ELSEVIER

Journal of Systems Architecture 46 (2000) 327–338

JOURNAL OF SYSTEMS ARCHITECTURE

www.elsevier.com/locate/sysarc

Schedulability analysis of periodic and aperiodic tasks with resource constraints

Giuseppe Lipari *, Giorgio Buttazzo

RETIS Laboratory, Scuola Superiore S. Anna, Via Carducci 40, I-56100 Pisa, Italy

Abstract

In this paper, we address the problem of scheduling hybrid task sets consisting of hard periodic and soft aperiodic tasks that may share resources in exclusive mode in a dynamic environment, where tasks are scheduled based on their deadlines. Bounded blocking on exclusive resources is achieved by means of a dynamic resource access protocol which also prevents deadlocks and chained blocking. Aperiodic responsiveness is enhanced by an efficient servicing technique which assigns each aperiodic request a suitable deadline. Feasibility conditions are extended to handle tasks with deadlines different from periods and a reclaiming technique is presented to deal with early completions. © 2000 Published by Elsevier Science B.V. All rights reserved.

Keywords: Real-time; Scheduling; Resource constraints; Soft aperiodic task; Stack resource policy

1. Introduction

Many real-world control applications are characterized by tasks with different timing characteristics. For example, time-critical control activities should be implemented as *hard* periodic tasks and guaranteed in worst-case conditions. Less critical activities (*soft* tasks) do not need to be guaranteed; however, they should be executed as soon as possible without jeopardizing the guarantee achieved on hard tasks. If soft activities interact with hard tasks through shared resources, such an interaction has to be taken into account in the schedulability analysis. The problem becomes more complex if periodic tasks are permitted to have deadlines different from their periods.

In the real-time literature, several algorithms have been proposed for dealing with specific subproblems, but, from our point of view, not sufficient work has been done for integrating tasks with different types of constraints, especially in dynamic environments.

Liu and Layland [15] found important results for the schedulability of periodic task sets, but under the assumption that tasks are independent. After that first fundamental contribution, much work has been done for dealing with periodic tasks with deadlines less than periods [1,26,3]. In particular, the ''processor demand approach'', proposed in Ref. [3] for the feasibility analysis of a sporadic task set, is general and can be easily extended to consider other types of constraints, as we do in this paper.

Concurrency control protocols for handling resource constraints have been proposed under the fixed priority systems [17] and dynamic priority systems [5,2,10], but without considering the presence of aperiodic tasks.

On the other hand, several aperiodic service mechanisms have been proposed under RM

* Corresponding author. E-mail: giorgio@sssup.it

1383-7621/00/$ - see front matter
PII: S1383-7621(99)00009-0

[12,13,1,26] and under EDF [20,9,22,23], but no resource constraints have been taken into account. In Ref. [19], a schedulability test is proposed for task sets consisting of hard periodic and sporadic tasks.

The problem of integrating resource and timing constraints has been addressed in Ref. [7] using an off-line scheduling approach. In this context, many types of constraints can be considered in a pre-runtime analysis, and an optimal (or suboptimal) solution can be found for the scheduling problem. This method is very powerful, but it is not flexible enough to be applied in dynamic environments, where task arrivals are not known in advance. Fohler [8] proposed a hybrid method for integrating on-line service of aperiodic requests with an off-line schedule of periodic tasks, but hard and soft tasks cannot share exclusive resources. The same problem has also been considered by a heuristic approach in the Spring kernel [18], but no optimal solutions can be found.

In this work, we address the problem of scheduling hard periodic and soft aperiodic tasks that may share exclusive resources in a dynamic environment, where tasks are scheduled based on their deadlines. The analysis is performed to consider periodic tasks with relative deadlines different from their periods. This problem is not trivial if we want to schedule aperiodic tasks as soon as possible.

The paper is organized as follows. Section 2 introduces the terminology and the assumptions used throughout the paper. Section 3 describes the concurrency control protocol used to access shared resources. Section 4 analyzes the general problem of scheduling hybrid (hard periodic and soft aperiodic) task sets and illustrates our integrated approach. Section 5 reports the extension for the case of deadlines different from periods and Section 6 presents our conclusions.

