Elastic Scheduling for Flexible Workload Management

Giorgio C. Buttazzo, *Member*, *IEEE*, Giuseppe Lipari, Marco Caccamo, and Luca Abeni

Abstract—An increasing number of real-time applications, related to multimedia and adaptive control systems, require greater flexibility than classical real-time theory usually permits. In this paper, we present a novel scheduling framework in which tasks are treated as springs with given elastic coefficients to better conform to the actual load conditions. Under this model, periodic tasks can intentionally change their execution rate to provide different quality of service and the other tasks can automatically adapt their periods to keep the system underloaded. The proposed model can also be used to handle overload conditions in a more flexible way and to provide a simple and efficient mechanism for controlling a system's performance as a function of the current load.

Index Terms—Real-time scheduling, overload management, rate adaptation.

1 Introduction

Periodic activities represent the major computational demand in many real-time applications since they provide a simple way to enforce timing constraints through rate control. For instance, in digital control systems, periodic tasks are associated with sensory data acquisition, low-level servoing, control loops, action planning, and system monitoring. In such applications, a necessary condition for guaranteeing the stability of the controlled system is that each periodic task is executed at a constant rate whose value is computed at the design stage based on the characteristics of the environment and on the required performance. For critical control applications (i.e., those whose failure may cause catastrophic consequences), the feasibility of the schedule has to be guaranteed a priori and no task can change its period while the system is running.

Such a rigid framework in which periodic tasks operate is also determined by the schedulability analysis that must be performed on the task set to guarantee its feasibility under the imposed constraints. To simplify the analysis, in fact, some feasibility tests for periodic tasks are based on quite rigid assumptions. For example, in the original Liu and Layland paper [13] on the Rate Monotonic (RM) and the Earliest Deadline First (EDF) algorithms, a periodic task τ_i is modeled as a cyclical processor activity characterized by two parameters, the computation time C_i and the period T_i, which are considered to be constant for all task instances. This is a reasonable assumption for most real-time control systems, but it can be too restrictive for other applications.

For example, in multimedia systems, timing constraints can be more flexible and dynamic than control theory usually permits. Activities such as voice sampling, image acquisition, sound generation, data compression, and video playing are performed periodically, but their execution rates are not as rigid as in control applications. Missing a deadline while displaying an MPEG video may decrease the quality of service (QoS), but does not cause critical system faults. Depending on the requested QoS, tasks may increase or decrease their execution rate to accommodate the requirements of other concurrent activities.

Even in some control applications, there are situations in which periodic tasks could be executed at different rates in different operating conditions. For example, in a flight control system, the sampling rate of the altimeters is a function of the current altitude of the aircraft: the lower the altitude, the higher the sampling frequency. A similar need arises in robotic applications in which robots have to work in unknown environments where trajectories are planned based on the current sensory information. If a robot is equipped with proximity sensors, in order to maintain a desired performance, the acquisition rate of the sensors must increase whenever the robot is approaching an obstacle.

In other situations, the possibility of varying tasks' rates increases the flexibility of the system in handling overload conditions, providing a more general admission control mechanism. For example, whenever a new task cannot be guaranteed by the system, instead of rejecting the task, the system can try to reduce the utilizations of the other tasks (by increasing their periods in a controlled fashion) to decrease the total load and accommodate the new request.

Unfortunately, there is no uniform approach for dealing with these situations. For example, Kuo and Mok [10] propose a load scaling technique to gracefully degrade the workload of a system by adjusting the periods of processes. In this work, tasks are assumed to be equally important and the objective is to minimize the number of fundamental frequencies to improve schedulability under static priority assignments. In [18], Nakajima and Tezuka show how a real-time system can be used to support an adaptive application: Whenever a deadline miss is detected, the

- *G.C. Buttazzo is with the University of Pavia, Via Ferrata, 1, 27100 Pavia, Italy. E-mail: buttazzo@unipv.it.*
- *G. Lipari, M. Caccamo, and L. Abeni are with the Scuola Superiore S. Anna, Via Carducci, 40, 56100 Pisa, Italy. E-mail: {lipari, caccamo, luca}@sssup.it.*

Manuscript received 17 May 2000; accepted 10 July 2001.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 112117.

0018-9340/02/$17.00 © 2002 IEEE

Authorized licensed use limited to: UNIVERSITA PISA S ANNA. Downloaded on October 7, 2008 at 6:19 from IEEE Xplore. Restrictions apply.

period of the failed task is increased. In [19], Seto et al. change tasks' periods within a specified range to minimize a performance index defined over the task set. This approach is effective at a design stage to optimize the performance of a discrete control system, but cannot be used for online load adjustment. In [12], Lee et al. propose a number of policies to dynamically adjust the tasks' rates in overload conditions. In [1], Abdelzaher et al. present a model for QoS negotiation to meet both predictability and graceful degradation requirements during overloads. In this model, the QoS is specified as a set of negotiation options, in terms of rewards and rejection penalties. In [16], [17], Nakajima shows how a multimedia activity can adapt its requirements during transient overloads by scaling down its rate or its computational demand. However, it is not clear how the the QoS can be increased when the system is underloaded. In [4], Beccari et al. propose several policies for handling overload through period adjustment. The authors, however, do not address the problem of increasing the task rates when the processor is not fully utilized.

Although these approaches may lead to interesting results in specific applications, we believe that a more general framework can be used to avoid a proliferation of policies and treat different applications in a uniform fashion.

Driven by the idea originally introduced in [7], this work presents a novel theoretical framework, the elastic task model, aimed at providing flexible workload management in real-time applications. In particular, the elastic approach provides the following advantages with respect to the classical "fixed-rate" approach:

- It allows tasks to intentionally change their execution rate to provide different quality of service;
- It can handle overload situations in a more flexible fashion;
- It provides a simple and efficient method for controlling the system's performance as a function of the current workload.

It is worth observing that the elastic approach is not limited to task scheduling. Rather, it represents a general resource allocation methodology which can be applied whenever a resource has to be allocated to objects whose constraints allow a certain degree of flexibility. For example, in a distributed system, dynamic changes in node transmission rates over the network could be efficiently handled by assigning each channel an elastic bandwidth, which could be tuned based on the actual network traffic.

Another interesting application of the elastic approach is to automatically adapt the task rates to the current load, without specifying the worst-case execution times of the tasks. If the system is able to monitor the actual execution time of each job, such data can be used to compute the actual processor utilization. If this is less than one, task rates can be increased according to elastic coefficients to fully utilize the processor. On the other hand, if the actual processor utilization is a little grater than one and some deadline misses are detected, task rates can easily be reduced to bring the processor utilization to a desired safe value.

TABLE 1
Task Set Parameters Used for the Example

Task	C_i	T_{i_0}	$T_{i_{max}}$	E_i
τ_1	10	20	25	1
τ_2	10	40	50	1
τ_3	15	35	80	1

The rest of the paper is organized as follows: Section 2 presents the elastic task model. Section 3 describes the guarantee algorithm for a set of elastic tasks. Section 4 extends the elastic approach in the presence of resource constraints. Section 5 presents some theoretical results which validate the proposed model. Section 6 illustrates some experimental results achieved on the HARTIK kernel. Finally, Section 7 contains our conclusions and future work.

2 The Elastic Model

The basic idea behind the elastic model proposed in this paper is to consider each task to be as flexible as a spring with a given rigidity coefficient and length constraints. In particular, the utilization of a task is treated as an elastic parameter whose value can be modified by changing the period within a specified range.

Each task is characterized by four parameters: a computation time C_i, a nominal period T_{i_0} (considered as the minimum period), a maximum period $T_{i_{max}}$, and an elastic coefficient $E_i \geq 0$, which specifies the flexibility of the task to vary its utilization for adapting the system to a new feasible rate configuration. The greater E_i, the more elastic the task. Thus, an elastic task is denoted as:

$$\tau_i(C_i, T_{i_0}, T_{i_{max}}, E_i).$$

In the following, T_i will denote the actual period of task τ_i, which is constrained to be in the range $[T_{i_0}, T_{i_{max}}]$. Any task can vary its period according to its needs within the specified range. Any variation, however, is subject to an *elastic* guarantee and is accepted only if there exists a feasible schedule in which all the other periods are within their range. Consider, for example, a set of three tasks, whose parameters are shown in Table 1. With periods $T_1 = 20$, $T_2 = 40$, and $T_3 = 70$, the task set is schedulable by EDF since

$$U_p = \frac{10}{20} + \frac{10}{40} + \frac{15}{70} = 0.964 < 1.$$

If task τ_3 reduces its period to 50, no feasible schedule exists since the utilization would be greater than 1:

$$U_p = \frac{10}{20} + \frac{10}{40} + \frac{15}{50} = 1.05 > 1.$$

However, notice that a feasible schedule exists ($U_p = 0.977$) for $T_1 = 22$, $T_2 = 45$, and $T_3 = 50$, hence, the system can accept the higher request rate of τ_3 by slightly decreasing the rates of τ_1 and τ_2. Task τ_3 can even run with a period $T_3 = 40$ since a feasible schedule exists with periods T_1 and T_2 within their range. In fact, when $T_1 = 24$, $T_2 = 50$, and $T_3 = 40$, $U_p = 0.992$. Finally, notice that if τ_3 requires

Authorized licensed use limited to: UNIVERSITA PISA S ANNA. Downloaded on October 7, 2008 at 6:19 from IEEE Xplore. Restrictions apply.

running at its minimum period ($T_3 = 35$), there is no feasible schedule with periods T_1 and T_2 within their range, hence, the request of τ_3 to execute with a period $T_3 = 35$ must be rejected.

