan, "Overload management in real-time control applications using (m,k)-firm guarantee," *IEEE Transactions on Parallel and Distributed Systems (Special issue on Dependable Real-time Systems)*, pp. 549–559, June 1999. [Online]. Available: http://ssrc.cse.ucsc.edu/PaperArchive/ramanathan-tpds99

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

[29] A. Cervin and J. Ecker, "Feedback scheduling of control tasks," in *Proceedings of the 39th IEEE Conference of Decision and Control*, June 2000.

[30] G. Buttazzo, M. Velasco, P. Martí, and G. Fohler, "Managing quality-of-control performance under overload conditions," in *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, July 2004.

[31] Q. Zhao and D.-Z. Zheng, "Stable and real-time scheduling of a class of perturbed hybrid dynamic systems," in *IFAC World Congress*, vol. J, 1999, pp. 91–96.

[32] D. Henriksson, A. Cervin, J. kesson, and K.-E. rzn, "Feedback scheduling of model predictive controllers," in *8th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS02)*, Sept. 2002.

[33] Y. Tipsuwan and M. Chow, "Control methodologies in networked control systems," *Control Engineering Practice*, vol. 11, pp. 1099–1111, 2003.

[34] S. H. Hong, "Scheduling algorithm of data sampling times in the integrated communication and control systems," *IEEE Transactions on Control Systems Technology*, vol. 3, no. 2, pp. 225–230, 1995.

[35] Q. Ling and M. Lemmon, "Soft real-time scheduling of networked control systems with dropouts governed by a markov chain," in *Proceedings of the 2002 American Control Conference*, June 2003.

[36] G. Walsh and H. Ye, "Scheduling of networked control systems," *IEEE Control Systems Magazine*, vol. 21, no. 1, pp. 57–65, 2001.

[37] V. Liberatore, "Scheduling of network access for feedback-based embedded systems," in *Quality of Service over Next Generation Internet, SPIE ITCom*, 2002, pp. 73–82.

[38] H. Park, Y. Kim, D. Kim, and W. Kwon, "A scheduling method for network-based control systems," *IEEE Transactions on Control Systems Technology*, vol. 10, no. 3, pp. 318–330, 2002.

[39] P. Otanez, J. Moyne, and D. Tilbury, "Using deadbands to reduce communication in networked control systems," in *Proceedings of the 2002 American Control Conference*, 2002.

[40] J. Yépez, P. Martí, and J. Fuertes, "Control loop scheduling paradigm in distributed control systems," in *29th Annual Conference of the IEEE Industrial Electronics Society*, Nov. 2003.