Real-Time Systems, 30, 187–215, 2005

DOI: 10.1007/s11241-005-2461-y

A Technique for Adaptive Scheduling of Soft Real-Time Tasks*

G. BECCARI
S. CASELLI
F. ZANICHELLI
Dipartimento di Ingegneria dell'Informazione, University of Parma, Italy

Published online: 06 July 2005

Abstract. A number of multimedia and process control applications can take advantage from the ability to adapt soft real-time load to available computational capacity. This capability is required, for example, to react to changed operating conditions as well as to ensure graceful degradation of an application under transient overloads. In this paper, we illustrate a novel adaptive scheduling technique based on rate modulation of a set of periodic tasks in a range of admissible rates. By casting constraints on rate ranges in a linear programming formulation, several adaptation policies can be considered, along with additional constraints reflecting various application requirements. The paper investigates the effectiveness of rate modulation strategies both on simulated task sets and on real experiments.

Keywords: adaptive scheduling, overload management, priority scheduling, rate modulation, graceful degradation, robotic applications, multimedia applications

1. Introduction

Real-time systems are being increasingly designed for complex applications and dynamic environments. For these applications, it is sometimes impractical or impossible to provide static guarantees to real-time computations. Consider an autonomous vehicle performing a surveillance task. The vehicle runs a number of sensor acquisition and interpretation tasks while patrolling an environment and avoiding obstacles. If ambiguous sensory values or intrusion clues are detected, the vehicle might be required to activate costly sensory operations, such as laser scanning or high resolution stereo vision, which otherwise would be normally kept off. If a fast moving intruder entity is perceived, the system might be required to activate an urgent computation to plan a path blocking the intruder.

In dynamic environments, and with the sensory and feature rich systems which are increasingly common nowadays, it becomes very difficult or impossible to guarantee real-time computations under all possible situations. Furthermore, many computational techniques, such as those derived from artificial intelligence, providing intelligence and flexibility to applications, exhibit highly variable computation times (e.g. search algorithms, motion planning, learning components). Hence, adopting their worst case

*Partial support for this research has been provided by MURST, Italy (PRIN project ISIDE on "Dependable reactive computing systems for industrial applications" and special project "RoboCare" funded by L. 449/97), and by ASI, Agenzia Spaziale Italiana (contract I/R/134/00).

execution time results into an unacceptable under-utilization or into a non-schedulable system.

On the other hand, real-time system computations may offer sources of flexibility which could actually help in determining a feasible schedule. For example, the sampling period of digital process control systems can often be adjusted within a given acceptable range, depending on the time constants of the plant, as long as the feedback control algorithm parameters are designed taking into account the chosen sampling rate (Kuo, 1992; Franklin, 1995). Likewise, the rates of sensory information acquisition and processing tasks can often be tuned in ranges based on the external dynamics. Suppose that the speed of an object must be inferred from consecutive camera frames. In order to perform the speed computation task, a range of acquisition rates could be acceptable, as long as the chosen period is known.

The prevailing real-time scheduling paradigms, both static, such as rate monotonic (RM) scheduling (Liu and Layland, 1973; Lehoczky et al., 1989), and dynamic, such as earliest deadline first (EDF) scheduling (Liu and Layland, 1973; Stankovic et al., 1998), do not fit well the requirements of advanced real-time applications in dynamic environments. These motivations have led to the emergence of adaptation as a major research issue in real-time scheduling (Beccari et al., 1999; Jones et al., 1997; Kuo and Mok, 1997; Lu et al., 1999, 2000; Nieh and Lam, 1997; Seto et al., 1996, 1998; Shin and Meissner, 1999; Stankovic et al., 1999; Steere et al., 1999).

In this paper, we describe a novel *adaptive scheduling* technique based on *rate modulation* of a set of periodic tasks in a range of admissible rates. During increasing load situations, such as those described in the previous examples, adaptation helps in maintaining the real-time system in a safer state by preventing starving tasks or reducing their number. In general, adaptation based on rate adjustment allows dynamic re-distribution of computational power to better fit the current application requirements. By casting constraints on rate ranges in a linear programming formulation, we show how several adaptation policies can be considered, along with additional constraints reflecting various application requirements.

Drawing from the domain of autonomous robots (Beccari et al., 1998, 1999), where task priority plays an essential role and high priority computations should be protected from missing their deadlines, we focus on adaptation of otherwise static real-time load consisting of periodic threads. Additionally, we restrict our attention to uniprocessor systems.

The paper is organized as follows. Section 2 reviews the related research done in the areas of adaptive scheduling and overload management. Section 3 presents and motivates the basic adaptation strategy proposed in this paper. Section 4 proposes a general framework for rate modulation yielding a formulation in terms of a linear programming problem. Section 5 describes several adaptation policies taking into account additional task requirements and specializing or extending the linear programming formulation. It also provides an example of constraints leading to a more complex computational problem. Section 6 discusses issues arising in transition to a new set of rates. Section 7 presents results from off-line assessments of the various rate adaptation policies, along with results and traces from their actual experimentation on a Solaris system. Section 8 outlines areas of further work and summarizes the contribution of this paper.

2. Related Work

The use of predefined rate schedules for different operating modes has been for a long time an empirical technique to manage variability in computational load. However, in the real-time literature until recently relatively few approaches have addressed the overload problem, the prevailing doctrine being that overload should never occur in real-time systems. A fine overview of prior art in overload management and adaptive scheduling techniques for real-time systems is given in Lu et al. (1999). Mechanisms for detecting and handling timing errors, including overloads, are discussed in Stewart and Khosla (1997), with emphasis on a specific application-oriented operating system (Stewart et al., 1992).

The need for an adaptive management of the QoS has been widely recognized in the domain of distributed multimedia systems. The work in Li and Nahrstedt (1998) exploits digital control theory to determine the states and the control algorithms to adapt the QoS to the dynamics of the distributed system. To this purpose, a task control model is developed where equilibrium and stability analyses are carried out for a PID control algorithm. A graceful degradation of the communication subsystem is obtained in Abdelzaher and Shin (1998) by means of QoS contracts specifying degraded acceptable QoS levels. Under overload or underutilization conditions, a QoS optimization process attempts at maximizing the aggregate reward, given the information on rewards and violation penalties specified by QoS contracts. Another framework for rate-controlled scheduling in multimedia applications is described in Yau et al. (1997). The on-line scheduler and admission control mechanisms allow applications to effectively adapt their reserved rates to actual execution rates.

In a different application domain such as robotics, the need for adaptation capabilities in real-time systems arises in so-called hybrid robot control architectures (Musliner et al., 1993; Schoppers, 1994). Our initial ideas on adaptive scheduling have been presented in Beccari et al. (1999), drawing their motivations from the needs of autonomous robots control architectures.

Significant research has been also devoted to schedulers providing some degree of adaptation to cope with dynamic, overloaded environments. For example, Jehuda and Israeli (1998) propose an automated meta-controller for adaptable real-time systems. The high-level meta-controller determines dynamic reconfigurations in connection with system mode changes. The need for scheduling systems providing real-time guarantees to a subsect of tasks withing a general operating system has been emphasized in Stankovic et al. (1996). The SMART scheduler (Nieh and Lam, 1997) dynamically balances between the needs of real-time and traditional applications, providing dynamic feedback to allow them to adapt to the current load. It considers a common importance attribute for both kinds of applications based on priorities and *weighted fair queueing*, using an urgency mechanism based on earliest deadline scheduling to optimize the order in which tasks are serviced. The Rialto scheduler (Jones et al., 1997) allows the applications to specify minimum guaranteed execution rates through CPU reservations whereas the feasibility of time constraint requests are analyzed by reasoning on a *precomputed scheduling graph*. In the scheduling system described in Steere et al. (1999) each thread is allocated a percentage of CPU cycles over a period of time and a feedback controller is exploited to monitor the rate of progress for the

thread as well as to compute new proportions and periods. This work, however, is oriented to general operating systems, and does not address the issue of real-time tasks performance.

Load-adjustable algorithms (Kuo and Mok, 1997) and value-based policies (Buttazzo and Stankovic, 1993; Koren and Shasha, 1992) are the main techniques proposed for graceful recovery from overload. A load adjustment mechanism is proposed in Kuo and Mok (1997) in order to handle periodic processes with varying temporal parameters. The aim of this work is to determine feasible time parameter configurations (execution time C and period T) and thus modify the real-time computation for collections of tasks. The configuration selection problem is solved by a harmonic approach achieving the maximum exploitation of the computational resources under any time parameter configuration. While appealing, this approach does not lend itself to many real-time systems, where execution times, in spite of their variability, cannot be set or chosen by the designer.

Value-based policies, e.g. RED (Buttazzo and Stankovic, 1993) and D_{over} (Koren and Shasha, 1992), adopt reject procedures that suspend low value tasks according to the application privileges they own. This approach is particularly suited for hard real-time applications. In many real-time architectures, while it is acceptable that certain soft real-time threads are computed at a lower rate, typically few or no threads can be rejected for an arbitrary amount of time without a serious impairment of system functionality.

Baruah and Haritsa (1997) propose a scheduling algorithm maximizing the effective processor utilization during overload, given a minimum slack factor for all tasks. The algorithm is developed in an EDF scheduling framework, hence it does not guarantee which tasks will not be affected by overload.

Other research deals with real-time policies compliant with soft time constraints. For instance, the jitter analysis in Baruah et al. (1997) makes the assumption that inaccuracy characterizes actual systems, hence some jitter, e.g., the uncertainty of the arrival times of individual frames in a communication system, should be taken into account during the design of real systems. The skip approach in Caccamo and Buttazzo (1997) accepts more flexible timing constraints than those allowed by usual real-time approaches. It assumes that certain tasks could abort some instance during a periodic execution, especially for data transmission applications. In our view such a soft real-time approach is well-suited also for other types of applications, such as process or robot control architectures, although some application-specific framework is required.

In Stankovic et al. (1999), Lu et al. (1999, 2000) the authors assume a flexibility in timing requirements similar to the one considered in this paper. To address the dynamics of the environment, they propose a modified EDF adaptive scheduling framework based on feedback control methods and use feedback control loops to maintain a satisfactory deadline miss ratio when task execution times change.