Clearly, for a given value of T_3, there can be many different period configurations which lead to a feasible schedule; thus, one of the possible feasible configurations must be selected. The great advantage of using an elastic model is that the policy for selecting a solution is implicitly encoded in the elastic coefficients provided by the user (for example, based on task importance). Thus, each task is varied according to its current elastic status and a feasible configuration is found, if there one exists.

As another example, consider the same set of three tasks with the same initial periods ($T_1 = 20$, $T_2 = 40$, and $T_3 = 70$), but suppose that a new periodic task $\tau_4(5, 30, 30, 0)$ enters the system at time t. In a rigid scheduling framework, τ_4 (or some other task selected by a more sophisticated rejection policy) must be rejected because the new task set is not schedulable, being

$$U_p = \frac{10}{20} + \frac{10}{40} + \frac{15}{50} + \frac{5}{30} = 1.131 > 1.$$

Using an elastic model, however, τ_4 can be accepted if the periods of the other tasks can be increased in such a way that the total utilization is less than one and all the periods are within their range. In our specific example, the period configuration given by $T_1 = 23$, $T_2 = 50$, $T_3 = 80$, $T_4 = 30$ creates a total utilization $U_p = 0.989$, hence, τ_4 can be accepted.

The elastic model also works in the other direction. Whenever a periodic task terminates or decreases its rate, all the tasks that have been previously "compressed" (in utilization) can increase their rates or even return to their nominal periods, depending on the amount of released bandwidth.

It is worth noting that the elastic model is more general than the classical Liu and Layland task model, so it does not prevent a user from defining hard real-time tasks. In fact, a task having $T_{i_{max}} = T_{i_0}$ is equivalent to a hard real-time task with fixed period, independent of its elastic coefficient. A task with $E_i = 0$ can arbitrarily vary its period within its specified range, but it cannot be varied by the system during load reconfigurations.

2.1 Equivalence with a Spring System

To understand how an elastic guarantee is performed in this model, it is convenient to compare an elastic task τ_i with a linear spring S_i, characterized by a rigidity coefficient k_i, a nominal length x_{i_0}, and a minimum length $x_{i_{min}}$. In the following, x_i will denote the actual length of spring S_i, which is constrained to be greater than or equal to $x_{i_{min}}$.

In this comparison, the length x_i of the spring is equivalent to the task's utilization factor $U_i = C_i/T_i$ and the rigidity coefficient k_i is equivalent to the inverse of the task's elasticity ($k_i = 1/E_i$). Hence, a set of n tasks with total utilization factor $U_p = \sum_{i=1}^{n} U_i$ can be viewed as a sequence of n springs with total length $L = \sum_{i=1}^{n} x_i$.

Using the same notation introduced by Liu and Layland [13], let U_{lub}^{A} be the *least upper bound* of the total utilization factor for a given scheduling algorithm A (we recall that, for n tasks, $U_{lub}^{RM} = n(2^{1/n} - 1)$ and $U_{lub}^{EDF} = 1$). Hence, a task set can be schedulable by A if $U_p \leq U_{lub}^{A}$. Under EDF, such a schedulability condition becomes necessary and sufficient.

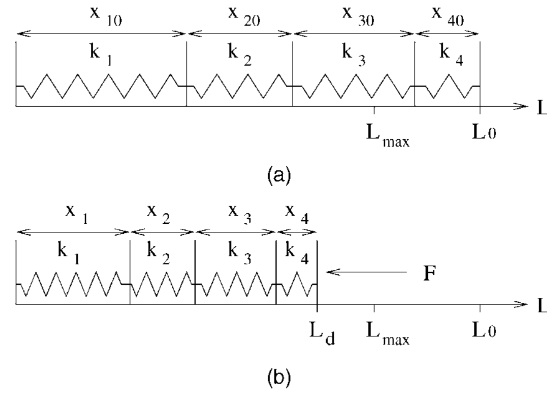

Fig. 1. A linear spring system: (a) The total length is L_0 when springs are uncompressed and (b) $L_d < L_0$ when springs are compressed by applying a force F.

Under the elastic model, given a scheduling algorithm A and a set of n tasks with $U_p > U_{lub}^{A}$, the objective of the guarantee is to compress tasks' utilization factors in order to achieve a new desired utilization $U_d \leq U_{lub}^{A}$ such that all the periods are within their ranges. In the linear spring system, this is equivalent of compressing the springs so that the new total length L_d is less than or equal to a given maximum length L_{max}. More formally, in the spring system, the problem can be stated as follows:

> Given a set of n springs with known rigidity and length constraints, if $L_0 = \sum_{i=1}^{n} x_{i_0} > L_{max}$, find a set of new lengths x_i such that $x_i \geq x_{i_{min}}$ and $L = L_d$, where L_d is any arbitrary desired length such that $L_d < L_{max}$.

For the sake of clarity, we first solve the problem for a spring system without length constraints, then we show how the solution can be modified by introducing length constraints, and, finally, we show how the solution can be adapted to the case of a task set.

2.2 Springs with No Length Constraints

Consider a set Γ of n springs with nominal length x_{i_0} and rigidity coefficient k_i positioned one after the other, as depicted in Fig. 1. Let L_0 be the total length of the array, that is the sum of the nominal lengths: $L_0 = \sum_{i=1}^{n} x_{i_0}$. If the array is compressed so that its total length is equal to a desired length L_d ($0 < L_d < L_0$), the first problem we want to solve is to find the new length x_i of each spring, assuming that, for all i, $0 < x_i < x_{i_0}$ (i.e., $x_{i_{min}} = 0$). L_d being the total length of the compressed array of springs, we have:

$$L_d = \sum_{i=1}^{n} x_i. \tag{1}$$

If F is the force that keeps a spring in its compressed state, then, for the equilibrium of the system, it must be:

$$\forall i \quad F = k_i(x_{i_0} - x_i),$$

from which we derive

Authorized licensed use limited to: UNIVERSITA PISA S ANNA. Downloaded on October 7, 2008 at 6:19 from IEEE Xplore. Restrictions apply.

$$\forall i \quad x_i = x_{i_0} - \frac{F}{k_i}. \tag{2}$$

By summing (2) we have:

$$L_d = L_0 - F\sum_{i=1}^{n}\frac{1}{k_i}.$$

Thus, force F can be expressed as

$$F = K_p(L_0 - L_d), \tag{3}$$

where

$$K_p = \frac{1}{\sum_{i=1}^{n}\frac{1}{k_i}}. \tag{4}$$

Substituting (3) into (2), we finally achieve:

$$\forall i \quad x_i = x_{i_0} - (L_0 - L_d)\frac{K_p}{k_i}. \tag{5}$$

Equation (5) allows us to compute how each spring has to be compressed in order to have a desired total length L_d.

2.3 Introducing Length Constraints

If each spring has a length constraint, in the sense that its length cannot be less than a minimum value $x_{i_{min}}$, then the problem of finding the values x_i requires an iterative solution. In fact, if, during compression, one or more springs reach their minimum length, the additional compression force will only deform the remaining springs. Thus, at each instant, the set Γ can be divided into two subsets: a set Γ_f of fixed springs having minimum length and a set Γ_v of variable springs that can still be compressed. Applying (5) to the set Γ_v of variable springs, we have

$$\forall S_i \in \Gamma_v \quad x_i = x_{i_0} - (L_{v_0} - L_d + L_f)\frac{K_v}{k_i}, \tag{6}$$

where

$$L_{v_0} = \sum_{S_i\in\Gamma_v} x_{i_0} \tag{7}$$

$$L_f = \sum_{S_i\in\Gamma_f} x_{i_{min}} \tag{8}$$

$$K_v = \frac{1}{\sum_{S_i\in\Gamma_v}\frac{1}{k_i}}. \tag{9}$$

Whenever there exists some spring for which (6) gives $x_i < x_{i_{min}}$, the length of that spring has to be fixed at its minimum value, sets Γ_f and Γ_v must be updated, and (6), (7), (8) and (9) recomputed for the new set Γ_v. If there exists a feasible solution, that is, if the desired final length L_d is greater than or equal to the minimum possible length of the array $L_{min} = \sum_{i=1}^{n} x_{i_{min}}$, the iterative process ends when each value computed by (6) is greater than or equal to its corresponding minimum $x_{i_{min}}$. The complete algorithm for compressing a set Γ of n springs with length constraints up to a desired length L_d is shown in Fig. 2.

```
Algorithm Spring_compress(Γ, L_d) {

    L_0 = Σ_{i=1}^{n} x_{i_0};
    L_min = Σ_{i=1}^{n} x_{i_min};
    if (L_d < L_min) return INFEASIBLE;

    do {

        Γ_f = {S_i | x_i = x_{i_min}};
        Γ_v = Γ − Γ_f;

        L_{v_0} = Σ_{S_i∈Γ_v} x_{i_0};
        L_f = Σ_{S_i∈Γ_f} x_{i_min};
        K_v = 1 / Σ_{S_i∈Γ_v} 1/k_i;

        ok = 1;
        for (each S_i ∈ Γ_v) {
            x_i = x_{i_0} − (L_{v_0} − L_d + L_f) K_v / k_i;
            if (x_i < x_{i_min}) {
                x_i = x_{i_min};
                ok = 0;
            }
        }

    } while (ok == 0);
    return FEASIBLE;
}
```

Fig. 2. Algorithm for compressing a set of springs with length constraints.