2. Assumptions and terminology

We consider a set $\mathscr{R}$ of r resources, a set $\mathscr{T}^{\mathscr{P}}$ of n hard periodic tasks, and a set $\mathscr{T}^{\mathscr{A}}$ of m soft aperiodic tasks, that have to be executed on a uniprocessor system. Tasks are preemptable and all the resources are accessed in exclusive mode using critical sections. To simplify our presentation, only single-unit resources are considered. However, the results can easily be extended to deal with multi-unit resources, as described in Ref. [2].

Now we state our terminology and notation used throughout the paper.

- Any task τ_i (periodic and aperiodic) is an infinite sequence of instances $J_{i,k}$, called jobs; every job has a worst-case execution time C_i, and a release time $r_{i,k}$.
- Any task τ_i is allowed to access shared resources through critical sections: a *critical section* ξ_{ih} is a 3-ple $(\rho_{ih}, \phi_{ih}, \gamma_{ih})$, where
  $\rho_{ih} \in \mathscr{R}$ is the resource accessed by the critical section;
  ϕ_{ih} is the earliest time, relative to the release time $r_{i,k}$, that τ_i can enter ρ_{ih};
  γ_{ih} is the worst-case execution time of the critical section.
- Each instance of a hard task τ_i has a relative deadline D_i; thus, its absolute deadline is given by
  $$d_{i,k} = r_{i,k} + D_i.$$
- A task τ_i is periodic if
  $$r_{i,k} = r_{i,1} + (k-1)T_i,$$
  where $r_{i,1}$ is its initial phase and T_i is its period.
- A task τ_i is aperiodic if $r_{i,k+1} > r_{i,k}$, and is sporadic if
  $$r_{i,k+1} \geqslant r_{i,k} + T_i,$$
  where $r_{i,1} \geqslant 0$, and T_i is the minimum interarrival time between two consecutive instances.

Summarizing, a hard periodic (or sporadic) task is described as

$$\tau_i = (C_i, \{\xi_{ih}\}, D_i, T_i),$$

whereas a soft aperiodic task is described as

$$\tau_j = (C_j, \{\xi_{jh}\}).$$

The task model presented here is slightly different from the classical model. In order to carry out the feasibility analysis under resource constraints, we need to define each critical section used by a task for accessing a resource. In particular, we need to know the *position* and the *duration* of each critical section.

Note that, in our model, critical sections can be properly nested in an arbitrary fashion. In other words, it is not necessary to access resources in a particular order, like in other concurrency control protocols.

To simplify the discussion, in many of the theorems presented in this paper, we use the concept of fully utilized interval which is defined as follows.

Definition 1. An interval $[t_1, t_2]$ is *fully utilized* if and only if there does not exist a time $t \in (t_1, t_2)$ such that all the active tasks complete their execution at t.

Throughout the paper, we assume that all tasks are scheduled based on their deadlines, according to the Earliest Deadline First (EDF) algorithm. Moreover, periodic tasks are synchronous (that is, their initial phases are equal to zero). Finally, to simplify the presentation of the analysis, we neglect the overhead introduced by the operating system.

3. Accounting for shared resources

To perform schedulability analysis under resource constraints in a dynamic priority environment, different resource access protocols have been proposed to bound the worst-case blocking of tasks due to mutual exclusion. The most important are the Dynamic Priority Ceiling (DPC) protocol [5], the Stack Resource Policy (SRP) [2] and the Dynamic Deadline Modification (DDM) protocol [10]. Although we base our discussion on SRP, which has been selected for its simplicity and generality, the feasibility analysis presented here is valid under any of these protocols.

In the remainder of this section, we briefly describe the SRP and then we extend the feasibility test proposed by Baker [2] by considering the position and the duration of each individual critical section. Moreover, we provide an algorithm for calculating the blocking time of each task. The resulting feasibility test is necessary and sufficient, although pseudo-polynomial in complexity. We finally compare the SRP and the DDM and show that they are equivalent from a scheduling point of view.