An interesting technique for overload management in hard real-time control applications is described in Ramanathan (1999). The author presents a scheduling policy deterministically guaranteeing m out of any k periodic task activations, along with a methodology able to minimize the effects of missed control-law updates. This work provides a solid foundation to graceful degradation policies of periodic real-time tasks. However, unless the overload duration is very short, the application could be significantly impaired by the loss of periodic execution for a number of real-time tasks.

The contributions in Buttazzo et al. (1998, 2002), Seto et al. (1996) and Shin and Meissner (1999) can be directly compared to the research described in this paper, as they all refer to periodic processes and pursue adaptation by changing task rates. In Buttazzo et al. (1998, 2002) periodic computations are modeled as springs with given elastic coefficients and minimum lengths. Requested variations in task execution rates or overload conditions are managed by changing the rates based on the springs' elastic coefficients. Compared to the general framework described later in this paper, the solution computed by solving the spring problem corresponds to a single rate adaptation policy implicitly encoded in the elastic coefficients. We feel that the user would be more comfortable managing concepts such as task values and priorities rather than spring coefficients. Furthermore, we show that several adaptation policies, emphasizing different criteria relevant to applications, can be made available to designers. Finally, the spring-based formulation does not allow additional constraints to be taken into accounts, nor does it provide a clear task value-to-elasticity mapping.

In Seto et al. (1996) task rates are allowed to vary within given ranges. Each task is assumed to be characterized by a performance index such that the index is a monotonically decreasing and convex function of the task rate. An optimization problem is then formulated as a nonlinear programming problem, and task rates are determined by means of a combination of search and Lagrangian multipliers. The approach, however, is conceived for digital control systems and tackles control system design together with task scheduling. For this reason, it cannot deal with tasks unrelated to the control function or lacking an associated performance index, such as tasks devoted to input data processing. Furthermore, the proposed technique, due to its computational complexity and its interaction with digital control synthesis, is only suitable for off-line, static schedulability analysis.

The work in Shin and Meissner (1999) aims to extend the approach in Seto et al. (1996) to make it suitable for on-line operation in multiprocessor systems. The paper shows the development of a performance index for a specific example, however the approach remains restricted to digital control systems or similar applications. Load adaptation is pursued by means two mechanisms: task reallocation to other processors, assumed to be uniform and without communication and migration costs, and period extensions in fixed steps. Unlike our own work, the proposed heuristics are restricted to the case of simply periodic processes, i.e. where all task frequencies are harmonic.

The aforementioned papers address a variety of problems similar to the one discussed in the present paper and investigate useful related approaches. The specific problem described in this paper is highly relevant in application domains such as process control and multimedia, and is addressed by means of a general rate adaptation framework.

3. The Adaptation Strategy

Let $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_N\}$ be a set of N independent real-time periodic tasks. Each task τ_i has a *worst case execution time (WCET)* C_i, a relative deadline D_i (relative to its *ready*/*request* time), and a period T_i, i.e., τ_i should be executed once every T_i time units. We assume that $\mathcal{T}$ is partitioned in two subsets $\mathcal{T}_h$ and $\mathcal{T}_s$, so that $\mathcal{T} = \mathcal{T}_h \cup \mathcal{T}_s$ and $\mathcal{T}_h \cap \mathcal{T}_s = \emptyset$. $\mathcal{T}_h$ is the

set of hard real-time tasks and $\mathcal{T}_s$ is the set of soft real-time tasks. Tasks in $\mathcal{T}$ are sorted by increasing period, i.e.,

$$T_i \leq T_{i+1}, \quad i = 1 \ldots N-1. \tag{1}$$

For the purpose of this paper, we assume that soft real-time tasks are characterized by a range of admissible rates, whereas hard real-time tasks are characterized by fixed rates.

Tasks in $\mathcal{T}_s$ satisfy the following hypothesis of non-null *rate modulation range*:

$$\begin{aligned} &T_{i,\min} \leq T_i \leq T_{i,\max} \\ &T_{i,\min} \neq T_{i,\max} \end{aligned} \tag{2}$$

i.e., under proper conditions, the periods of soft real-time tasks can be adjusted within their allowed range. For each task in $\mathcal{T}_s$, a preferred *nominal period* $T_{i,n}$ within the range (2) can also be defined.

Tasks in $\mathcal{T}_h$ are characterized by a null modulation range:

$$T_{i,\min} = T_i = T_{i,\max} \tag{3}$$

Tasks are scheduled by *Rate Monotonic* (RM) scheduling (Liu and Layland, 1973) and priorities are set *a priori* accordingly: namely, if P_i is the priority of τ_i and P_{i+1} is the priority of τ_{i+1}, then $P_i > P_{i+1}$. The processor *Utilization Factor* of each task τ_i is $U_i = C_i/T_i$. The least upper bound for a set of N tasks to be feasibly scheduled by the RM algorithm is $L(N) = N(2^{\frac{1}{N}} - 1)$, i.e., if $\mathcal{U}(\mathcal{T}) = \sum_{i=1}^{N} C_i/T_i \leq L(N)$ then the task set is guaranteed to be schedulable by the RM algorithm (Liu and Layland, 1973). Later research in RM scheduling has determined improved least upper bounds for $\mathcal{U}(\mathcal{T})$ along with necessary and sufficient schedulability conditions (Adusley, 1991; Burchard et al., 1995; Han and Tyan, 1997; Kuo and Mok, 1997; Joseph and Pandya, 1986; Lehoczky et al., 1989). Since the present paper is not directly concerned with the bound itself, in the following we assume $\mathcal{U}_{\text{lub}}$ to be any chosen processor utilization least upper bound fully guaranteeing the set of real-time tasks.

The utilization factor can be written as:

$$\mathcal{U}(\mathcal{T}) = \mathcal{U}(\mathcal{T}_h) + \mathcal{U}(\mathcal{T}_s) \tag{4}$$

where $\mathcal{U}(\mathcal{T}_h) = \sum_{\tau_i \in \mathcal{T}_h} C_i/T_i$ is the utilization factor of the hard real-time subset and $\mathcal{U}(\mathcal{T}_s) = \sum_{\tau_i \in \mathcal{T}_s} C_i/T_i$ is the utilization factor of the soft real-time subset (Figure 1(a)).

The proposed adaptation policy, hereafter termed *rate adaptation policy* (RAP), is based on active management of the $\mathcal{U}(\mathcal{T}_s)$ utilization factor as a means to achieve an appropriate task schedule. In general, the need to perform such adaptation may arise for a number of reasons, mostly related to the dynamic nature of the environment and the partial modeling of the real-time system itself. We coarsely classify situations requiring adaptation as follows:

– less utilization factor is available for the set of tasks,

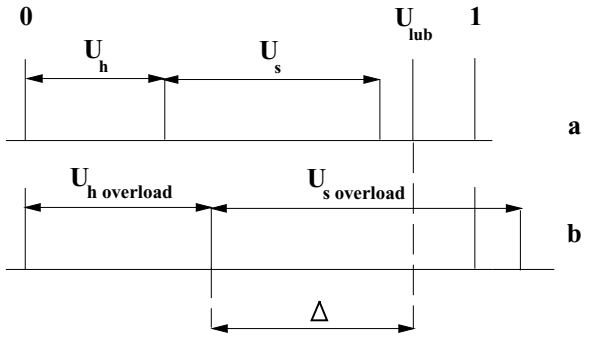

Figure 1. Requested CPU utilization before and after overload.

– more utilization factor is available for the set of tasks,

– need to adjust the utilization factors of the tasks as close as possible to their nominal values.

Overload conditions belong to the first class of situations. Overload conditions can arise because of hard or soft real-time tasks taking longer than expected, hard real-time sporadic or aperiodic tasks unaccounted for, and modification of scheduling parameters induced by the real-time application itself, e.g. to reserve CPU utilization to a critical task or to modify its rate. In case of overload, the active management of the $\mathcal{U}(\mathcal{T}_s)$ utilization factor is achieved by means of a reduction of the rate $f_i = 1/T_i$ for some or all the tasks $\tau_i \in \mathcal{T}_s$, leading to a schedule more suitable for the current context or implementing a graceful degradation of the application.

It should be noted that adaptation could also be triggered by the opposite situation; namely, additional utilization factor could become available to a set of soft real-time tasks, either because of decreased hard real-time load or because soft real-time tasks are deactivated in the current application context. In case of increased utilization factor becoming available, rate adaptation allows its dynamic redistribution to those tasks which can take advantage of higher execution rates.

Finally, if the nominal rates assigned by the designer bear special value for the application, an alternative goal of rate adaptation can be to keep task rates close to their nominal rates while minimizing overloads.

In the following, we phrase our discussion on the overload case, even though the same techniques apply when rate modulation is mandated by a control layer of the application or when more CPU utilization is available for the set of tasks. A separate discussion addresses the nominal rates case.

The following inequality:

$$\Delta = \mathcal{U}_{\text{lub}} - \mathcal{U}(\mathcal{T}_h) \geq 0 \tag{5}$$

holds even during overload, or otherwise some hard real-time deadline could be missed and the system would fail. Δ is the processor utilization actually available to soft real-time tasks.

When an overload occurs $\mathcal{U}(\mathcal{T})$ becomes greater than $\mathcal{U}_{\text{lub}}$, and some tasks miss their deadlines. Let Δ_{ov} be the available processor utilization for soft real-time tasks in an overload situation. Whatever the source of overload, since Eq. (5) must hold, we have:

$$\mathcal{U}(\mathcal{T}_s)_{\text{ov}} > \Delta_{\text{ov}} \tag{6}$$

Note that *both* members in (6) might have changed from their values prior to overload (Figure 1(b)). The system could be recovered if the adopted policy can reduce $\mathcal{U}(\mathcal{T}_s)$ below its present $\mathcal{U}(\mathcal{T}_s)_{\text{ov}}$ value while still executing soft real-time tasks at acceptable rates. The proposed adaptation policy exploits the degree of freedom offered by the $\mathcal{T}_s$ subset, which allows choosing in a range of possible rates for each task in order to re-establish

$$\mathcal{U}(\mathcal{T}_s) \leq \Delta_{\text{ov}} \tag{7}$$

This policy determines a controlled exit from the overload situation as long as a feasible rate assignment (compatible with constraints (2)) exists. However, the application level control system must be notified of the new set of rates, as the high-level control policy may need to be changed accordingly. Moreover, each interested task may possibly need to resynthesize any rate-dependant parameter, e.g. in PID controllers.