3 Compressing Tasks' Utilizations

When dealing with a set of elastic tasks, (6), (7), (8), and (9) can be rewritten by substituting all length parameters with the corresponding utilization factors and the rigidity coefficients k_i and K_v with the corresponding elastic coefficients E_i and E_v. Similarly, at each instant, the set Γ of periodic tasks can be divided into two subsets: a set Γ_f of fixed tasks having minimum utilization and a set Γ_v of variable tasks that can still be compressed. Let $U_{i_0} = C_i/T_{i_0}$ be the nominal utilization of task τ_i, $U_0 = \sum_{i=1}^{n} U_{i_0}$ be the nominal utilization of the task set, U_{v_0} be the sum of the nominal utilizations of tasks in Γ_v, and U_f be the total utilization factor of tasks in Γ_f. Then, to achieve a desired utilization $U_d < U_0$, each task has to be compressed up to the following utilization:

$$\forall \tau_i \in \Gamma_v \quad U_i = U_{i_0} - (U_{v_0} - U_d + U_f)\frac{E_i}{E_v}, \tag{10}$$

where

$$U_{v_0} = \sum_{\tau_i\in\Gamma_v} U_{i_0} \tag{11}$$

$$U_f = \sum_{\tau_i\in\Gamma_f} U_{i_{min}} \tag{12}$$

Authorized licensed use limited to: UNIVERSITA PISA S ANNA. Downloaded on October 7, 2008 at 6:19 from IEEE Xplore. Restrictions apply.

```
Algorithm Task_compress(Γ, U_d) {

  U_0 = Σ_{i=1}^{n} C_i/T_{i_0};
  U_min = Σ_{i=1}^{n} C_i/T_{i_max};
  if (U_d < U_min) return INFEASIBLE;

  do {

      U_f = E_v = 0;
      for (each τ_i) {
          if ((E_i == 0) or (T_i == T_{i_max}))
              U_f = U_f + U_i;
          else E_v = E_v + E_i;
      }

      U_{v_0} = U_0 - U_f;

      ok = 1;
      for (each τ_i ∈ Γ_v) {
          if ((E_i > 0) and (T_i < T_{i_max})) {
              U_i = U_{i_0} - (U_{v_0} - U_d + U_f)E_i/E_v;
              T_i = C_i/U_i;
              if (T_i > T_{i_max}) {
                  T_i = T_{i_max};
                  ok = 0;
              }
          }
      }

  } while (ok == 0);
  return FEASIBLE;
}
```

Fig. 3. Algorithm for compressing a set of elastic tasks.

$$E_v = \sum_{\tau_i \in \Gamma_v} E_i. \tag{13}$$

If there exist tasks for which $U_i < U_{i_{min}}$, then the period of those tasks has to be fixed at its maximum value $T_{i_{max}}$ (so that $U_i = U_{i_{min}}$), sets Γ_f and Γ_v must be updated (hence, U_f and E_v recomputed), and (10) applied again to the tasks in Γ_v. If there exists a feasible solution, that is, if the desired utilization U_d is greater than or equal to the minimum possible utilization $U_{min} = \sum_{i=1}^{n} \frac{C_i}{T_{i_{max}}}$, the iterative process ends when each value computed by (10) is greater than or equal to its corresponding minimum $U_{i_{min}}$. The algorithm[1] for compressing a set Γ of n elastic tasks up to a desired utilization U_d is shown in Fig. 3.

3.1 Decompression

All tasks' utilizations that have been compressed to cope with an overload situation can return toward their nominal values when the overload is over. Let Γ_c be the subset of compressed tasks (that is, the set of tasks with $T_i > T_{i_0}$), let Γ_a be the set of remaining tasks in Γ (that is, the set of tasks with $T_i = T_{i_0}$), and let U_d be the current processor utilization of Γ. Whenever a task in Γ_a voluntarily increases its period, all tasks in Γ_c can expand their utilizations according to their elastic coefficients so that the processor utilization is kept at the value of U_d.

Now, let U_c be the total utilization of Γ_c, let U_a be the total utilization of Γ_a after some task has increased its period, and let U_{c_0} be the total utilization of tasks in Γ_c at their nominal periods. It can easily be seen that if $U_{c_0} + U_a \le U_{lub}$, all tasks in Γ_c can return to their nominal periods. On the other hand, if $U_{c_0} + U_a > U_{lub}$, then the release operation of the tasks in Γ_c can be viewed as a compression, where $\Gamma_f = \Gamma_a$ and $\Gamma_v = \Gamma_c$. Hence, it can still be performed by using (10), (12), and (13) and the algorithm presented in Fig. 3.

3.2 Period Rescaling

If the elastic coefficients are set equal to task nominal utilizations, elastic compression has the effect of a simple rescaling, where all the periods are increased by the same percentage. In order to work correctly, however, period rescaling must be uniformly applied to all the tasks, without restrictions on the maximum period. This means having $U_f = 0$ and $U_{v_0} = U_0$. Under this assumption, by setting $E_i = U_{i_0}$, (10) becomes:

$$\forall i \quad U_i = U_{i_0} - (U_0 - U_d)\frac{U_{i_0}}{U_0} = \frac{U_{i_0}}{U_0}[U_0 - (U_0 - U_d)] = \frac{U_{i_0}}{U_0}U_d$$

from which we have that

$$T_i = T_{i_0}\frac{U_0}{U_d}.$$

This means that, in overload situations ($U_0 > 1$), the compression algorithm causes all task periods to be increased by a common scale factor

$$\eta = \frac{U_0}{U_d}.$$

Notice that, after compression is performed, the total processor utilization becomes:

$$U = \sum_{i=1}^{n} \frac{C_i}{\eta T_{i_0}} = \frac{1}{\eta}U_0 = \frac{U_d}{U_0}U_0 = U_d$$

as desired.

If a maximum period needs to be defined for some task, an online guarantee test can easily be performed before compression to check whether all the new periods are less than or equal to the maximum value. This can be done in $O(n)$ by testing whether

$$\forall i = 1, \ldots, n \quad \eta T_{i_0} \le T_i^{max}.$$

By deciding to apply period rescaling, we lose the freedom of choosing the elastic coefficients since they must be set equal to task nominal utilizations. However, this technique has the advantage of leaving the task periods ordered as in the nominal configuration, which simplifies the compression algorithm in the presence of resource constraints, as discussed in Section 4.3.

1. The actual implementation of the algorithm contains more checks on tasks' variables, which are not shown here to simplify its description.

Authorized licensed use limited to: UNIVERSITA PISA S ANNA. Downloaded on October 7, 2008 at 6:19 from IEEE Xplore. Restrictions apply.

4 Handling Resource Constraints

We now extend the elastic model to deal with resource constraints, thus allowing tasks to interact through shared memory buffers. In order to estimate maximum blocking times due to mutual exclusion and analyze task schedulability, we assume that critical sections are accessed through the Stack Resource Policy [2]. For the sake of completeness, the features and the main properties of this protocol are briefly recalled below.

4.1 The Stack Resource Policy

The Stack Resource Policy (SRP) is a concurrency control protocol proposed by Baker [2] to bound the priority inversion phenomenon in static as well as dynamic priority systems. Under the EDF scheduling algorithm, each task τ_i is assigned a dynamic priority p_i inversely proportional to its absolute deadline d_i and a static *preemption level* π_i, such that the following property holds:

Property 1. *Task* τ_i *is not allowed to preempt task* τ_j *unless* $\pi_i > \pi_j$.

Under EDF, Property 1 is verified if each periodic task is assigned a preemption level inversely proportional to its relative deadline D_i. That is,

$$\pi_i \propto \frac{1}{D_i}.$$

In addition, every resource R_k is assigned a static[2] *ceiling*, defined as

$$ceil(R_k) = \max_i \{\pi_i \mid \tau_i \ \ needs \ \ R_k\}, \tag{14}$$

and a dynamic *system ceiling* is defined as

$$\Pi_s(t) = \max[\{ceil(R_k) \mid R_k \text{ is currently busy}\} \cup \{0\}].$$

Then, the SRP scheduling rule states that:

> a task is not allowed to preempt until its priority is the highest among those of the active tasks and its preemption level is greater than the system ceiling.

Such a protocol guarantees that each task can be blocked for at most the duration of one critical section. Moreover, it ensures that, once a task is started, it will never block until completion; it can only be preempted by higher priority tasks. As a consequence, the blocking time B_i considered in the schedulability analysis refers to the time during which task τ_i is kept in the ready queue by the preemption test, waiting for tasks with lower preemption levels to free shared resources. Blocking at preemption time also allows tasks to share a single stack, so reducing the total stack size when more tasks have the same preemption level. Finally, the SRP implementation is straightforward and there is no need to implement semaphore queues since a task never blocks during execution, but simply cannot preempt if its preemption level is not high enough.

Using the SRP, the maximum blocking time for a task τ_i is bounded by the duration of the longest critical section among those that can block τ_i, that is, those with a ceiling greater than or equal to π_i belonging to tasks with a relative deadline greater than D_i:

$$B_i = \max_{j,h}\{s_{jh} \mid (D_i < D_j) \ \ \wedge \ \ \pi_i \leq \mathrm{ceil}(\rho_{jh})\}, \tag{15}$$

where s_{jh} is the worst-case execution time of the hth critical section of task τ_j and ρ_{jh} is the resource used inside it. Then, the feasibility of a task set with resource constraints can be tested by the following sufficient condition [2]:

$$\forall i, \ \ 1 \leq i \leq n \quad \sum_{k=1}^{i} \frac{C_k}{D_k} + \frac{B_i}{D_i} \ \leq \ 1, \tag{16}$$

where it is assumed that tasks are sorted by decreasing preemption levels so that $\pi_i \geq \pi_j$ only if $i < j$.