3.1. The stack resource policy

According to this protocol, every hard (periodic and sporadic) task τ_i is assigned a dynamic priority p_i based on EDF and a static preemption level π_i, such that the following essential 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 periodic task τ_i is assigned the following preemption level

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

In addition, every resource R_k is assigned a static [1] *ceiling* defined as

$$\mathrm{ceil}(R_k) = \max_i \{\pi_i \mid \tau_i \text{ needs } R_k\}.$$

Moreover, a dynamic *system ceiling* is defined as

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

Then, the SRP scheduling rule states that

> a job is not allowed to start executing until its priority is the highest among the active jobs and its preemption level is greater than the system ceiling.

The SRP ensures that once a task is started, it will never block until completion; it can only be preempted by higher priority tasks.

This protocol has several interesting properties. For example, it applies to both static and dynamic scheduling algorithms, prevents deadlocks, bounds the maximum blocking times of tasks, reduces the number of context switches, can be easily extended to multi-unit resources, allows tasks to share

[1] In the case of multi-units resources, the ceiling of each resource is dynamic as it depends on the number of units actually free.

stack-based resources, and its implementation is straightforward.

Under the SRP, there is no need to implement waiting queues. In fact, a task never blocks its execution: it simply cannot start executing if its preemption level is not high enough.

As a consequence, the blocking time B_i considered in the schedulability analysis refers to the time for which task τ_i is kept in the ready queue by the preemption test. Although the task never blocks, B_i is considered as a "blocking time" because it is caused by tasks having lower preemption levels.

Assuming relative deadlines equal to periods, the maximum blocking time for each task τ_i can be calculated as the longest critical section among those with a ceiling greater than or equal to the preemption level of τ_i

$$B_i = \max\{\gamma_{jh} \mid (T_i < T_j - \phi_{jh}) \wedge \pi_i \leqslant \mathrm{ceil}(\rho_{jh})\}. \tag{1}$$

The meaning of condition $T_i < T_j - \phi_{jh}$ is shown in Fig. 1. If T_i is greater than $T_j - \phi_{jh}$, it is not possible for task τ_i to preempt task τ_j when it is inside critical section ξ_h. In fact, if τ_i arrives after τ_j entered the critical section, then $d_i > d_j$, so it cannot preempt τ_j.

Given these definitions, in the Baker's original paper (when only periodic and sporadic tasks are considered), the feasibility of a task set with resource constraints is tested by the following sufficient condition

$$\forall i, \quad 1 \leqslant i \leqslant n, \quad \sum_{k=1}^{i} \frac{C_k}{T_k} + \frac{B_i}{T_i} \leqslant 1, \tag{2}$$

which assumes that all the tasks are sorted by decreasing preemption levels, so that $\pi_i \geqslant \pi_j$ only if $i < j$.

The following theorem extends this analysis by providing a tighter schedulability test.

Theorem 1. *Let $\mathscr{T}^{\mathscr{P}}$ be a set of n hard sporadic tasks ordered by decreasing preemption level (so that $\pi_i \geqslant \pi_j$ only if $i < j$), such that $U_p = \sum_{i=1}^{n} \frac{C_n}{T_n} \leqslant 1$. Then, $\mathscr{T}^{\mathscr{P}}$ is schedulable by EDF + SRP if and only if*

$$\forall j, \quad 1 \leqslant j \leqslant n \; \forall L, \quad 0 < L \leqslant T_j, \quad L \geqslant \sum_{i=1}^{j} \left\lfloor \frac{L}{T_i} \right\rfloor C_i + \max(0, B_j - 1). \tag{3}$$

Proof. *Only if*. Suppose that Eq. (3) is not verified for $j = \bar{j}$ and $L = \bar{L}$ and that task set $\mathscr{T}^{\mathscr{P}}$ is still schedulable.

If $B_{\bar{j}} \leqslant 1$, then suppose that all the tasks start at time 0: the demand of processor utilization in $[0, \bar{L}]$ is greater than $\bar{L}$ and the task set cannot be schedulable.

If $B_{\bar{j}} > 1$, then, for construction of $B_{\bar{j}}$, exist τ_l and h such that $\gamma_{lh} = B_{\bar{j}}$. Suppose that all the tasks τ_i, with $i \leqslant \bar{j}$, start at time x and that task τ_l starts at time $(x - \phi_{lh} - 1)$: the situation is shown in Fig. 2.