The system can be successfully adapted to the new operating conditions by rate modulation if

$$\sum_{\tau_i \in \mathcal{T}_s} \frac{C_i}{T_{i,\max}} = \mathcal{U}(\mathcal{T}_s)_{\min} \leq \Delta_{\text{ov}} \tag{8}$$

However, even when (8) does not hold, the rate modulation policy helps in taking the system to a possibly safer state, i.e., in general less tasks will be missing their deadlines after adaptation.

In order to simplify the presentation of the algorithms implementing the adaptation policy, we initially assume that tasks in $\mathcal{T}$ are sorted in such a way that $\forall \tau_i \in \mathcal{T}$, τ_i has a period T_i lower than the period T_{i+1} of τ_{i+1}, with $i = 1 \ldots N-1$. More precisely, because of the rate modulation range hypothesis in Equation (2), we assume that:

$$\begin{aligned} T_{i,\max} &\leq T_{i+1} \quad \text{or} \\ T_i &\leq T_{i+1,\min} \end{aligned} \tag{9}$$

Furthermore, we assume that $\forall \tau_i \in \mathcal{T}$, $T_i \leq T_{i,\max}$ in nominal operative conditions.

3.1. Basic Approach: Simple Rescaling

Let us define the *Rescaling Factor* $\mathcal{F}$ as $\mathcal{F} = \frac{\Delta_{ov}}{\mathcal{U}(\mathcal{T}_s)_{ov}}$ and its reciprocal *Relaxation Factor* η as $\eta = \frac{1}{\mathcal{F}}$. If $\forall \tau_i \in \mathcal{T}_s, \eta T_i \leq T_{i,\max}$ then a *fair* RAP is simply obtained by replacing T_i with $\eta\, T_i$.

This straightforward replacement will re-establish the real-time guarantees for the system as required by Eq. (7). Note however that if:

$$\exists \tau_i \in \mathcal{T}_s : \eta T_i > T_{i,\max} \tag{10}$$

a simple rescaling of the tasks' periods cannot be applied.

4. A General Rate Modulation Framework

In general, finding out a feasible modulation that guarantees the task set of a soft real-time application cannot be achieved by simple rescaling, due to the constraints on the allowable rate modulation ranges. We next describe a general problem formulation framework incorporating such constraints, along with several solution algorithms.

The goal is to determine a set of periods T_i for tasks $\tau_i \in \mathcal{T}_s$ so that the constraint (7) holds. Given the inverse relation between task periods and utilization factors, $\mathcal{U}_i = C_i/T_i$, the C_i can be assumed as constants.[1] We investigate approaches that search a guaranteed state along different directions in the space of the *Utilization Factors* $\mathcal{U} = \{\mathcal{U}_i : \mathcal{U}_{i,\min} \leq \mathcal{U}_i \leq \mathcal{U}_{i,\max}\}$, with $\mathcal{U}_{i,\min} = \frac{C_i}{T_{i,\max}}$ and $\mathcal{U}_{i,\max} = \frac{C_i}{T_{i,\min}}$.

Let Δ^* be a known value set to less than or equal to the maximum available utilization factor for soft real-time tasks, e.g., $\Delta^* = \Delta_{ov}$ in (7). Let $x_j = \mathcal{U}_j$, with $\tau_j \in \mathcal{T}_s, j = 1 \cdots S$ and $S = |\mathcal{T}_s|$. The x_j are real variables, Δ^* is a real constant, and $0 \leq x_j, \Delta^* \leq 1$.

We translate the rate modulation constraints (2) and the capacity constraint (7) in terms of the new variables x_j, leading to the following *linear programming* (LP) formulation:

Find

$$\max \sum_{j=1}^{S} v_j x_j \tag{11}$$

subject to

$$\begin{cases} x_{j,\min} \leq x_j \leq x_{j,\max}, & j = 1 \ldots S \\ \sum_{j=1}^{S} x_j \leq \Delta^* \end{cases}$$

where $x_{j,\min}$, $x_{j,\max}$ and v_j, $j = 1$ ... S, are known positive constants; weights v_j express the logical task value of $\tau_j \in \mathcal{T}_s$ (discussed below), $x_{j,\min} = \mathcal{U}_{j,\min}$, and $x_{j,\max} = \mathcal{U}_{j,\max}$.

As well known, linear programming problems can be solved with polynomial algorithms, such as the ellipsoid method (Garey and Johnson, 1973), or with the standard and usually efficient simplex algorithm (Hillier and Lieberman, 2000).

Weights v_j appearing in the LP formulation (11) can be used to alter the otherwise implicit assumption that the logical value of each task is essentially the same as its priority determined by the scheduling policy. As a matter of fact, in many application domains it is not uncommon to regard a low rate activity (hence lower priority in RM scheduling) as more valuable than higher rate computations (higher RM priority). Additionally, the scheduling priority and the logical value of a task can be considered as distinct parameters when scheduling guarantees have to be re-established after an overload.

4.1. Partially Ordered Rate Modulation Ranges

The rate modulation framework (11) yields a new schedule which does not affect task priorities. In fact, constraints in (9) ensure that all tasks maintain their original RM ordering after rescaling. However, in many real applications the disjoint rate modulation ranges constraints in (9) do not hold. In general, any two period ranges could be partially overlapping or properly included. Because of rate modulation, the priority of a subset of tasks, expressed by their RM ordering, could be modified from the original assignment.

The linear programming formulation (11) remains valid with general rate modulation ranges. Found solutions, though, might modify the original priority assigned to the task set, due to the non-uniform rate modulation.

In order to maintain the original priority assignment, if a feasible one exists, the following additional set of constraints (expressing the rate ordering preserving constraint (1)) can be included in the LP formulation in (11):

$$a_j\, x_j \geq a_{j+1}\, x_{j+1}, \quad j = 1 \ldots S-1 \tag{12}$$

where the a_j are known real constants, $a_j = 1/C_j, \;\; j = 1 \ldots S$.

Indeed, a major value of the LP formulation is that it provides a framework where several constraints can be coherently formulated based on the application needs. Unfortunately, some constraints move the problem out of the LP domain. Examples of such constraints are *minimizing the distance from nominal rates* and scheduling with *harmonic constraint*, which will be also discussed in the next section.

4.2. Discussion

While the LP problem (11), possibly supplemented by the additional constraints (12), is generally benign, its solution time with standard linear programming solvers like the simplex could prove too demanding for on-line adaptation if large task schedules must be dealt with. Furthermore, sometimes the optimal solution space is very large, with a region of solutions exhibiting the same value, as the constraints in (11) provide little bias to the solution.

These considerations motivate the investigation of adaptation policies which can take advantage from heuristic solution algorithms for the problem (11), both to obtain more efficient adaptation techniques and to bias the solution toward specific properties of interest. The policies described below are representative of the spectrum of techniques rather than exhaustive. They have been chosen because of the specific property they emphasize: low computational cost (greedy), fairness (iterative saturation), priority (prioritized saturation), protection of more valuable computations (value-based approaches), closeness to nominal scheduling parameters.

5. Adaptation Policies

5.1. Greedy Algorithm

Due to its simple structure, the particular LP instance (11) can be solved in linear time with a *greedy* approach.

RAP Algorithm 1: Greedy

Let $\mathcal{X}_{\min} = \sum_{j=1}^{S} x_{j,\min}$ and $\mathcal{X}_{\max} = \sum_{j=1}^{S} x_{j,\max}$. If $\mathcal{X}_{\max} \leq \Delta^*$ we set $x_j = x_{j,\max}$ for each $\tau_j \in \mathcal{T}_s$, i.e., each soft real-time task can be set to its maximum rate or to any desired rate compatible with the rate modulation constraints (2). If $\mathcal{X}_{\min} > \Delta^*$ than no feasible solution exists, and overload must be managed by a second-order graceful degradation mechanism. Nonetheless, all periods could be set to their maximum values to simplify overload management and reduce the number of tasks experiencing uncontrolled deadline misses.

The *greedy* algorithm re-establishes Eq. (7) by maximizing the utilization factor of a subset of tasks in $\mathcal{T}_s$ and relaxing the residual subset of tasks so that if after L steps, $L \leq$ S, we have:

$$\sum_{j=1}^{L} x_{j,\max} + \sum_{j=L+1}^{S} x_{j,\min} < \Delta^* \tag{13}$$

and after step L+1:

$$\sum_{j=1}^{L+1} x_{j,\max} + \sum_{j=L+2}^{S} x_{j,\min} > \Delta^* \tag{14}$$

then a feasible solution is given by the following choice:

$$\begin{aligned} x_j &= x_{j,\max}, \quad j = 1 \ldots L \\ x_j &= x_{j,\min}, \quad j = L+2 \ldots S \\ x_{L+1} &= \Delta^* - \left(\sum_{j=1}^{L} x_{j,\max} + \sum_{j=L+2}^{S} x_{j,\min} \right) \end{aligned} \tag{15}$$

While simple, the greedy approach does not yield a solution ideally suited for most applications, as it is strongly biased toward maximizing x_j for higher priority (lower indexes) tasks at the expense of lower priority ones. In this respect, the greedy algorithm may be considered as the opposite of the rescaling algorithm in a fairness spectrum. Furthermore, the reference period for soft real-time tasks does not necessarily correspond to the minimum one. When nominal rates are available, an improved greedy behavior is obtained by replacing maximum utilization factors $x_{j,\max}$ with nominal ones $x_{j,n}$ in Eqs. (13) to (15). With this modification, favoured tasks will run at most at their nominal rate.