A simpler, but less tight, sufficient condition to verify the schedulability of a task set in the presence of resource constraints can be derived from condition (16):

$$\sum_{i=1}^{n} \frac{C_i}{D_i} + \max_k \left(\frac{B_k}{D_k}\right) \ \leq \ 1. \tag{17}$$

A tighter test can be performed (in pseudopolynomial time) using a *processor demand criterion* [6]. However, in order to keep the runtime overhead low, we decided to perform the compression using the simplest test given by (17).

4.2 The Compression Algorithm

In the presence of resource constraints, the compression algorithm must be modified to take blocking terms into account. Using the simplified schedulability test expressed in (17) and, under the assumption that relative deadlines are equal to actual periods, the utilizations of the compressed tasks can be computed as follows:

$$\forall \tau_i \in \Gamma_v \quad U_i = U_{i_0} - \left[U_0 - U_d + \max_k\left(\frac{B_k}{T_k}\right) + U_f\right]\frac{E_i}{E_v}. \tag{18}$$

It is worth noting that, even in this simple case, performing compression using the actual blocking factors would increase the worst-case complexity of the algorithm since blocking times are not constant, but depend on the reciprocal period relations. Thus, if, during compression, the order of periods is not preserved, the blocking factor of some task could increase significantly and make the new task configuration infeasible. In this case, the algorithm should perform an additional compression step by using the actual blocking times computed under the new period configuration, but the same problem could arise again. Unfortunately, bounding the number of additional steps required to get a feasible schedule in the presence of resource constraints is not easy, so we decided to adopt an approximate solution which allows us to keep the complexity of the compression algorithm the same as without resource constraints.

The key idea is to overestimate the $max_k(\frac{B_k}{T_k})$ term with a blocking utilization factor U_b which is independent from the period configuration. If B_k^{wc} denotes the worst-case blocking time of τ_k for each possible periods' combination, we have that

2. In the case of multiunit resources, the ceiling of each resource is dynamic as it depends on the number of units actually free.

Authorized licensed use limited to: UNIVERSITA PISA S ANNA. Downloaded on October 7, 2008 at 6:19 from IEEE Xplore. Restrictions apply.

$$\max_k\left(\frac{B_k}{T_k}\right) \leq \max_k\left(\frac{B_k^{wc}}{T_{k_0}}\right).$$

Hence, the compression can be performed according to the following equation:

$$\forall \tau_i \in \Gamma_v \quad U_i = U_{i_0} - (U_0 - U_d + U_b + U_f)\frac{E_i}{E_v}, \tag{19}$$

where

$$U_b = \max_k\left(\frac{B_k^{wc}}{T_{k_0}}\right).$$

It is worth observing that, since U_b is constant with respect to period change, it can be computed offline, so it does not affect the complexity of the compression algorithm. To compute U_b, the worst-case blocking time B_k^{wc} must be estimated as a function of the minimum relative deadline D_i^{min} and the maximum relative deadline D_i^{max} of each task τ_i (we have $D_i^{min} = T_{i_0}$ and $D_i^{max} = T_{i_{max}}$). By fixing D_i^{min}, we can assign each task τ_i a *maximum preemption level* $\pi_i^{max} = 1/D_i^{min}$ that will be used to compute the *maximum ceiling* $ceil^{max}(R_k)$ of each resource offline. More formally, $ceil^{max}(R_k)$ can be defined as follows:

$$ceil^{max}(R_k) = \max_i\{\pi_i^{max} \mid \tau_i \ \ needs \ \ R_k\}. \tag{20}$$

The maximum ceiling of a resource represents the highest ceiling for any possible period combination; this notion is used for computing the worst-case blocking time B_i^{wc}, which is defined as follows:

$$B_i^{wc} = \max\{s_{jh} \mid (D_i^{min} < D_j^{max}) \ \wedge \ \pi_i^{min} \leq ceil^{max}(\rho_{jh})\}, \tag{21}$$

where $\pi_i^{min} = 1/D_i^{max}$ represents the lowest value of π_i for any possible period combination. If D_i is the actual relative deadline of elastic task τ_i, the following inequality holds:

$$\forall i, \ \ \pi_i^{min} \leq \frac{1}{D_i} \leq \pi_i^{max}.$$

The consequence of overestimating the blocking factor U_b is that the compression algorithm cannot exploit all the available bandwidth. However, if the feasibility test is satisfied, the compression algorithm always gives a solution which, in the worst case, consists of setting each task period equal to its maximum value.

Finally, to take resource constraints into account, the feasibility test in the original algorithm shown in Fig. 3 has to be modified as follows:

if $(U_d < U_{min} + max_k(\frac{B_k^*}{T_{k_{max}}}))$ return INFEASIBLE;

where B_k^* is the maximum blocking time computed by (15) assuming that each task has a period equal to $T_{k_{max}}$.

4.3 Rescaling with Resource Constraints

The compression algorithm can be greatly simplified if a uniform rescaling policy is used to handle overloads. In fact, as shown in Section 3.2, the rescaling algorithm reduces the load by increasing all the periods by the same scale factor

$$\eta = \frac{U_0}{U_d}.$$

This means that the compression algorithm does not affect the order of periods in the task set (and the order of preemption levels). As a consequence, the blocking times do not change during period adjustment and can be computed offline without making overestimations. Hence, (18) can be simplified as:

$$\forall \tau_i \quad U_i = U_{i_0} - \left[U_0 - U_d + \max_k\left(\frac{B_k}{T_k}\right)\right]\frac{U_{i_0}}{U_0}. \tag{22}$$

In order to compute the new periods, we start resolving the equation corresponding to task τ_j such that

$$\frac{B_j}{T_j} = \max_k\left(\frac{B_k}{T_k}\right).$$

Notice that j can be computed offline. In fact, in the case of rescaling, the relative order of the periods and the blocking factors do not change at runtime; hence,

$$\forall i \quad \frac{B_i}{T_{i_0}} < \frac{B_j}{T_{j_0}} \Rightarrow \frac{B_i}{T_i} < \frac{B_j}{T_j}.$$

Thus, j can be computed offline as the index of that task τ_j such that

$$\frac{B_j}{T_{j_0}} = \max_k\left(\frac{B_k}{T_{k_0}}\right).$$

Once j has been identified, the jth equation can be written as:

$$\frac{C_j}{T_j} = U_{j_0} - \left[U_0 - U_d + \frac{B_j}{T_j}\right]\frac{U_{j_0}}{U_0},$$

and solved for T_j:

$$\frac{C_j}{T_j} = U_d\frac{U_{j_0}}{U_0} - \frac{B_j U_{j_0}}{T_j U_0}$$

$$\frac{1}{T_j}\left(C_j + B_j\frac{U_{j_0}}{U_0}\right) = U_d\frac{U_{j_0}}{U_0}$$

$$T_j = \frac{C_j\frac{U_0}{U_{j_0}} + B_j}{U_d}.$$

After computing T_j, the term $\max_k(\frac{B_k}{T_k}) = \frac{B_j}{T_j}$ is known and can be substituted in all the remaining equations to compute the periods of the other tasks. Notice that, in this case, the solution is not approximated, that is:

$$\sum_{i=1}^{n} U_i + \max_k\left(\frac{B_k}{T_k}\right) = U_d.$$

In conclusion, in the presence of resource constraints, using a rescaling method in place of the most general one has some interesting advantages:

- It allows us to obtain an exact solution;
- Since the relative order of the tasks' periods does not change during compression/decompression, a task can only be preempted by the same set of tasks that have a shorter period;

Authorized licensed use limited to: UNIVERSITA PISA S ANNA. Downloaded on October 7, 2008 at 6:19 from IEEE Xplore. Restrictions apply.

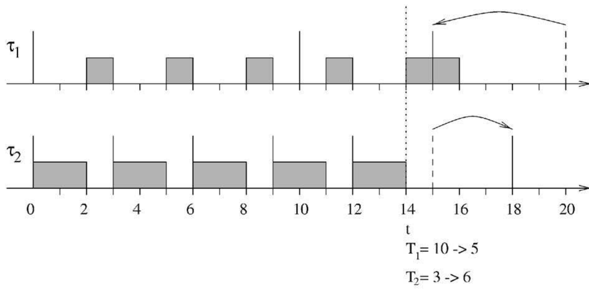

Fig. 4. A task can miss its deadline if a period is decreased at arbitrary time.

- Finally, since the order is preserved, it is possible to perform some offline optimization. For example, we can find an ordering such that the maximum blocking time is minimized.

5 Theoretical Validation of the Model

In this section, we derive some theoretical results which validate the elastic approach. In particular, we show that, in order to avoid transient overloads during a transition, a period reconfiguration has to be performed at opportune time instants to preserve the schedulability of the task set.

The amount of delay needed to perform a safe transition depends on the particular state in which the system is found when the reconfiguration is required. In the following, we first address the problem in the case of independent tasks and then we extend the results in the presence of resource constraints.

5.1 The Case of Independent Tasks

Intuitively, when tasks do not share resources (except for the processor), the period of those tasks decreasing their rate can be changed immediately, without compromising the schedulability. In fact, since constraints are being relaxed, tasks that were schedulable before compression will still be schedulable after their period is enlarged.

On the other hand, if a new task has to be accepted in the system (or a task rate has to be increased), the mode change cannot occur immediately, but it must be delayed until the required utilization is fully available. The problem will be presented through the following examples.