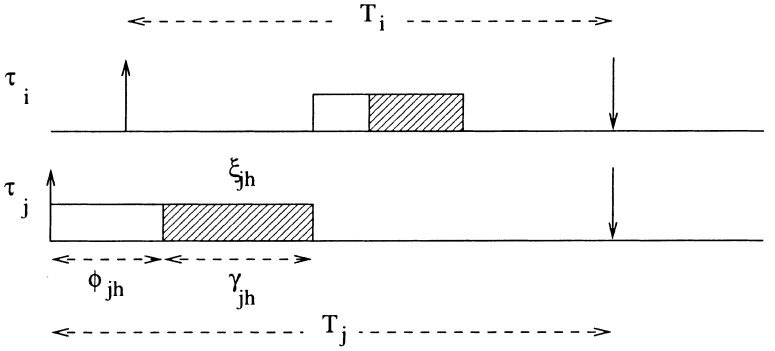

Fig. 1. γ_{jh} does not contribute to B_i.

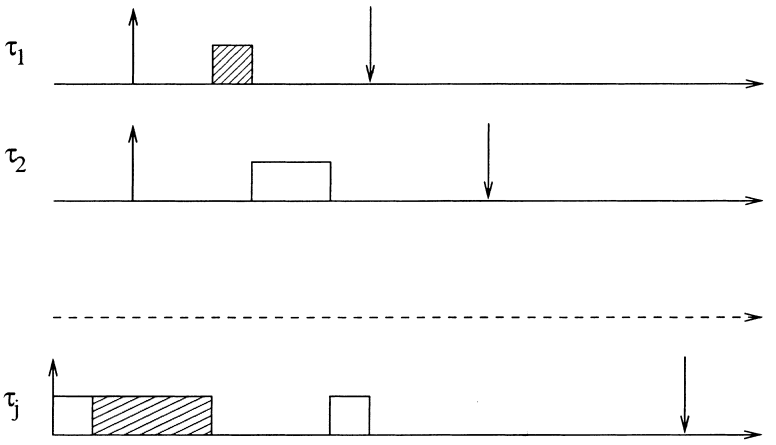

Fig. 2. How a low priority task can delay high priority tasks.

The processor demand in interval $[x, x+\overline{L}]$ is

$$\sum_{i=1}^{\bar{j}} \left\lfloor \frac{\overline{L}}{T_i} \right\rfloor C_i + B_{\bar{j}} > \overline{L}.$$

Hence, the system is not feasible.

If. Suppose that Eq. (3) is verified and that the task set is not schedulable. Hence, there is a deadline miss at time y. Let x be the last instant of time such that there are no pending requests of tasks arrived before x and with deadline not after y. Let $\mathscr{A}$ be the set of tasks that execute in $[x, y]$ and with deadline not after y, and let $\mathscr{B}$ be the set of tasks that execute in $[x, y]$ and with deadline after y. Notice that tasks in $\mathscr{B}$ are inside a critical section at time t_1. In fact, they begin executing and enter the critical section before x; they continue their execution until the end of the critical sections, and then they can be preempted. We now show that $\mathscr{B}$ contains at most one task. Suppose that $\mathscr{B}$ contains two tasks, τ_a, τ_b: both of these must hold resources at time x. Since they block jobs in $\mathscr{A}$, they both hold resources with current ceilings higher than π_i for some $i \in A$. However, a task cannot wait for more than one critical section. Hence, only one task can be in $\mathscr{B}$. If $(y-x) \geqslant T_n$, then $\mathscr{B}$ is empty and, following the same argument as in [15], it follows that $U_p > 1$, which contradicts the hypothesis. If $(y-x) < T_n$, then let τ_j be the task with the lowest preemption level in $\mathscr{A}$. Then,

$$y - x < C_{\mathscr{A}} + C_{\mathscr{B}} \leqslant \sum_{i=1}^{j} \left\lfloor \frac{y-x}{T_i} \right\rfloor C_i + \max(0, B_j - 1).$$

This is a contradiction, since there exists an $L = y - x$ that does not satisfy Eq. (3). Note that $L < T_h$ with $\tau_h \in B$, and hence $L < T_n$. □

It is worth making some consideration on the proposed feasibility test. First of all, condition (3) is necessary and sufficient only for sporadic tasks. In fact, for periodic tasks, the problem of deciding feasibility in the presence of resource constraints is NP-hard [10]. In addition, since condition (3) is necessary and sufficient, the SRP is optimal; that is, every feasible task set can be scheduled by EDF with the SRP.