5.2. Saturation Approaches

It should be noted that the simple rescaling RAP provides, where it is applicable, a maximal fairness approach in constant time. Several algorithms can be conceived to overcome the limitations of simple rescaling, providing an approximation of the ideal rescaling solution when it is not applicable.

We describe two simple algorithms differing in the way they choose the tasks that will be set to their maximum allowed period and hence only partially rescaled (*saturated* tasks). Both algorithms have quadratic complexity in the number of soft real-time tasks, but in the general case they obtain a more balanced allocation of processor utilization than the greedy algorithm.

RAP Algorithm 2: Iterative Saturation

Let $\mathcal{T}_{s,\text{sat}} \subset \mathcal{T}_s$ be the set of tasks $\tau_j : \eta\, T_j > T_{j,\max}$ in (10). These tasks are saturated, i.e.:

$$\forall \tau_k \in \mathcal{T}_{s,\text{sat}}\ T_k = T_{k,\max} \tag{16}$$

$\mathcal{T}_s$ is then partitioned in such a way that:

$$\mathcal{T}_s = \mathcal{T}_{s,\text{unsat}} \bigcup \mathcal{T}_{s,\text{sat}} \tag{17}$$

On the current set of unsaturated tasks $\mathcal{T}_{s,\text{unsat}}$ we attempt again the rescaling $\eta\, T$ mechanism, with:

$$\Delta = \Delta^* - \mathcal{U}(\mathcal{T}_{s,\text{sat}}) \tag{18}$$

and $\mathcal{F}$, η modified accordingly. Since additional tasks may need saturation, the algorithm continues until the condition $\mathcal{U}(\mathcal{T}_s) \leq \Delta^*$ is re-established.

RAP Algorithm 3: Prioritized Saturation

This algorithm obtains a set of acceptable rate modulation values by saturating tasks in order of increasing priority. Thus, at each iteration K, $K \leq$ S:

$$T_j = T_{j,\max}, \quad j = S \ldots K \tag{19}$$

$\mathcal{T}_s$ is partitioned so that:

$$\begin{aligned} \mathcal{T}_s &= \mathcal{T}_s^k \bigcup \mathcal{T}_{s,\text{sat}} \quad \text{where} \\ \mathcal{T}_s^k &= \{\tau_1, \ldots, \tau_{K-1}\} \quad \text{and} \\ \mathcal{T}_{s,\text{sat}} &= \{\tau_K, \ldots, \tau_S\} \end{aligned}$$

Tasks in $\mathcal{T}_s^k$ are modulated, if possible, by replacing T_j with $\eta\, T_j$, where:

$$\Delta = \Delta^* - \mathcal{U}(\mathcal{T}_{s,\text{sat}}) \tag{20}$$

and $\mathcal{F}$, η are modified accordingly. Iterations start with $K =$ S and continue with $K =$ S $-$ 1, S $-$ 2, ... until condition $\mathcal{U}(\mathcal{T}_s) \leq \Delta^*$ is re-established.

Both saturation approaches described in this section can also be applied when rate modulation ranges are only partially ordered. In this case, however, a task re-ordering operation must be executed at each iteration when RM consistency is not preserved owing to modulation.

5.3. Value-Based Heuristics

Greedy and *prioritized saturation* algorithms can be easily modified to take into account logical task values rather than RM priorities to establish which tasks will be saturated. The greedy algorithm partitions $\mathcal{T}_s$ so that in the first subset $x_j = x_{j,\max}$ with j low index (i.e. τ_j high RM priority task), in the second subset $x_j = x_{j,\min}$ with j high index (i.e. τ_j low RM priority task), whereas in the third subset a single task with intermediate index (i.e. intermediate RM priority) is adjusted according to (15). A different "greedy" adaptation can be obtained simply by changing the task order in $\mathcal{T}_s$. Therefore, a greedy value-based algorithm can be defined by a new task order in $\mathcal{T}_s$, where $\forall \tau_i, \tau_j \in \mathcal{T}_s$ $i \leq j$ entails that $v_i \geq v_j$. In fact, high value tasks within this new partition are prioritized (i.e. run at their maximum or nominal rate) during overload regardless of their RM priority.

As the prioritized saturation algorithm also considers the high index (low RM priority) tasks to be of lower importance, a different, value-based task arrangement in $\mathcal{T}_s$ affects the final modulation as well. Hence, a value-based re-ordering of $\mathcal{T}_s$ leads to a different rate modulation that prioritizes high logical value tasks.

5.4. *Scheduling with Nominal Rates*

Previously described RAP algorithms exploit task values (and priorities) to re-establish a suitable scheduling under overload conditions, given a set of timing constraints. However, the new parameters might noticeably modify the nominal, guaranteed scheduling in use before the overload. Nominal task rates have been typically assigned at design time in order to endow the application with the desired behavior while guaranteeing soft real-time constraints. For this reason, nominal rates might also be taken into account while re-establishing the scheduling after the overload.

It should be noted that any adaptation policy can be supplemented with a constant time test to restore nominal rates if sufficient utilization factor is available for soft real-time tasks, i.e., $\sum_{\tau_j \in \mathcal{T}_s} C_j/T_{j,n} \leq \Delta = \mathcal{U}_{\text{lub}} - \mathcal{U}(\mathcal{T}_h)$. The following algorithm addresses the problem of re-establishing a guaranteed schedule while maintaining task rates as close as possible to a given set of nominal values, where closeness is related to a distance function.

RAP Algorithm 4: Minimum Distance

Let $x_j = \mathcal{U}_j$ be the utilization factor and $x_{j,n} = \mathcal{U}_{j,n} = \frac{C_j}{T_{j,n}}$ be the nominal utilization factor $\forall \tau_j \in \mathcal{T}_s$, with $j = 1$ ...S and $S = |\mathcal{T}_s|$. Let $f(x_j) = v_j\,(x_j - x_{j,n})^2$ be a set of convex positive functions. The goal of maintaining rates close to the nominal ones leads to the following quadratic programming (QP) problem:

Find

$$\min \sum_{j=1}^{S} v_j\,(x_j - x_{j,n})^2 \tag{21}$$

subject to

$$\begin{cases} x_{j,\min} \leq x_j \leq x_{j,\max}, & j = 1 \ldots S \\ \sum_{j=1}^{S} x_j \leq \Delta^* \end{cases}$$

where $x_{j,\min}$, $x_{j,\max}$, and v_i, $j = 1$...S, are known constants, with $x_{j,\min} = \mathcal{U}_{j,\min}$, $x_{j,\max} = \mathcal{U}_{j,\max}$, and v_j is the logical task value.

Being the quadratic objective function positive definite, (21) is a convex programming problem, which is well known to be in P (the class of problems solvable in polynomial time). Hence it can be efficiently solved by a number of algorithms for convex quadratic programming (Floudas and Visweswaran, 1995), including simple extensions of LP solution algorithms (Hillier and Lieberman, 2000; Horst and Pardalos, 1995).

5.5. *Scheduling with Complex Constraints*

We next discuss an example of a scheduling constraint which could be useful in several applications but unfortunately moves the problem out of the LP or QP domains.

Adding this constraint to the basic framework in section 4 implies that a more complex solution algorithm is required, thus possibly preventing on-line adaptation for large schedules.

RAP Algorithm 5: Minimal Harmonic Base

The *minimal harmonic base* constraint specifies that the new rate values computed by the adaptation algorithm should be harmonic, or at least as harmonic as possible. This constraint increases the guaranteed utilization factor $\mathcal{U}_{\text{lub}}$ for RM scheduling from $L(N)$ up to $L(H) \geq L(N)$, where $H \leq N$ is the cardinality of the *harmonic base* (Kuo and Mok, 1997). Moreover, a reduced harmonic base could be sought in order to simplify the actual dispatching of the real-time tasks. Taking advantage from the definitions in equations (2) and (3), the problem of determining a minimal harmonic base solution for N periodic *real-time* tasks can be described as follows:

Find

$$\max \sum_{j=1}^{N} v_j x_j \tag{22}$$

subject to

$$\begin{cases} x_{j,\min} \leq x_j \leq x_{j,\max}, & j = 1 \ldots N \\ \sum_{j=1}^{N} x_j \leq \Delta_H & \\ x_j = C_j \, k_{j,h} \, f_h, & j = 1 \ldots N, \quad h = 1 \ldots H < N \end{cases}$$

where $x_{j,\min}, x_{j,\max}$ and $v_j, j = 1 \ldots N$, are known positive constants, $x_{j,\min} = C_j / T_{j,\max}$, $x_{j,\max} = C_j / T_{j,\min}$, and weights v_j express the logical task value $\forall \tau_i \in \mathcal{T}$. In problem (22) $k_{j,h}, f_h$ and H are search variables: $k_{j,h}$ are integers, f_h are the H linear independent frequencies of the harmonic base, and H is an integer. Note that the second constraint in (22) depends on H, namely $\Delta_H = \mathcal{U}_{\text{lub}} = H(2^{\frac{1}{H}} - 1)$.

This problem is in NP (Kuo and Mok, 1997) and not amenable to on-line solution. An approach for off-line computation involves integration of *Constraints Satisfaction Programming* (Marriot and Stuckey, 1998) and LP techniques via an exhaustive search that minimizes H. Of the various adaptation strategies considered in this paper, pursuing a minimal harmonic base is the only one not suitable for an on-line implementation.

5.6. *Comparison with Elastic Scheduling*

The rate adaptation heuristics presented in this section might be considered, at a first glance, similar to the "elastic"-based adaptation proposed in Buttazzo et al. (1998). We next highlight the main difference between the two approaches considering their basic mechanisms.

Suppose that adaptation of soft real-time tasks is required to make room for an extra utilization demand of hard real-time tasks. The overall utilization factor available to soft real-time tasks is thus decreased from Δ_s to Δ_s'. Under the elastic adaptation in Buttazzo et al. (1998), if all soft real-time tasks have identical elastic coefficient, the utilization ($\Delta_s - \Delta_s'$) is subtracted in equal parts to all soft real-time tasks, regardless of their current utilization. For low-utilization tasks, the amount to be subtracted could be comparable to, or even larger than, their current utilization, whereas high-utilization tasks would be minimally affected by adaptation. Thus, if all tasks have the same elasticity, elastic adaptation can easily lead to the setting of low-utilization tasks at their minimum rate, yielding a behavior close to the Greedy algorithm previously discussed. In contrast, under simple rescaling (the basic adaptation mechanism proposed in this paper) the utilization ($\Delta_s - \Delta_s'$) is subtracted to soft real-time tasks in proportion to their requested nominal utilization. Thus, this approach will lead to a much more balanced distribution of available utilization factor.