5.1.1 Example A: Shortening a Period

Consider the example shown in Fig. 4, where two tasks, with computation times $C_1 = 3$ and $C_2 = 2$ and periods $T_1 = 10$ and $T_2 = 3$, start at time 0. The processor utilization is $U_p = \frac{29}{30}$, thus the task set is schedulable by EDF. Suppose that, at time $t = 14$, τ_1 wants to change its period from $T_1 = 10$ to $T_1' = 5$ so that the compression algorithm increases the period of τ_2 from $T_2 = 3$ to $T_2' = 6$ to keep the system schedulable. The new processor utilization is $U_p' = \frac{28}{30}$, so the task set is still schedulable; however, if periods are changed immediately (i.e., at time $t = 14$), task τ_1 misses its deadline at time $t = 15$.

In general, although the periods of the tasks that decrease their rate can be changed immediately, the periods of the tasks that increase their rate can be changed only at their next release time.

TABLE 2
Task Set Parameters for Example B

Task	C_i	T_{i_0}	$T_{i_{max}}$	E_i
τ_1	5	10	20	1
τ_2	5	10	10	0
τ_3	1	4	4	0

5.1.2 Example B: New Task Arrival

Consider the periodic task set reported in Table 2 and suppose that τ_1 and τ_2 start at time $t = 0$, whereas τ_3 arrives at time $t = 5$. Since the three tasks cannot be scheduled with their nominal periods, the compression algorithm is invoked at time $t = 5$ to accommodate the execution of τ_3. As can easily be verified, a feasible solution exists if the period of τ_1 is increased at the value $T_1 = 20$, in fact:

$$U_p = \frac{5}{20} + \frac{5}{10} + \frac{1}{4} = 1.$$

Although the new task set is schedulable ($U_p = 1$), Fig. 5a shows that, if τ_3 is started at time $t = 5$, τ_2 misses its deadline. Why does it happen?

The reason for the deadline miss can be explained as follows: At time $t = 5$, when the period of τ_1 is increased from $T_1 = 10$ to $T_1' = 20$, its utilization decreases from $U_1 = 0.5$ to $U_1' = 0.25$, creating the required bandwidth for τ_3. However, such a bandwidth is not available immediately because τ_1 already executed in its period, consuming all the bandwidth allocated to it until time $t = 10$. As a consequence, the freed utilization will be available for τ_3 from time $t = 10$ on. Indeed, Fig. 5b shows that if the activation of τ_3 is delayed till time $t = 10$, no deadline is missed and the new task set is schedulable.

In general, if, at time t, a task τ_i increases its period from T_i to T_i', its execution can be split into two pieces: a portion $e_i(t)$ already executed up to t and a portion $c_i(t) = C_i - e_i(t)$ to be executed with the new period. If r_i is the release time of the current instance of τ_i, the portion $e_i(t)$ is executed with deadline $d_i = r_i + T_i$, whereas the remaining portion $c_i(t)$ will be executed with deadline $d_i' = r_i + T_i'$. Since, in the interval $[r_i, t]$ only $e_i(t)$ units of time (out of C_i) have been used by τ_i, the bandwidth U_i was consumed up to time δ_i such that:

$$U_i = \frac{e_i(t)}{\delta_i - r_i},$$

from which we derive that

$$\delta_i = r_i + \frac{e_i(t)}{U_i} = r_i + \frac{C_i - c_i(t)}{U_i} = r_i + \frac{C_i}{U_i} - \frac{c_i(t)}{U_i} = d_i - \frac{c_i(t)}{U_i}.$$

Hence, only from time δ_i on, the freed utilization $(U_i - U_i')$ is available to the system and can be allocated to other tasks. In general, if $\mathcal{T}_c$ is the set of tasks that decrease their rates at time t, the total bandwidth $(U_p - U_p')$ created by the compression algorithm will be available at time:

$$\delta_{max} = \max_{\tau_i \in \mathcal{T}_c}(\delta_i). \tag{23}$$

Authorized licensed use limited to: UNIVERSITA PISA S ANNA. Downloaded on October 7, 2008 at 6:19 from IEEE Xplore. Restrictions apply.

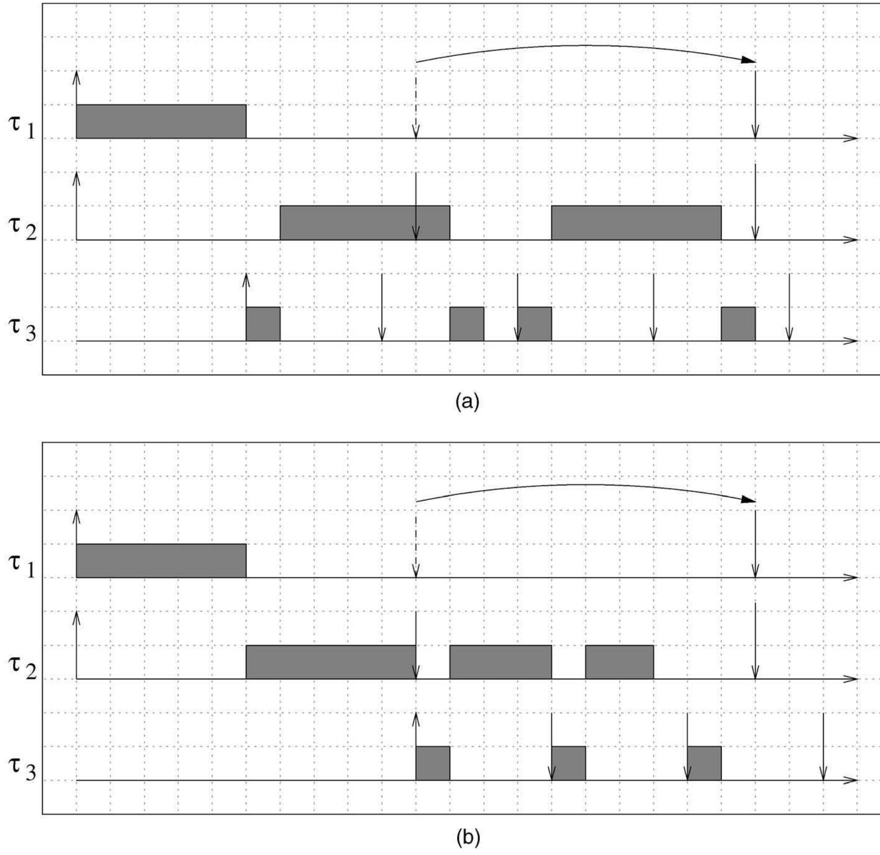

Fig. 5. (a) Task set schedulability is jeopardized if a task is started when compression is performed, (b) but no deadline is missed if the creation of the new task is postponed at an opportune time.

Note that the residual computation time $c_i(t) = C_i - e_i(t)$ of the current instance of τ_i can also be scheduled with a deadline d_i^* such that, in $[\delta_i, d_i^*]$, a bandwidth $U_i' = C_i/T_i'$ is consumed. That is,

$$d_i^* = \delta_i + \frac{c_i(t)}{U_i'}.$$

In the following, we use the *processor demand criterion* [6], originally proposed in [3], to formally prove that, if all the deadlines of the compressed tasks are postponed from d_i to d_i^*, then, from δ_{max} on, the bandwidth of the system will be no greater than $U_p' = \sum U_i'$.

Definition 1. *The processor demand of task τ_i in any interval $[t_1, t_2]$, denoted as $D_i(t_1, t_2)$, is the sum of the computation times of all the instances of τ_i with arrival time greater than or equal to t_1 and deadline less than or equal to t_2.*

The following theorem, proven in [9], will be used to verify the schedulability of a task set under the EDF algorithm.

Theorem 1 (Jeffay and Stone, 93). *A task set $\mathcal{T}$ is schedulable by EDF if and only if, in every interval, the total demand of the task set is less than or equal to the length of the interval, that is, if and only if*

$$\forall t_1, t_2 \;:\; t_1 < t_2 \quad D(t_1, t_2) = \sum_i D_i(t_1, t_2) \leq (t_2 - t_1).$$

We now bound the demand of a compressed task with the following lemma.