The complexity of the proposed schedulability test is pseudo-polynomial. As a consequence, for large task sets, it might be too costly for providing on-line guarantee. Rather, this method can be used off-line to guarantee the schedulability of all critical periodic and sporadic tasks in the presence of resource constraints.

3.2. Comparison between SRP and DDM

The problem of scheduling sets of sporadic tasks that share resources was also considered by Jeffay in [10]. He considers a task as a sequence of phases. In each phase, a task can only execute

normally or entering a critical section at the beginning of the phase and leave it at the end of the phase. His algorithm, called DDM, is essentially based on EDF: when a task is inside a critical section, it dynamically modifies the tasks' deadlines in order not to allow preemption by other tasks that share that resource. A necessary and sufficient condition is given and the algorithm is optimal.

The model considered in Ref. [10], however, does not take nested critical sections into account. This is due to the fact that it is not possible (or rather complicated) to express such constraints with the phase model. Nevertheless, SRP and EDF/DDM are equivalent (when we consider no nesting), in the sense that given a task set, they produce the same schedule. This can be seen with the following considerations:

- the rule for dynamically modifying deadlines in DDM is analogous to the use of preemption levels and system ceiling in SRP;
- EDF/DDM and SRP use the same parameters in different ways, and with a different notation;
- both algorithms are optimal.

In addition, our model has the following advantages with respect to DDM:

- it can express nested critical sections;
- SRP is very easy to implement with low overhead;
- SRP can be applied to both dynamic priority and fixed priority scheduling algorithm (however, for fixed priority algorithm, the schedulability test is different and it is not optimal).

4. Sharing resources between soft and hard tasks

When soft aperiodic tasks share resources with periodic tasks, the duration of their critical sections must be taken into account in the feasibility analysis, even if the aperiodic requests are scheduled in the background. The blocking factor introduced by aperiodic tasks depends on the duration of critical sections and on the policy adopted for scheduling aperiodic jobs.

In order to bound the maximum blocking time of each task and analyze the schedulability of hybrid task sets, we assume that the SRP is used to access shared resource. Hence, each aperiodic request must be assigned a suitable preemption level according to the SRP protocol.

The simplest way to service aperiodic requests is to schedule them in the background: soft aperiodic tasks are served FIFO and with the lowest priority in the system. For our purpose, we assign each aperiodic request a deadline equal to infinity and a preemption level equal to 0. With this method, an aperiodic task can never preempt a hard periodic task, and Property 1 is verified, so guaranteeing the correctness of the protocol. To perform the schedulability test, the critical sections of soft aperiodic tasks have to be considered in the blocking factor defined in Eq. (1), just as done for periodic tasks. Using background service, however, aperiodic tasks can experience high response times and it is not easy to calculate their worst case completion time.

A better approach that can be followed in dynamic environments is to assign suitable deadlines to aperiodic requests, and schedule them as hard tasks. An efficient method for assigning deadlines to soft aperiodic requests has been proposed by Spuri and Buttazzo in Ref. [23]. In this work the authors describe an algorithm, the Total Bandwidth Server (TBS) for enhancing the response time of soft aperiodic tasks in hard real-time environments. Although not optimal, this approach has three great advantages: it exhibits a good performance/cost ratio, it can easily be extended to deal with tasks with firm deadlines [22], and it allows aperiodic tasks to share resources with hard periodic tasks, as described below.

In the rest of this section, we first show a method for assigning "safe" deadlines to aperiodic tasks, so that the whole hybrid task set can be guaranteed even under resource constraints. We then illustrate how to reclaim processor utilization for enhancing aperiodic response time in the case of early completions. Finally, we show that capacity- based servers are not suited in the presence of resource constraints.