Indeed, in the elastic approach higher elasticity can be assigned to those tasks that are deemed more suitable for adaptation. The extra utilization factor would then be subtracted to soft real-time tasks in proportion to their elastic coefficient. Different elastic coefficients could thus be exploited to alleviate the problem of imbalanced adaptation. However, the proper setting of these coefficients remains difficult, given that elasticity has no relation with the processor utilization requested by soft real-time tasks. In the set of algorithms proposed in this paper, the higher suitability for adaptation of certain tasks can be specified by assigning them lower values. The value concept is supported both in the general LP formulation and in the Greedy, Prioritized saturation, and Minimum distance RAP algorithms. The LP formulation will provide a set of rates compatible with available utilization that maximizes the overall value of the computation, whereas the Minimum distance RAP will provide a set of rates as close as possible to the nominal rates while maximizing the overall value of the computation.

6. Transition

When adaptation is triggered by a higher level control system, an issue to be considered with any rate adaptation policy is how to avoid transient overloads. In general, if the rate of some tasks must be increased, transition to the new set of rates should be carefully managed in order not to exceed at any instant of time the available utilization factor. On the other hand, when adaptation is required to recover from an overload condition, soft real-time tasks are already experiencing deadline misses, and the prominent goal is to bring the system under control.

Let us assume that the task set is guaranteed both before and after adaptation. Let $\{T_i'\}$ be the new set of periods computed by the specific rate adaptation policy. We partition the task set $\mathcal{T}_s$ in such a way that $\mathcal{T}_{s,\text{incr}}$ is the set of tasks whose period will either increase or remain unchanged ($T_i' \geq T_i$), and $\mathcal{T}_{s,\text{decr}}$ is the set of tasks whose period will decrease ($T_i' < T_i$). Let t^* be the instant of time in which adaptation to the new set of rates is requested. Under the hypotheses of this paper, tasks in $\mathcal{T}_{s,\text{incr}}$ can be immediately adapted, thereby reducing, in general, the future utilization demand. Hence, in t^* the current execution deadline of the tasks whose period must be increased is delayed according to the new period.

We thus focus our attention on tasks in $\mathcal{T}_{s,\text{decr}}$, whose rate must be increased. Let $\tau_i \in \mathcal{T}_{s,\text{decr}}$ be the first of such tasks and $\beta_i = \lceil (t^* - a_i)/T_i \rceil \; T_i$, $\beta_i \geq t^*$, its next execution deadline, where a_i is the activation time of τ_i. In β_i, all tasks in $\mathcal{T}_{s,\text{incr}}$ have already been adapted. Since by hypothesis the initial and final schedules are both guaranteed, in β_i the period of τ_i can be safely set to its new value T_i'. This argument can be repeated for all tasks in $\mathcal{T}_{s,\text{decr}}$. Liu and Layland's theorem (Liu and Layland, 1973) applied at the next execution deadline β_i of each $\tau_i \in \mathcal{T}_{s,\text{decr}}$ ensures that the utilization factor remains below $\mathcal{U}_{\text{lub}}$. A similar argument applies to any schedulability check based on a least upper bound of utilization factor.

In summary, rate adaptation of rate-increasing tasks can be safely performed at their next release instant following t^*. This strategy guarantees that transient overloads are not induced by adaptation and bounds transition time. In the worst case, the delay between the time at which adaptation is requested and the end of the transition is equal to the longest period T_i of soft real-time tasks. A similar result was obtained in Buttazzo et al. (1998) for elastic-based adaptation in connection with EDF task scheduling. Note that the proposed transition strategy is based on a sufficient-only schedulability condition. The problem of designing an algorithm to minimize transition time in each adaptation instance remains open.

7. Experimental Results

A set of experiments has been carried out to assess the behavior of the proposed algorithms onto a test application when its execution is affected by an artificial overload. The experimental validation has involved both off-line simulations and actual execution of multi-threaded applications on a Solaris workstation. This section presents the results we have obtained by modulating the nominal rate of application tasks scheduled with RM scheduling. We first describe the off-line simulation of a simple test case and compare the resulting scheduling adaptations. Next, we summarize the results of a statistical investigation involving the generation of 1,000 test cases. Finally, we report the results obtained with Solaris thread-based applications, which illustrate the effect of rate modulation algorithms in actual real-time systems.

7.1. Off-Line Assessment on a Simple Test Case

Table 1 shows the parameters for a simple off-line assessment of the rate modulation algorithms. All tasks in $\mathcal{T}_s = \{\tau_1 \ldots \tau_6\}$ are considered soft real-time tasks owing to the fact that their periods (utilization factors) can be chosen within admissible ranges. Nominal scheduling parameters (T_n and U_n) are also included in the table. Ranges are neither interleaved nor nested, hence the total ordering assumption holds. Rate monotonic analysis shows that tasks are schedulable as $\mathcal{U}_{\text{tot}} = 0.7705 < M(3) = 0.7797$ (where $M(K)$ is Kuo and Mok's upper bound (Kuo and Mok, 1997) for a size K of the harmonic base). Task indexes in Table 1 are tied to their priority assignment.

Table 1. Timing parameters for a set of soft real-time tasks (all times in ms).

Tasks	C	$T_{\min}$	T_n	$T_{\max}$	$U_{\max}$	U_n	$U_{\min}$	Priority	Value
τ_1	3	20	30	40	0.15	0.1	0.075	6	6
τ_2	4	40	60	80	0.1	0.0667	0.05	5	1
τ_3	20	80	120	180	0.25	0.1667	0.1111	4	2
τ_4	44	180	270	300	0.2444	0.1630	0.1467	3	5
τ_5	60	320	540	600	0.1875	0.1111	0.1	2	3
τ_6	150	600	920	1200	0.25	0.163	0.1250	1	4
U_{tot}					1.1819	0.7705	0.6078		

The simulation assumes that at a certain time (t_{init}) task τ_1 needs to be scheduled with a shorter period, $T_1 = 20$ ms, thus increasing the overall requested utilization factor to $\mathcal{U}_{\text{tot}} = \sum_{i=1}^{6} \mathcal{U}_i = 0.8204$, and possibly causing a timing fault. As a matter of fact, the increase of the utilization factor of τ_1 could equally be induced by an unexpected elongation of its execution time.

The real-time analysis shows that tasks in $\mathcal{T}' = \{\tau_1 \ldots \tau_5\}$ are still guaranteed, as $\sum_{i=1}^{5} \mathcal{U}_i = 0.6574 \leq L(5) = 0.7435 \leq M(3) = 0.7797$, whereas $\mathcal{T}' \cup \{\tau_6\}$ is no longer guaranteed by RM scheduling and τ_6 could miss its deadlines ($\sum_{i=1}^{6} \mathcal{U}_i = 0.8204 > M(3)$).

Tables 2 and 3 show how the rate modulation algorithms described in the paper alter task rate parameters to re-establish scheduling guarantees. The column labels P and V refer to the heuristics based on task priorities and task values respectively. Within this simulation τ_1, which is now required to be scheduled with a fixed period of $T_1 = 20$ ms, is regarded as a "hard" real-time task, i.e., not subject to the rate adaptation strategy, and it is hence labeled as *hard* in Tables 2 and 3.

The *simple rescaling* algorithm is not applicable for this problem owing to the constraints on task periods. Given the underconstrained nature of this problem, the results provided by the general *linear programming* formulation are not particularly useful and are omitted from Tables 2 and 3.

Table 2. Adaptations resulting from an off-line assessment of some RAP algorithms for the set of tasks in Table 1 (periods in ms).

	Greedy P		IterSat		PrioSat P		MinDist	
Tasks	T	policy	T	policy	T	policy	T	policy
τ_1	20	*hard*	20	*hard*	20	*hard*	20	*hard*
τ_2	40	*min*	70	*adapt*	66	*adapt*	65	*adapt*
τ_3	176	*adapt*	140	*adapt*	132	*adapt*	148	*adapt*
τ_4	300	*max*	300	*max*	297	*adapt*	286	*adapt*
τ_5	600	*max*	600	*max*	600	*max*	600	*max*
τ_6	1200	*max*	1078	*adapt*	1200	*max*	1107	*adapt*
$\rho^2{}_x$		0.0058		0.0016		0.0021		0.0015

Table 3. Adaptations resulting from an off-line assessment of value-based RAP algorithms for the set of tasks in Table 2 (periods in ms).

	Greedy V		PrioSat V		MinDist V	
Tasks	T	policy	T	policy	T	policy
τ_1	20	*hard*	20	*hard*	20	*hard*
τ_2	80	*max*	80	*max*	80	*max*
τ_3	180	*max*	180	*max*	147	*adapt*
τ_4	221	*adapt*	277	*adapt*	292	*adapt*
τ_5	600	*max*	564	*adapt*	600	*max*
τ_6	1200	*max*	944	*adapt*	1014	*adapt*
${\rho^2}_x$		0.0062		0.0034		0.0017

The solution obtained by the *greedy* algorithm (Table 2) is easily computed, although it exhibits lack of fairness. The task set is partitioned in such a way that task τ_2 is executed at its maximum rate, i.e. with its minimum period (*min*), task τ_3 is executed at an intermediate rate (*adapt*), whereas all other tasks are maximally relaxed, i.e., their periods are set to the maximum allowable values (*max*). The *iterative saturation* algorithm results into a more balanced solution, since it is able to adapt as many task periods as possible while saturating a reduced set to their maximum period. The *prioritized saturation* algorithm clearly favors high priority tasks so that it saturates low priority ones. Columns *Greedy V*, *PrioSat V* and *MinDist V* in Table 3 show how the the various RAPs implemented are significantly affected by value parameters. The *minimum distance* algorithm obtains the scheduling parameters closest to the nominal rates, as indicated in Table 2 by ${\rho^2}_x$, i.e. the quadratic residual for the utilization factors with respect to the nominal rates.