Lemma 1. *For any task $\tau_i \in \mathcal{T}_c$, its demand $D_i(t_1, t_2)$ in every interval $[t_1, t_2]$ such that $t_1 \leq \delta_i$ and $t_2 \geq \delta_i$ can be bounded as follows:*

$$D_i(t_1, t_2) \leq (\delta_i - t_1)U_i + (t_2 - \delta_i)U_i'. \qquad (24)$$

Proof. The demand of task τ_i can be written as the sum of two components, D_i^{old} and D_i^{new}. The D_i^{old} component takes into account the instances of τ_i with deadline less than or equal to d_i, that is, the instances executing before the period change. The D_i^{new} component takes into account the instances of τ_i with a deadline greater than or equal to d_i^*, that is, those instances executing after the period change. Notice that the instance of τ_i executing at the compression time t contributes to both components. Hence, $\forall t_1 \leq \delta_i,\ t_2 \geq \delta_i$,

$$D_i(t_1, t_2) = D_i^{old}(t_1, t_2) + D_i^{new}(t_1, t_2).$$

Considering the first component, we have that $\forall t_1 \leq r_i,\ t_2 \geq d_i$,

Authorized licensed use limited to: UNIVERSITA PISA S ANNA. Downloaded on October 7, 2008 at 6:19 from IEEE Xplore. Restrictions apply.

$$D_i^{old}(t_1,t_2) \le \left\lfloor \frac{d_i - t_1}{T_i} \right\rfloor C_i - c_i(t)$$
$$\le (d_i - t_1)U_i - c_i(t) = \left(d_i - \frac{c_i(t)}{U_i}\right)U_i - t_1 U_i$$
$$= (\delta_i - t_1)U_i.$$

$\forall t_1,\ r_i < t_1 \le \delta_i$, $\forall t_2,\ \delta_i \le t_2 < d_i$

$$D_i^{old}(t_1,t_2) = 0 \ \le\ (\delta_i - t_1)U_i.$$

Now, considering the second component, we have that $\forall t_1 \le r_i,\ t_2 \ge d_i^*$,

$$D_i^{new}(t_1,t_2) \le c_i(t) + \left\lfloor \frac{t_2 - d_i^*}{T_i'} \right\rfloor C_i \le$$
$$= c_i(t) + \left\lfloor \frac{t_2 - (\delta_i + \frac{c_i(t)}{U_i'})}{T_i'} \right\rfloor C_i \le$$
$$\le c_i(t) + (t_2 - \delta_i)U_i' - c_i(t) = (t_2 - \delta_i)U_i'.$$

$\forall t_1,\ r_i < t_1 \le \delta_i$, $\forall t_2,\ \delta_i \le t_2 < d_i^*$

$$D_i^{new}(t_1,t_2) = 0 \ \le\ (t_2 - \delta_i)U_i'.$$

Hence, the thesis follows. □

We now prove that δ_i is the earliest time after which the bandwidth created by the compression algorithm is fully available. To do that, we illustrate a counterexample which shows that Lemma 1 is no longer valid for any $\delta_i^{new} < \delta_i$. Suppose that a task $\bar{\tau}$ with bandwidth $\bar{U} = 1$, period and execution time equal to six units of time, starts at time zero and executes until $t = 3$. At this time, its period is changed in order to decrease the utilization from 1 to 0.5. Clearly, $\bar{\delta} = 3$ and the residual computation time is $\bar{c}(t) = 3$. By setting $\bar{\delta}^{new} = 3 - \epsilon$, then $\bar{d}^* = \bar{\delta}^{new} + \frac{\bar{c}(t)}{\bar{U}'} = 9 - \epsilon$. Hence, the processor demand of $\bar{\tau}$ in $[0, \bar{d}^*]$ is six units of time, whereas (24) gives

$$D_i(t_1,t_2) \le (\bar{\delta}^{new} - t_1)U_i + (t_2 - \bar{\delta}^{new})U_i'$$
$$\le (3-\epsilon) + (9 - \epsilon - 3 + \epsilon)0.5 = 6 - \epsilon.$$

We now formally prove that, after time δ_{max}, the bandwidth of the task set is not greater than U_p'.

Theorem 2. *Let $\mathcal{T}$ be a task set with utilization U_p and let $\mathcal{T}_c$ be the subset of tasks that, at time t, increase their period so that the total processor utilization is compressed down to $U_p' < U_p$. Let us define time δ_{max} as in (23). Then, from time δ_{max} on, the bandwidth used by the task set is not greater than U_p'.*

Proof. The thesis is proven by showing that, for every possible task τ_{new} arriving at time δ_{max} with bandwidth $U_{new} \le 1 - U_p'$, the system remains schedulable. By using the *processor demand criterion*, it is sufficient to show that, in every interval of time $[t_1, t_2]$, the demand of the task set, including the new task, does not exceed the length of the interval. We distinguish three cases:

- $\forall t_1, t_2,\ t_1 < t_2 \le \delta_{max}$, the new task is not included in these intervals and, for $t_2 \in [\delta_i, \delta_{max}]$, the demand of τ_i can be bounded using Lemma 1. Hence:

$$D(t_1,t_2) \le \sum_{t_2<\delta_i} (t_2 - t_1)U_i$$
$$+ \sum_{\delta_i \le t_2 \le \delta_{max}} \{(\delta_i - t_1)U_i + (t_2 - \delta_i)U_i'\}$$
$$\le \sum_{\tau_i \in \mathcal{T}} (t_2 - t_1)U_i \le (t_2 - t_1)U_p \le (t_2 - t_1).$$

- $\forall t_1 < \delta_{max},\ t_2 > \delta_{max}$, the demand in these intervals is the sum of the old and the new instances of the compressed tasks and the instances of τ_{new}. Hence:

$$D(t_1,t_2) \le \sum_{t_1<\delta_i} \{(\delta_i - t_1)U_i + (t_2 - \delta_i)U_i'\}$$
$$+ \sum_{\delta_i \le t_1 < \delta_{max}} (t_2 - t_1)U_i' + (t_2 - \delta_{max})U_{new}$$
$$\le \sum_{\tau_i \in \mathcal{T}} (\delta_{max} - t_1)U_i + \sum_{\tau_i \in \mathcal{T}} (t_2 - \delta_{max})U_i'$$
$$+ (t_2 - \delta_{max})U_{new}$$
$$\le (\delta_{max} - t_1)U_p + (t_2 - \delta_{max})U_p'$$
$$+ (t_2 - \delta_{max})U_{new} \le (t_2 - t_1).$$

- $\forall t_1, t_2,\ \delta_{max} \le t_1 < t_2$, only new instances must be considered in these intervals. Hence:

$$D(t_1,t_2) \le \sum_{\tau_i \in \mathcal{T}} (t_2 - t_1)U_i' + (t_2 - t_1)U_{new} \le (t_2 - t_1).$$

Hence, if a new task is created at δ_{max} with bandwidth $U_{new} \le 1 - U_p'$ and arbitrary period, the system remains schedulable. This implies that, from time δ_{max} on, the bandwidth of the task set is no greater than U_p'. □

It is worth observing that, in order to simplify the implementation of the compression algorithm, each current compressed instance can be scheduled with a deadline $d_i' = r_i + T_i'$, rather than with d_i^*. Changing d_i^* with d_i' does not jeopardize the task set schedulability since $d_i^* \le d_i'$, as proven by the following lemma.

Lemma 2. *For every task $\tau_i \in \mathcal{T}_c$ whose period is increased from T_i to $T_i' > T_i$, the minimum deadline d_i^* that can be used to schedule its residual computation time $c_i(t)$ is such that*

$$d_i^* \le d_i'.$$

Proof. From the definition of d_i^* we have:

$$d_i^* = \delta_i + \frac{c_i(t)}{U_i'} = \left(d_i - \frac{c_i(t)}{U_i}\right) + \frac{c_i(t)}{U_i'}$$
$$= d_i - \frac{c_i(t)}{C_i}\left(T_i - T_i'\right) \le d_i - T_i + T_i' = r_i + T_i' = d_i'.$$

Thus, the lemma follows. □

5.2 The Case of Resource Constrained Tasks

In the presence of resources constraints, additional care must be taken before applying the compression algorithm in order to preserve the main properties of the SRP. The following example illustrates a problem which can arise if

Authorized licensed use limited to: UNIVERSITA PISA S ANNA. Downloaded on October 7, 2008 at 6:19 from IEEE Xplore. Restrictions apply.

TABLE 3
Parameters of the Task Set

$task$	C_i	T_{i_0}	$T_{i_{max}}$	E_i	R_1	R_2
τ_1	1	10	10	0	-	-
τ_2	4	11	16	1	2	-
τ_3	2	10	14	1	1	1
τ_4	4	20	20	0	2	2

periods are changed while some task is inside a critical section.

Consider a task set consisting of four periodic tasks τ_1, τ_2, τ_3, and τ_4, which share two resources R_1 and R_2, as shown in Table 3. The numbers reported in the R_i columns represent the time that each task needs to execute the critical section corresponding to resource R_i.

Suppose τ_3 wants to increase its rate at time $t = 6$: The compression algorithm computes a new period equal to 14 for task τ_2. Fig. 6 shows what happens: At time t, task τ_2 is inside the critical section on resource R_1; if its period is increased immediately, the current instance of τ_3 cannot make preemption, so task τ_3 is blocked twice: by τ_4 at time $t = 1$ and by τ_2 at time $t = 6$. Note that, when τ_3 is blocked by τ_2, task τ_1 contributes to the blocking time of τ_3, whose execution is further delayed. Hence, performing a compression when a task is using a shared resource may break a fundamental property of SRP according to which each task can be blocked at most for the duration of one critical section. However, Fig. 7 shows that, if task τ_2 increases its period at time $t = 7$, after exiting the critical section, and task τ_3 decreases its period at the next release time ($t = 15$), the SRP properties are preserved. This example helps to better understand that, in the presence of resource constraints, task compression has to be performed according to the following rule.

General compression rule. *Whenever a period change is required at time t, the compression algorithm can be executed only when the system ceiling is equal to zero (i.e., when no task is inside a critical section). Let t_0 be such a time and let δ_{max} be the time after t_0 defined in (23). At this time, if the new configuration is feasible, the new periods, preemption levels, and resource ceilings can be computed. Then,*

- *the compressed tasks can immediately increase their period at time t_0,*
- *a new task can be activated at time δ_{max}, and*
- *the periods of the tasks increasing their rate can be changed at their next release time after time δ_{max}.*

Some considerations must be done to show that nothing wrong will happen using the previous rule:

- Since, at time t_0, the system ceiling is zero, no task is inside a critical section, hence, all instances active at t_0 cannot be blocked until completion. It follows that preemption levels can be safely changed without invalidating the SRP properties.
- Since, according to the previous rule, no new tasks can be activated before δ_{max} and active tasks do not increase their utilization, the system remains schedulable in $[t_0, \delta_{max}]$.
- From time δ_{max} on, the utilization of the new task set (including the newly arrived tasks) can be computed by (19) and guaranteed by Theorem 2.