4.1. Assigning deadlines to soft tasks

Each time an aperiodic request enters the system, a "fictitious" deadline is assigned to it based

on the available processor bandwidth. [2] In particular, when the kth aperiodic request arrives at time $t = r_k$, it receives an *eligible time*

$$e_k = \max(r_k, d_{k-1}), \tag{4}$$

where d_{k-1} is the deadline assigned to the previous aperiodic job. By definition, $d_0 = 0$. At time e_k, the aperiodic job J_k enters the ready queue and receives the following deadline

$$d_k = e_k + \frac{C_k}{U_s}, \tag{5}$$

where C_k is the worst-case execution time of the request and U_s is the server utilization factor (i.e., its bandwidth), whose value is provided at design stage. Thus, at time r_k, request J_k is temporarily inserted in a queue of pending jobs and, at time e_k, it is inserted into the ready queue together with the other hard tasks.

Note that the bandwidth already assigned to previous requests is taken into account by simply considering the maximum between r_k and d_{k-1}. The following theorem states an important result for the TBS.

Theorem 2. *In every interval of time $[t_1, t_2]$, the total computational demand of aperiodic tasks scheduled by a TBS with utilization U_s never exceeds $(t_2 - t_1)U_s$.*

Based on this result, proved in Ref. [23], a set of periodic tasks with utilization factor $U_p = \sum_{i=1}^{n}(C_i/T_i)$ and a TBS with a bandwidth U_s is schedulable by EDF (without resource constraints) if and only if

$$U_p + U_s \leqslant 1. \tag{6}$$

4.2. Integration

To allow soft aperiodic tasks to be scheduled with hard tasks under the SRP protocol, each aperiodic request must be assigned the following preemption level

$$\pi_k = \frac{U_s}{C_k}. \tag{7}$$

Notice that $(C_k/U_s) = [d_k - e_k]$ is the interval between the time e_k at which the aperiodic request becomes eligible to execute and the absolute deadline assigned to it by the TBS. Since this is equivalent to a relative deadline, the preemption level defined by Eq. (7) is consistent with Property 1.

Once a preemption level is assigned to each aperiodic task, and a ceiling is associated with each resource, the maximum blocking time of periodic task τ_i can still be calculated using Eq. (1)

$$B_i = \max\{\gamma_{j,h} \mid T_i < T_j - \phi_{jh} \ \wedge \ \pi_i \leqslant \mathrm{ceil}(\rho_{jh})\},$$

where now j ranges over the whole task set, including both periodic and aperiodic tasks, and, if τ_j is an aperiodic task, $T_j = (C_j/U_s)$. Then, the schedulability of the hybrid task set can be guaranteed based on the following theorem.

Theorem 3. *Let $\mathscr{T}^{\mathscr{P}}$ be a set of n hard periodic tasks ordered by decreasing preemption level (so that $\pi_i \geqslant \pi_j$ only if $i < j$), and let $\mathscr{T}^{\mathscr{A}}$ be a set of aperiodic tasks scheduled by a TBS with utilization U_s, such that $U_p + U_s \leqslant 1$. Also, let π^* be the maximum preemption level among those assigned to aperiodic tasks. Then, $\mathscr{T}^{\mathscr{P}}$ is schedulable by EDF + SRP + TBS if*

$$\forall i, \quad 1 \leqslant i \leqslant n \ \forall L, \quad 0 < L < T_i$$

$$L \geqslant \sum_{j=1}^{i} \left\lfloor \frac{L}{T_j} \right\rfloor C_j + \max\{0, B_i - 1\} + S(i)U_s L$$

where $S(i)$ is a select function defined as

$$S(i) = \begin{cases} 0 & \text{if } \pi_i \geqslant \pi^*, \\ 1 & \text{if } \pi_i < \pi^*. \end{cases} \tag{8}$$

Proof. The proof is a straightforward extension of the proof of Theorem 1 and is not reported here. It can be found in Ref. [16]. □

4.3. Reclaiming processor utilization

Using TBS, any unused computation time can easily be reclaimed to enhance responsiveness of

[2] The formal definition of the TBS presented here is slightly different from the original formulation.

aperiodic tasks. This is a very useful feature since, in practice, tasks often execute less than their worst case computation time. To reclaim spare time is sufficient to advance the aperiodic fictitious deadline by an opportune value. We distinguish the case of early completion of periodic and aperiodic tasks. The former case is considered by the following Lemma.