7.2. Statistical Assessment

A statistical assessment of the rate modulation algorithms has been performed by means of a simulation-based methodology. One thousand rate modulation cases have been simulated for randomly generated task sets of $N = 20$ tasks each. In each experiment, all tasks have the same computation time C_i and the same admissible range for their utilization factor $\mathrm{U_i}$, i.e. $U_i \in [U_{\min}, U_{\max}]$. The initial value for U_i is sampled from a uniform distribution between $U_{\min}$ and $U_{\max}$. This value is assumed as the desired, nominal utilization factor $U_{i,n}$ for each task.

Minimum and maximum values for utilization factors are chosen to be $U_{\min} = L\,(N)/5\,N$ and $U_{\max} = 4\,L(N)/N$. Since $N = 20$, we obtain $L(20) = 0.7053$, $U_{\min} = 0.00705$, and $U_{\max} = 0.14105$. With this set of values, it is quite likely that the sum of utilization factors exceeds $L(N)$ and the set of tasks cannot be scheduled at the requested nominal rates. Rates are then adapted based on the RAP algorithms previously described.

Figures 2 and 3 depict the results of rate adaptation performed by the algorithms to obtain an overall utilization factor equal to $L(20)$. To enable a comparison of the various policies,

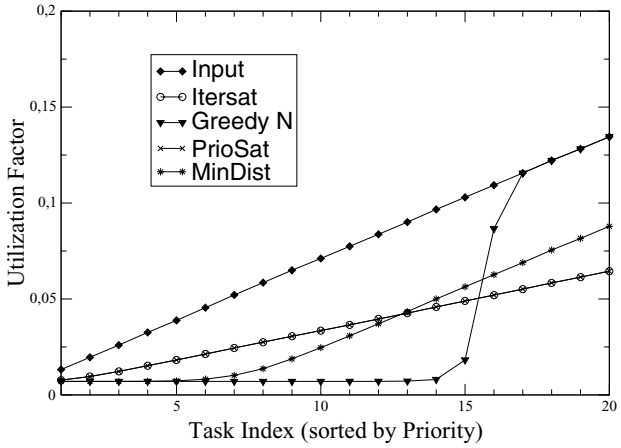

Figure 2. Statistical assessment of the rate modulation algorithms: task indexes sorted by priority.

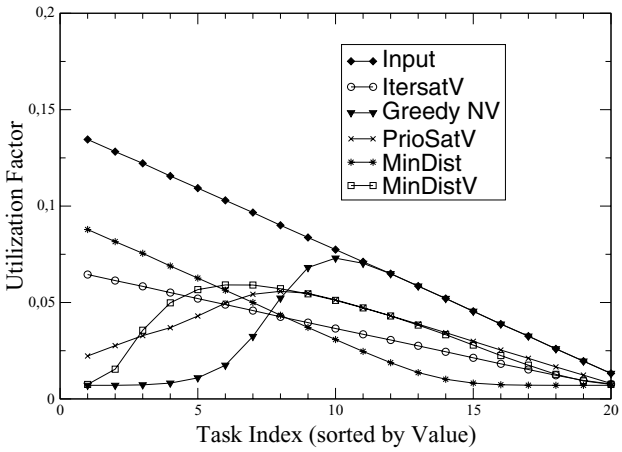

Figure 3. Statistical assessment of the rate modulation algorithms: task indexes sorted by value.

in Figure 2 tasks are sorted by increasing *priority*, i.e. the task index increases with the requested nominal rate. In Figure 3, instead, tasks are sorted by increasing *value*, i.e. the task index increases with the task value, which in turn is assumed to run in the reverse order with respect to requested nominal rates. Values are then computed as $V_i = 10 \cdot i$. In both figures the average values of the requested and granted utilization factors across the 1000 random sets of tasks are reported.

The curve labeled as *Input* represents the requested, nominal utilization factor for each task. The remaining curves represent the outcome of the various RAPs. *Iterative saturation* and *prioritized saturation* exhibit identical average behavior in Figure 2, whereas *prioritized saturation with value*, labeled as PrioSat V, markedly favors higher value tasks in Figure 3.

Greedy is implemented with reference to nominal rates in both Figures 2 and 3, and hence labeled as Greedy N or Greedy NV when different task values are considered: in each task set, a few tasks are set to their nominal rate, one at an intermediate rate, and the remaining at their minimum rate. *Minimum distance* provides the adaptation closer to the requested *Input* utilization factors considering all tasks. When different task values are provided for the task set, the resulting adaptation (labeled MinDist V) is significantly affected, as shown in Figure 3.

From a general viewpoint, the *minimum distance* and the *iterative saturation* algorithms appear more suitable for a generic soft real-time application, owing to their ability to maintain task rates closer to their nominal values. The *prioritized saturation* algorithm may prove more effective in specific applications, where rate assignments closely follow the relative importance even of soft real-time tasks.

7.3. Experimental Evaluation

In the set of experiments described hereafter, rate modulation has been applied to a multi-threaded application running on a Solaris workstation. Experiments leverage upon Solaris multi-threading capabilities and soft real-time oriented features, including reduced latency to interrupts, frequent kernel preemption points, separate scheduling classes (*Real-Time, Sys, User*) for thread execution, process memory locking to avoid paging, and high resolution real-time timers (Sun, 2000).

Building on the POSIX facilities of Solaris, we have also developed a custom library aiding the design and development of real-time control architectures for the robot domain. This library, termed `rt-lib` (Beccari et al., 1998), implements mechanisms for periodic computations, handling of timing faults, monitoring of CPU utilization, and procedures that allow on-line modification of thread scheduling parameters (period, deadline and priority). Times in `rt-lib` are measured using `gethrtime/gethrvtime` calls (with 1 μs resolution on Solaris 8 for SPARC), whereas periodic computations exploit the `nanosleep` system call (1 ms resolution).

Table 4 shows the *soft real-time* parameters of the thread-based application, developed with `rt-lib` under Solaris. In this test application, a specific thread *rt_monitor* (acting

Table 4. Timing parameters for the set of soft real-time tasks under Solaris (all times in ms).

Tasks	C	$T_{\min}$	T_n	$T_{\max}$	$U_{\max}$	U_n	$U_{\min}$	Priority	Value
rt_mon	2.9097	10	30	40	0.2910	0.097	0.0727	6	6
τ_1	4.2864	40	60	80	0.1072	0.0714	0.0536	5	1
τ_2	17.1343	80	120	160	0.2142	0.1428	0.1071	4	2
τ_3	45.825	200	250	500	0.2291	0.1833	0.0917	3	5
τ_4	61.4847	500	500	700	0.123	0.123	0.0878	2	4
τ_5	153.8896	700	750	2000	0.2198	0.2052	0.0769	1	3
U_{tot}					1.1843	0.8227	0.4898		

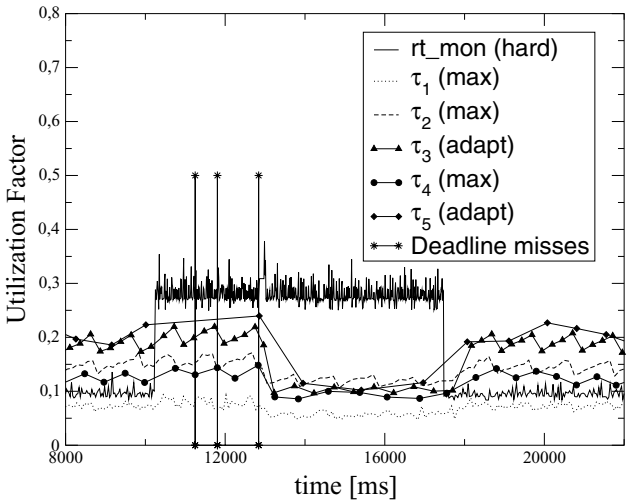

Figure 4. Adaptation of utilization factors by means of Iterative Saturation. After three deadline misses (at around 12000 ms) rates are modulated to bring the system to a safe condition. When the overload ends, nominal periods are restored for all tasks.

as a system manager) traces the application utilization factor (hereafter $\mathcal{U}_{\text{app}}$), while other threads $\{\tau_1 \ldots \tau_5\}$ execute dummy workloads.

Threads are activated in nominal operating conditions with RM priority assignment, yielding a harmonic base cardinality $K = 3$. Although the system utilization factor $U_{\text{tot}} = 0.8227$ is greater than L(6) and M(3), the real-time analysis proves the schedulabilty of these tasks. Again, rate ranges are totally ordered.

In this experiment overload occurs roughly at $t_{\text{init}} = 10.8$ s (Figure 4) on account of the increased activation frequency of the *rt_monitor* thread, which the application level requires now to be executed with a shorter period $T_1 = 10$ ms. Overload ends roughly at $t_{\text{end}} = 17.8$ s, when the application level decreases the *rt_monitor* activation period to its original value $T_1 = 30$ ms. During the overload the system is not guaranteed as the desidered utilization factor becomes $\mathcal{U}_{\text{app}} = 1.0167$. We observe that while the thread subset $\mathcal{T} = \{rt\_monitor\} \cup \{\tau_1 \ldots \tau_4\}$ is schedulable (the frequency set is 2-harmonic and $\mathcal{U}(\mathcal{T}) = 0.8115 < M(2)$), the whole application is not, and τ_5 execution cannot be guaranteed, yielding multiple deadline misses. In the same experiment run under plain Solaris and without any adaptation, the utilization factor for τ_5 is drastically reduced (yielding a stream of deadline misses for this thread), while all other threads remain unaffected.