As a final remark, we notice that the worst-case delay between the time at which a compression is required and the time at which the ceiling becomes zero is bounded by $max_i(T_i)$, the longest period in the system. In fact, if the system is feasible, each task must exit every critical section before its deadline. In the worst case, we have to wait for the task with the longest period to exit its outer critical section.

6 Experimental Results

The elastic task model has been implemented on top of the HARTIK kernel [5], [11] to perform some experiments on multimedia applications and verify the results predicted by the theory. In particular, the elastic guarantee mechanism has been implemented as a high priority task, the QoS manager, activated by the other tasks when they are created or when they want to change their period. Whenever activated, the QoS manager calculates the new periods and

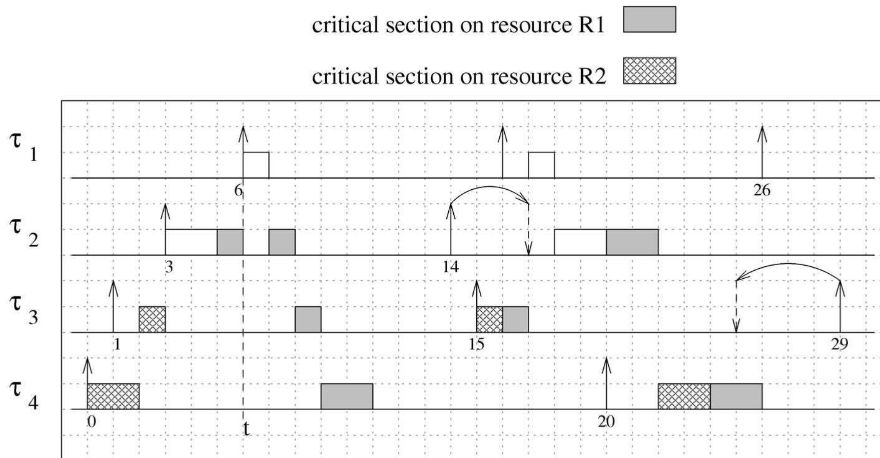

Fig. 6. Example of tasks' compression with resource constraints.

Authorized licensed use limited to: UNIVERSITA PISA S ANNA. Downloaded on October 7, 2008 at 6:19 from IEEE Xplore. Restrictions apply.

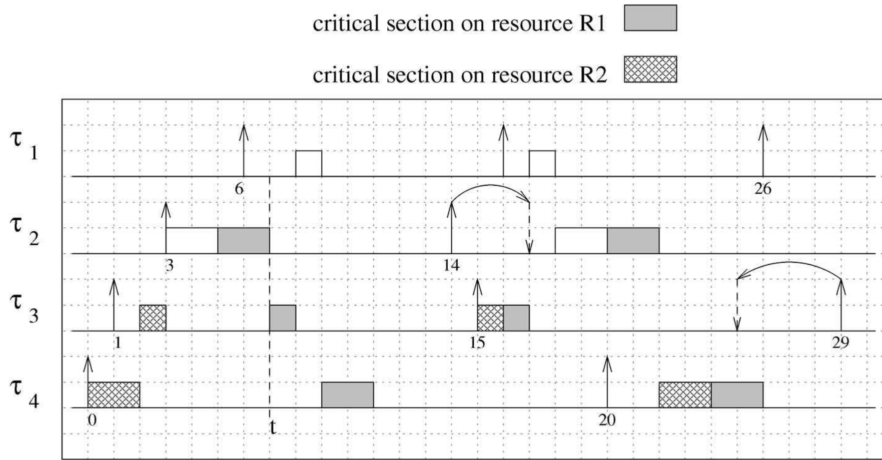

Fig. 7. Example of tasks' compression with resource constraints.

changes them atomically. According to the results presented in Section 5, periods are changed at the next release time of the task whose period is decreased. If more tasks ask to decrease their period, the QoS manager will change them, if possible, at their next release time.

In the first experiment, four periodic tasks are created at time $t = 0$. The tasks' parameters are shown in Table 4, while the actual number of instances executed by each task as a function of time is shown in Fig. 8. All the tasks start executing at their nominal period and, at time $t_1 = 10sec$, τ_1 decreases its period to $T'_1 = 33msec$. We recall that a task cannot decrease its period by itself, but must perform a request to the QoS manager, which checks the feasibility of the request and calculates the new periods for all the tasks in the system. So, at time t_1, since the schedule is found to be feasible, the period of τ_1 is decreased and the periods of τ_2, τ_3, and τ_4 are increased according to their elastic coefficients. The values of all the periods are indicated in the graph.

At time $t_2 = 20sec$, τ_1 returns to its nominal period, so the QoS manager can change the periods of the other tasks to their initial values, as shown in the graph. In this manner, the QoS manager ensures that, when a task needs to change its period, the task set remains schedulable and the variation of each task period can be controlled by the elastic factor.

In the second experiment, we tested the elastic model as an admission control policy. Three tasks start executing at time $t = 0$ at their nominal period, while a fourth task starts at time $t_1 = 10sec$. The tasks' parameters are shown in Table 5. When τ_4 is started, the task set is not schedulable with the current periods, thus the QoS manager, in order to accommodate the request of τ_4, increases the periods of the other tasks according to the elastic model. The actual execution rates of the tasks are shown in Fig. 9. Notice that, although the first three tasks have the same elastic coefficients, their periods are changed by a different amount because tasks have different utilization factors.

7 Conclusions

In this paper, we presented a flexible scheduling theory in which periodic tasks are treated as springs, with given elastic coefficients. Under this framework, periodic tasks can intentionally change their execution rate to provide different quality of service and the other tasks can automatically adapt their periods to keep the system underloaded. The proposed model can also be used to handle overload situations in a more flexible way. In fact, whenever a new task cannot be guaranteed by the system, instead of rejecting the task, the system can try to reduce the utilizations of the other tasks (by increasing their periods in a controlled fashion) to decrease the total load and accommodate the new request. As soon as a transient overload condition is over (because a task terminates or voluntarily increases its period), all the compressed tasks may expand up to their original utilization, eventually recovering their nominal periods.

The major advantage of the proposed method is that the policy for selecting a solution is implicitly encoded in the elastic coefficients provided by the user. Each task is varied based on its current elastic status and a feasible configuration is found, if one exists.

TABLE 4
Task Set Parameters Used for the First Experiment

Task	C_i	T_{i_0}	$T_{i_{min}}$	$T_{i_{max}}$	E_i
τ_1	24	100	30	500	1
τ_2	24	100	30	500	1
τ_3	24	100	30	500	1.5
τ_4	24	100	30	500	2

Periods and computation times are expressed in milliseconds.

TABLE 5
Task Set Parameters Used for the Second Experiment

Task	C_i	T_{i_0}	$T_{i_{min}}$	$T_{i_{max}}$	E_i
τ_1	30	100	30	500	1
τ_2	60	200	30	500	1
τ_3	90	300	30	500	1
τ_4	24	50	30	500	1

Periods and computation times are expressed in milliseconds.

Authorized licensed use limited to: UNIVERSITA PISA S ANNA. Downloaded on October 7, 2008 at 6:19 from IEEE Xplore. Restrictions apply.

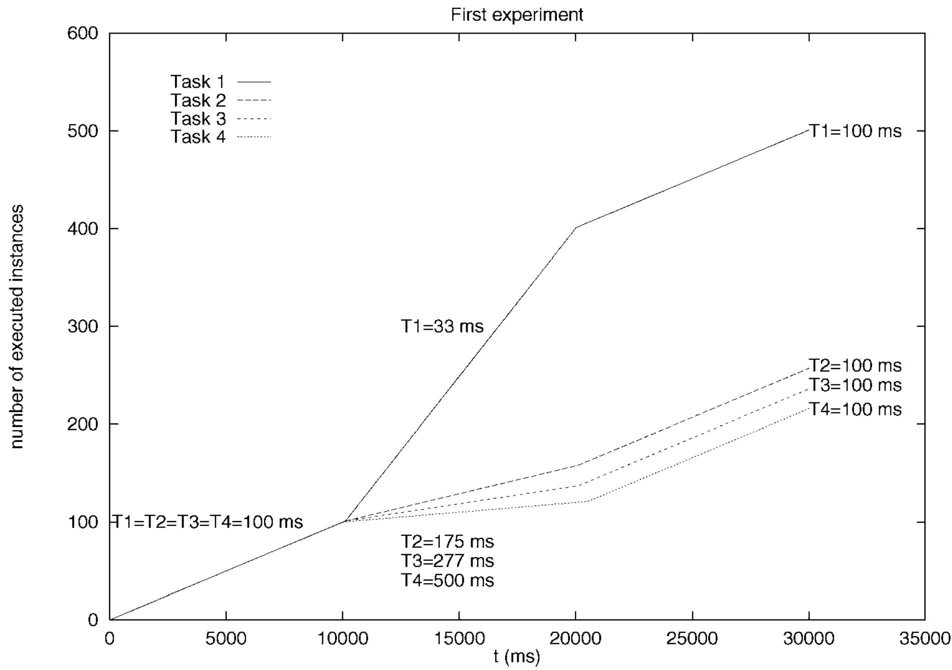

Fig. 8. Dynamic period change.