Table 5 shows the parameters computed by some of the rate modulation algorithms described in Section 5 that allow scheduling guarantees to be re-established. Thread *rt_monitor* is assumed to be a *hard* real-time thread whose period cannot be modified. To cope with the limited computational capacity available, the various RAPs cause a few soft realt-time threads to be rescaled *(adapt)*, whereas the remaining ones are set to their maximum period

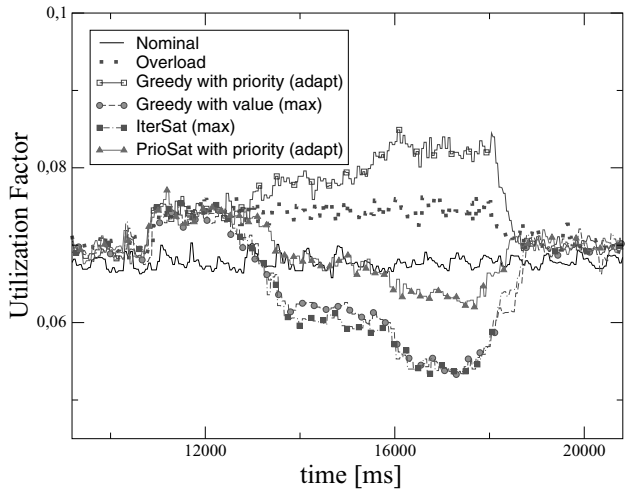

Figure 5. Measured utilization factor of thread 1 under various rate adaptation policies.

(max). For the specific problem, *priority saturation with value* finds the same modulation parameters obtained by *iterative saturation*.

In these simple experiments (Figures 4 to 7) there is no run-time task acceptance nor dynamic computation of the adaptation. Rather, the rate modulation algorithms have been computed off-line, and the adapted timing parameters have been directly coded into the thread-based application and properly handled by `rt-lib` facilities following the on-line overload detection, namely the third deadline miss occurrence. All rate adaptation policies change $\mathcal{U}_{\text{app}}$ in a way that it becomes less or equal to $L(6) = 0.7348$. When the cause of overload ceases, a straightforward policy has been adopted in these tests to re-establish nominal operating conditions: namely, when the *rt_monitor* thread detects $\mathcal{U}_{\text{app}} < 0.6$ for more than 10 cycles, it resets all soft real-time thread periods to their prior nominal values.

Figure 4 reports the actual traces of the utilization factors measured in an experiment of overload management through *iterative saturation*. After the third deadline miss occurrence, thread *rt_monitor* enforces the new set of rates ($t_{\text{mod}} \approx 12$ s), bringing the system to a safe state. When the overload ends, the nominal scheduling parameters of Table 4 are estabilished by thread *rt_monitor*, and therefore utilization factors are restored accordingly ($t_{\text{end}} \approx 18$ s). Figures 5 and 6 depict how the different rate adaption algorithms handle threads τ_1 and τ_5. Measured utilization factors are consistent with results in Table 5.

An additional perspective on these experiments is given by Figure 7. Cumulative utilization factors measured when no overload affects the system (first part of Figure 7) faithfully follow the expected, nominal behavior. When an overload occurs (at time 10.8 s in Figure 7), the requested overall utilization factor clearly exceeds the RM bound and does so until rate adaptation according to *iterative saturation* is applied.

We conclude this section by presenting the result of an *on-line* adaptation experiment (Figure 8). This particular experiment has been executed with a modified library which also performs on-line acceptance. Parameters for the set of tasks are those given in Table 4.

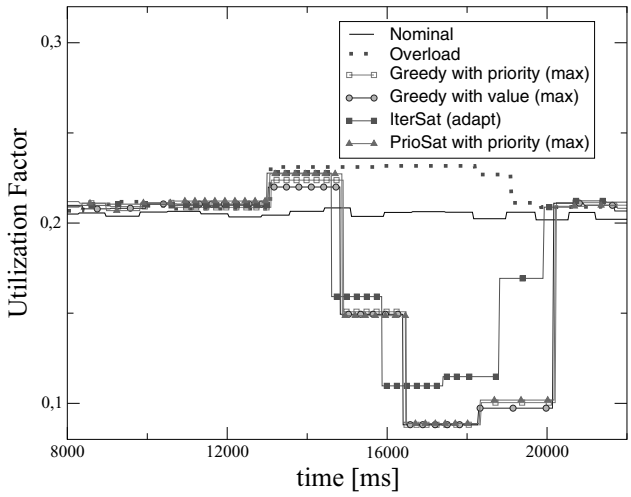

Figure 6. Measured utilization factor of thread 5 under various rate adaptation policies.

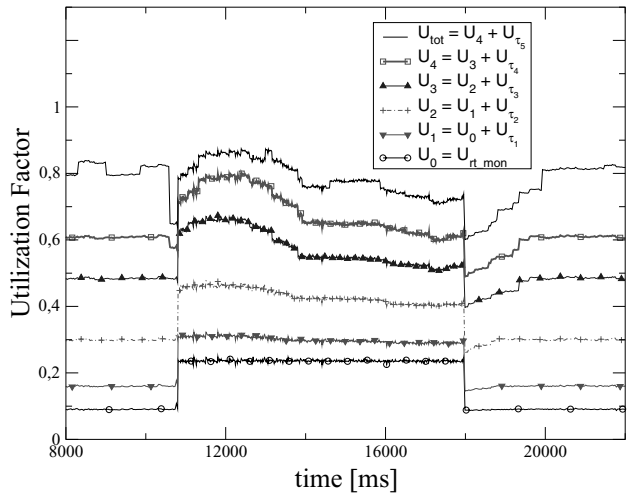

Figure 7. Measured cumulative utilization factors with scheduling parameters modulated by Iterative Saturation.

The rate of task *rt_monitor* is increased at time 10 s by request of the high-level control system. The remaining tasks rates then are computed on-line by running *iterative saturation*. When the high-level control system demands that *rt_monitor* be scheduled at its prior rate, all remaining tasks are restored to their nominal rates. Since it is directly triggered by the on-line acceptance mechanism, adaptation in Figure 8 proves quite effective.

As mentioned earlier, all adaptation heuristics, except pursuing a minimal harmonic base, have manageable complexity. In practice, on a standard Pentium 4 2.4 GHz CPU, *greedy,*

Table 5. Rate modulation evaluation: adaptations computed for the set of tasks in Table 4 (periods in ms).

	Greedy P		Greedy V		IterSat		PrioSat P		PrioSat V	
Tasks	T	Policy	T	Policy	T	Policy	T	Policy	T	Policy
rt_mon	10	*hard*	10	*hard*	10	*hard*	10	*hard*	10	*hard*
τ_1	53	*adapt*	80	*max*	80	*max*	68	*adapt*	80	*max*
τ_2	160	*max*	160	*max*	160	*max*	137	*adapt*	160	*max*
τ_3	500	*max*	387	*adapt*	497	*adapt*	500	*max*	497	*adapt*
τ_4	700	*max*	700	*max*	700	*max*	700	*max*	700	*max*
τ_5	2000	*max*	2000	*max*	1491	*adapt*	2000	*max*	1491	*adapt*

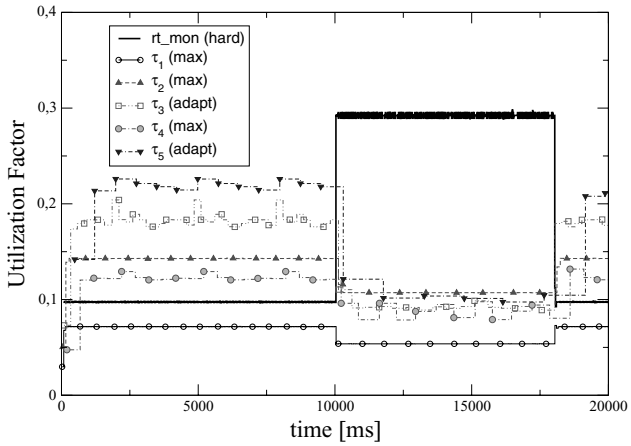

Figure 8. Measured utilization factor per thread in an on-line adaptation experiment based on Iterative Saturation.

iterative saturation, priority saturation, and *minimum distance* rate adaptation algorithms have been measured at less than 1 ms for on-line adaptation of 20 task sets. In the 6 task *iterative saturation* experiment in Figure 8, computation of the new set of rates requires less than 100 μs.

8. Conclusions

Several real-time applications demand overload management and graceful degradation mechanisms. In this paper, we have presented a rate adaptation scheduling framework for soft real-time periodic tasks characterized by a range of admissible rates. The framework is based on a linear programming problem formulation which can be integrated by other pertinent application requirements.

Several solution heuristics have been discussed in the paper. Most of the heuristics and problems variations are suitable for on-line application to task schedulers.

When the rate of some task is required to change by the high-level control system, the remaining soft real-time tasks can be adapted based upon one of the proposed heuristics. In other situations, adaptation must be triggered by events in the real-time systems itself, such as deadline misses and on-line workload measurement. In the experiments reported in this paper, we have considered rather simplistic criteria, namely when three or more consecutive deadline misses are detected within a given time window or when the measured utilization factor falls below a fixed threshold and a more valuable rate assignment exists. These criteria are motivated by the following considerations:

- sporadic deadline misses can occur in many real-time systems, yet an isolated miss should not trigger by itself a full adaptation round;
- rate adaptation should be an unfrequent event, due to its associated overhead and to the need to avoid jeopardizing applications; hence, some hysteresis in adaptation thresholds could be useful.

On the other hand, relying on the deadline misses of the lowest priority thread is an inherently slow mechanism. Faster and less *ad hoc* triggering mechanisms are required, an issue thus deserving further research.

Another issue discussed in the paper is transition to the new set of rates once they have been computed. In the experiment shown in Figure 8, on-line adaptation is particularily fast and transition is seamless. In general, if the rate of some tasks must be increased, transition to the new set of rates should be carefully managed in order not to exceed at any instant of time the available utilization factor. Adapting rate-increasing tasks at their next release instant guarantees that no transient overload will occur. On the other hand, when a soft real-time system is overloaded and experiencing deadline misses, the prominent goal is to bring it under control by rate adaptation, and transition becomes a secondary issue.

We are also currently working toward the exploitation of the rate adaptation mechanisms presented in this paper into the real-time control of sensor-based, autonomous mobile and manipulator robots.

Acknowledgments

We thank Mirko Mazzoli for his valuable contribution in developing the new version of the real-time library and in running the on-line adaptation experiments.

Note

1. The potential variability of actual execution times with respect to assumed constant values further reinforces the case for an adaptive scheduling strategy.

References

Abdelzaher, T. F. and Shin, K. G 1998. End-host architecture for QoS-adaptive communication. In *Proc. IEEE Real-Time Technology and Application Symposium, RTAS'98*, Denver, CO.

Adusley, N. C., Burns, A., Richardson, M. F., Tindell, K. and Weillings, A. J 1991. Hard real-time scheduling: The deadline monotonic approach. In *Proc. IEEE Workshop on Real-Time Operating Systems and Software*.

Baruah, S. K., Chen, D. and Mok, A. K. 1997. Jitter concerns in periodic task systems. In *Proc. IEEE Real-Time Systems Symposium, RTSS'97*, San Francisco, CA.

Baruah, S. K. and Haritsa, J. R 1997. Scheduling for overload in real-time systems. *IEEE Transactions on Computers* 46(9):1034–1039.

Beccari, G., Caselli, S., Reggiani, M. and Zanichelli, F. 1998. A real-time library for the design of hybrid robot control architectures. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'98*, Victoria, BC.

Beccari, G., Caselli, S., Reggiani, M. and Zanichelli, F. 1999. Rate modulation of soft real-time tasks in autonomous robot control systems. In *Proc. Euromicro Conference on Real-Time Systems, ECRTS'99*, York, UK.

Burchard, A., Liebeherr, J., Oh, Y. and Son, S. H. 1995. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Transactions on Computers*, 44(12):1429–1442.

Buttazzo, G. C., Lipari, G. and Abeni, L. 1998. Elastic task model for adaptive rate control. In *Proc. IEEE Real-Time Systems Symposium, RTSS'98*, Madrid, Spain.

Buttazzo, G. C., Lipari, G., Caccamo, M. and Abeni, L. 2002. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3):289–302.

Buttazzo, G. and Stankovic, J. A. 1993. RED: A robust earliest deadline scheduling algorithm. In *Proc. 3rd International Workshop on Responsive Computing Systems*, Austin, TX.

Caccamo, M. and Buttazzo, G. 1997. Exploiting skips in periodic tasks for enhancing aperiodic responsiveness. In *Proc. IEEE Real-Time Systems Symposium, RTSS'97*, San Francisco, CA.

Floudas, C. A. and Visweswaran, V. 1995. Quadratic optimization In R. Horst and P. M. Pardalos, editors, *Handbook of Global Optimization*, pp. 217–269, Dordrecht: Kluwer Academic Publisher.

Franklin, G. F., Powell, J. D. and Emami-Naeini, A.: *Feedback Control of Dynamic Systems*, 3rd edition, Addison-Wesley, 1994.

Garey, M. R. and Johnson, D. S. 1979. *Computers and Intractability — A Guide to the Theory of NP-Completeness* New York: W. H. Freeman and Co.

Han, C. -C. and Tyan, H. -y. 1997. A better polynomial-time scheduling test for real-time fixed priority scheduling algorithms. In *Proc. IEEE Real-Time Systems Symposium, RTSS'97*, San Francisco, CA.

Hillier, F. S. and Lieberman, G. J. 2000. *Introduction to Operations Research*. 7th edition, New York: McGraw-Hill.

Horst, R. and Pardalos, P. M. (eds.). 1995. *Handbook of Global Optimization*. Dordrecht: Kluwer Academic Publishers.

Jehuda, J. and Israeli, A. 1998. Automated meta-control for adaptable real-time software. *Real-Time Systems* 14: 107–134.

Jones, M. B., Rosu, D. and Rosu, M. -C. 1997. CPU reservations and time constraints: Efficient, predictable scheduling of independent activities. In *Proc. 16th ACM Symposium on Operating Systems Principles*, Saint Malo, France, pp. 198–211.

Joseph, M. and Pandya, P. 1986. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395.

Koren, G. and Shasha, D. 1992. D-over: An optimal on-line scheduling algorithm for overloaded real-time systems. In *Proc. IEEE Real-Time Systems Symposium, RTSS'92*.

Kuo, B. C. 1992. *Digital Control Systems*, 2nd edition, Oxford University Press.

Kuo, T. -W. and Mok, A. K. 1997. Incremental Reconfiguration and Load Adjustment in Adaptive Real-Time Systems. *IEEE Transactions on Computer*, 46(12):1313–1324.

Lehoczky, J., Sha, L. and Ding, Y. 1989. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proc. IEEE Real-Time Systems Symposium, RTSS'89*, Santa Monica, CA.

Li, B. and Nahrstedt, K. 1998. A control theoretical model for quality of service adaptations. In *Proc. IEEE International Workshop on Quality of Service*.

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

Lu, C., Stankovic, J. A., Abdelzaher, T. F., Tao, G., Son, S. H. and Marley, M. 2000. Performance specifications and metrics for adaptive real-time systems. In *Proc. IEEE Real-Time Systems Symposium, RTSS'00*, Orlando, FL.

Lu, C., Stankovic, J. A., Tao, G. and Son, S. H. 1999. Design and evaluation of a feedback control edf scheduling algorithm. In *Proc. IEEE Real-Time Systems Symposium, RTSS'99*, Phoenix, AZ.

Marriot, K. and Stuckey, P. J. 1998. *Programming with Constraints — An Introduction*. MIT Press.

Musliner, D. J., Durfee, E. H. and Shin, K. G. 1993. CIRCA: A cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man, and Cybernetics* 23(6).

Nieh, J. and Lam, M. S. 1997. The design, implementation and evaluation of SMART: A scheduler for multimedia applications. In *Proc. 16th ACM Symposium on Operating Systems Principles*, Saint Malo, France, pp. 184–197.

Ramanathan, P. 1999. Overload management in real-time control applications using (m,k)-Firm Guarantee. *IEEE Transactions on Parallel and Distributed Systems*, 10(6).

Schoppers, M. 1994. A software architecture for hard real-time execution of automatically synthesized plans or control laws. In *Proc. Conf. on Intelligent Robotics in Field, Factory, Service, and Space (CIRFFSS'94)*.

Seto, D., Lehoczky, J. P., Sha, L. and Shin, K. G. 1996. On task schedulability in real-time control systems. In *Proc. IEEE Real-Time Systems Symposium, RTSS'96*, Washington, DC.

Seto, D., Lehoczky, J. P. and Sha, L. 1998. Task periodic selection and schedulability in real-time systems. In *Proc. IEEE Real-Time Systems Symposium, RTSS'98*, Madrid, Spain.

Shin, K. G. and Meissner, C. L. 1999. Adaptation and graceful degradation of control system performance by task reallocation and period adjustment In *Proc. Euromicro Conference on Real-Time Systems, ECRTS'99*, York, UK.

Stankovic, J. A. et al. 1996. Strategic directions in real-time and embedded systems. *ACM Computing Surveys*, 28(4).

Stankovic, J. A., Lu, C., Son, S. H. and Tao, G. 1999. The case for feedback control real-time scheduling. In *Proc. Euromicro Conference on Real-Time Systems, ECRTS'99*, York, UK.

Stankovic, J. A., Spuri, M., Ramamritham, K. and Buttazzo, G. C. 1998. *Deadline Scheduling for Real-Time Systems — EDF and Related Algorithms* Kluwer Academic Publishers.

Steere, D. C., Goel, A., Gruenberg, J., McNamee, D., Pu, C. and Walpole, J. 1999. A feedback-driven proportion allocator for real-rate scheduling. In *Proc. Third USENIX Symposium on Operating Systems Design and Implementation, OSDI'99*, New Orleans, LA, pp. 145–158.

Stewart, D. B. and Khosla, P. K. 1997. Mechanisms for detecting and handling timing errors. *Communications of the ACM*, 40(1).

Stewart, D. B., Schmitz, D. E. and Khosla, P. K. 1992. The Chimera II Real-time operating system for advanced sensor-based control applications. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(6).

Sun Microsystems 2000. Scalable Real-Time Computing in the Solaris Operating Environment Sun Microsystems Whitepaper, available at *http://www.sun.com/software/white-papers/wp-realtime/*.

Yau, D. K. Y. and Lam, S. S. 1997. Adaptive rate-controlled scheduling for multimedia applications. *IEEE Transactions on Networking*, 5(4).

Giuseppe Beccari received the Laurea degree in Electronic Engineering in 1993, and the Ph.D. in Information Technology in 1999, both from the University of Parma, Italy. In 1995 he was visiting scholar at the Technical University of Delft, Holland, and at the Laboratoire de Robotique de Paris, France. In 1999 he was employed by CSELT (Centro Studi E Laboratori Telecomunicazioni, currently TILAB, the Telecom Italia Group research center). In 2002 he moved to a spin off company involved in the EUROSAM/FSAF (Future Surface-to-Air Family self defense missile

system) project. While his current professional duties focus more on software development and team coordination, dr. Beccari still enjoys investigating real-time scheduling issues and technology.

Stefano Caselli received a Laurea degree in Electronic Engineering in 1982 and the Ph.D. degree in Computer and Electronic Engineering in 1987, both from the University of Bologna, Italy. In 1989-90 he has been visiting scholar at the University of Florida. From 1990 to 1999 he has held research fellow and associate professor positions at the University of Parma, Italy. He is now professor of Computer Engineering at the University of Parma, where he is also director of the Laboratory of Robotics and Intelligent Machines (RIMLab). His current research interests include development of autonomous and remotely operated robot systems, service robotics, and real-time systems.

Francesco Zanichelli received a Laurea degree in Electronic Engineering in 1987 from the University of Bologna, Italy and the Ph.D. degree in Information Technologies in 1994 from the University of Parma, Italy. Since 1996 he has been an Assistant Professor with the Department of Information Engineering of the University of Parma where he is currently teaching Operating Systems, Information Systems and Multimedia Systems courses. His current research interests include distributed multimedia architectures and protocols, real-time systems, security and Quality of Service technologies for wireless networks, as well as service-oriented Grid middleware.