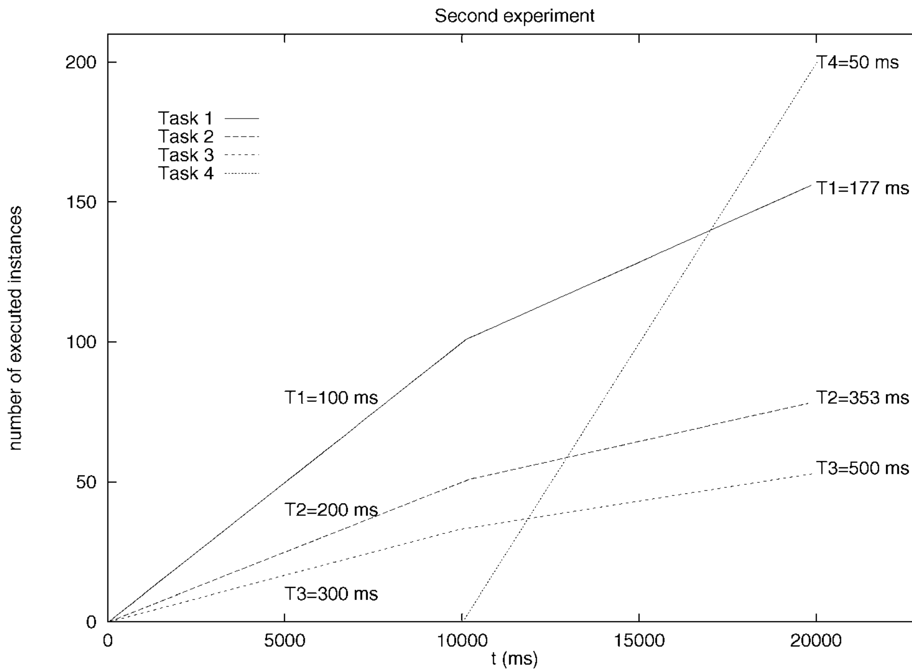

Fig. 9. Dynamic task activation.

The elastic model is extremely useful for supporting both multimedia systems and control applications in which the execution rates of some computational activities have to be dynamically tuned as a function of the current system state. Furthermore, the elastic mechanism can easily be implemented on top of classical real-time kernels and can be used under fixed or dynamic priority scheduling algorithms. The experimental results shown in this paper have been conducted by implementing the elastic mechanism on the HARTIK kernel [5], [11].

As future work, we are investigating the possibility of using the elastic scheduling methodology with an execution time estimator embedded in the kernel for developing an adaptive mechanism which automatically allocates the requested bandwidth to each task without forcing the user to specify worst-case execution times. In this way, the user would specify, for each task, only a desired activation rate and an importance value (related to the elastic coefficient), while the system would work to satisfy the specification as closely as possible, using the importance values as metrics.

Authorized licensed use limited to: UNIVERSITA PISA S ANNA. Downloaded on October 7, 2008 at 6:19 from IEEE Xplore. Restrictions apply.

REFERENCES

[1] T.F. Abdelzaher, E.M. Atkins, and K.G. Shin, "QoS Negotiation in Real-Time Systems and Its Applications to Automated Flight Control," *Proc. IEEE Real-Time Technology and Applications Symp.,* June 1997.

[2] T.P. Baker, "Stack-Based Scheduling of Real-Time Processes," *J. Real-Time Systems,* vol. 3, no. 1, pp. 67-100, 1991.

[3] S.K. Baruah, R.R. Howell, and L.E. Rosier, "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic Real-Time Tasks on One Processor," *J. Real-Time Systems,* vol. 2, 1990.

[4] G. Beccari, S. Caselli, M. Reggiani, and F. Zanichelli, "Rate Modulation of Soft Real-Time Tasks in Autonomous Robot Control Systems," *IEEE Proc. 11th Euromicro Conf. Real-Time Systems,* June 1999.

[5] G. Buttazzo, "HARTIK: A Real-Time Kernel for Robotics Applications," *Proc. 14th IEEE Real-Time Systems Symp.,* pp. 201-205, Dec. 1993.

[6] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications,* pp. 102-103. Boston: Kluwer Academic, 1997.

[7] G. Buttazzo, G. Lipari, and L. Abeni, "Elastic Task Model for Adaptive Rate Control," *Proc. 19th IEEE Real-Time Systems Symp.,* Dec. 1998.

[8] M.L. Dertouzos, "Control Robotics: The Procedural Control of Physical Processes," *Information Processing,* vol. 74, 1974.

[9] K. Jeffay and D.L. Stone, "Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems," *Proc. IEEE Real-Time Systems Symp.,* Dec. 1993.

[10] T.-W. Kuo and A.K. Mok, "Load Adjustment in Adaptive Real-Time Systems," *Proc. 12th IEEE Real-Time Systems Symp.,* Dec. 1991.

[11] G. Lamastra, G. Lipari, G. Buttazzo, A. Casile, and F. Conticelli, "HARTIK 3.0: A Portable System for Developing Real-Time Applications," *Proc. IEEE Real-Time Computing Systems and Applications,* Oct. 1997.

[12] C. Lee, R. Rajkumar, and C. Mercer, "Experiences with Processor Reservation and Dynamic QOS in Real-Time Mach," *Proc. Multimedia Japan 96,* Apr. 1996.

[13] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM,* vol. 20, no. 1, pp. 40-61, 1973.

[14] C.W. Mercer, S. Savage, and H. Tokuda, "Processor Capacity Reserves for Multimedia Operating Systems," *Proc. IEEE Int'l Conf. Multimedia Computing and Systems,* May 1994.

[15] A.K. Mok and D. Chen, "A Multiframe Model for Real-Time Tasks," *Proc. IEEE Real-Time System Symp.,* Dec. 1996.

[16] T. Nakajima, "Dynamic QOS Control and Resource Reservation," *Proc. IEICE Real-Time Processing (RTP '98),* 1998.

[17] T. Nakajima, "Resource Reservation for Adaptive QOS Mapping in Real-Time Mach," *Proc. Sixth Int'l Workshop Parallel and Distributed Real-Time Systems,* Apr. 1998.

[18] T. Nakajima and H. Tezuka, "A Continuous Media Application Supporting Dynamic QOS Control on Real-Time Mach," *Proc. ACM Multimedia '94,* 1994.

[19] D. Seto, J.P. Lehoczky, L. Sha, and K.G. Shin, "On Task Schedulability in Real-Time Control Systems," *Proc. IEEE Real-Time Systems Symp.,* Dec. 1997.

[20] M. Spuri and G.C. Buttazzo, "Efficient Aperiodic Service under Earliest Deadline Scheduling," *Proc. IEEE Real-Time System Symp.,* Dec. 1994.

[21] M. Spuri, G.C. Buttazzo, and F. Sensini, "Robust Aperiodic Scheduling under Dynamic Priority Systems," *Proc. IEEE Real-Time Systems Symp.,* Dec. 1995.

[22] M. Spuri and G.C. Buttazzo, "Scheduling Aperiodic Tasks in Dynamic Priority Systems," *Real-Time Systems,* vol. 10, no. 2, 1996.

[23] J. Sun and J.W.S. Liu, "Bounding Completion Times of Jobs with Arbitrary Release Times and Variable Execution Times," *Proc. IEEE Real-Time Systems Symp.,* Dec. 1996.

[24] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J.W.-S. Liu, "Probabilistic Performance Guarantee for Real-Time Tasks with Varying Computation Times," *Proc. IEEE Real-Time Technology and Applications Symp.,* Jan. 1995.

Giorgio C. Buttazzo graduated in electronic engineering from the University of Pisa in 1985, received the Master's degree in computer science from the University of Pennsylvania in 1987, and the PhD degree in computer engineering from the Scuola Superiore S. Anna of Pisa in 1991. He is an associate professor of computer engineering at the University of Pavia, Italy. His main research interests include real-time operating systems, dynamic scheduling algorithms, quality of service control, multimedia systems, advanced robotics applications, and neural networks. He is a member of the IEEE and the IEEE Computer Society.

Giuseppe Lipari graduated in computer engineering from the University of Pisa in 1996 and received the PhD degree in computer engineering from the Scuola Superiore S. Anna in 2000. He is an assistant professor of computer engineering at the Scuola Superiore S. Anna. During 1999, he was a visiting student at the University of Vermont and at the University of North Carolina at Chapel Hill, working with Professor S. Baruah and Professor K. Jeffay on novel scheduling algorithms for resource reservation on real-time systems. His main research interests include real-time operating systems, scheduling algorithm for soft real-time systems, and component-based real-time kernels for embedded systems.

Marco Caccamo is a PhD student in computer engineering at the Scuola Superiore S. Anna of Pisa, Italy. In 1997, he graduated in computer engineering from the University of Pisa. His reasearch activity is focused on the development and analysis of flexible scheduling algorithms for real-time systems. His reasearch interests include real-time operating systems, scheduling algorithms, aperiodic service mechanisms, fault-tolerant systems, quality of service control, and multimedia applications.

Luca Abeni is a PhD student in computer engineering at the Scuola Superiore S. Anna of Pisa, Italy. He graduated in computer engineering from the University of Pisa in 1998. During 2000, he was a visiting student at Carnegie Mellon University working with Professor Ragunathan Rajkumar on resource reservation algorithms for real-time kernels. His main research interests include real-time operating systems, scheduling algorithms, quality of service management, and multimedia applications.

▷ **For more information on this or any computing topic, please visit our Digital Library at** http://computer.org/publications/dlib.

Authorized licensed use limited to: UNIVERSITA PISA S ANNA. Downloaded on October 7, 2008 at 6:19 from IEEE Xplore. Restrictions apply.