Lemma 4. *Given a hybrid set $\mathscr{T}$ schedulable by EDF+TBS, let t' be the time at which all the active tasks completed their execution and let J_k be a pending aperiodic job not eligible (i.e., with $e_k > t'$). If the eligible time and the deadline assigned to J_k are modified as*

$$e'_k = t'$$
$$d'_k = e'_k + \frac{C_k}{U_s}$$

then, in each fully utilized interval $[t_1, t_2]$, the aperiodic execution demand never exceeds $(t_2 - t_1)U_s$ and set $\mathscr{T}$ remains schedulable.

Proof. By contradiction, suppose that there is a time-overflow (i.e., a deadline miss) at time t_2, and let t_1 be the last instant of time before t_2 such that all the active tasks with deadline less than t_2 completed their execution. Thus, $[t_1, t_2]$ is fully utilized (see definition 1). Let J_{k_1} be the first aperiodic job with $e_{k_1} \geqslant t_1$ and let J_{k_2} be the last aperiodic job with $d_{k_2} \leqslant t_2$. Clearly, by definition, $t_1 \geqslant t'$ and $k_1 \geqslant k$. The total demand of aperiodic execution in $[t_1, t_2]$ is

$$C_a = \sum_{j=k_1}^{k_2} C_j = \sum_{j=k_1}^{k_2} (d_j - e_j)U_s \geqslant (t_2 - t_1)U_s.$$

Since a time-overflow has occurred,

$$t_2 - t_1 < \sum_{i=1}^{n} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i + C_a \leqslant (t_2 - t_1)(U_p + U_s)$$

and hence,

$$U_p + U_s > 1,$$

which is a contradiction. □

Whenever an aperiodic job is completed earlier, again we can advance the eligible time and the deadline of the next aperiodic request, as shown by the following Lemma.

Lemma 5. *Given a hybrid set $\mathscr{T}$ schedulable by EDF+TBS, if aperiodic job J_{k-1} saves Δ units of computation time, the eligible time and the deadline of job J_k can be advanced as follows*

$$e'_k = \max\left(r_k, f_{k-1}, d_{k-1} - \frac{\Delta}{U_s}\right)$$
$$d'_k = e'_k + \frac{C_k}{U_s}$$

then, in each fully utilized interval $[t_1, t_2]$, the aperiodic execution demand never exceeds $(t_2 - t_1)U_s$ and set $\mathscr{T}$ remains schedulable.

Proof. For the sake of clarity, we restrict to the interval $[e_{k-1}, d_k]$. The total aperiodic execution demand in this interval is

$$\begin{aligned} C_{k-1} - \Delta + C_k &= U_s(d_{k-1} - e_{k-1}) - \Delta + U_s(d_k - e_k) \\ &\leqslant U_s\left(d_{k-1} - \frac{\Delta}{U_s}\right) - U_s e_{k-1} + U_s d_k - U_s\left(d_{k-1} - \frac{\Delta}{U_s}\right) \\ &= U_s(d_k - e_{k-1}). \end{aligned}$$

From Theorem 2 and Lemma 4, it follows that in every interval of length L that contains $[e_{k-1}, d_k]$, the total aperiodic execution demand never exceeds LU_s. Hence, $\mathscr{T}$ remains schedulable and the lemma follows. □

Why such a complicated rule for resource reclaiming? The reason descends from the fact that, under the SRP, a task must be executed with a deadline which is consistent with the preemption level assigned statically. In the original formulation of the TBS algorithm, an aperiodic request is immediately inserted in the ready queue with the suitable deadline. In this way, the reclaiming of processor utilization is performed automatically. But an aperiodic task could execute with a relative deadline greater than (C_k/U_S), and this fact is not consistent with Property 1 on preemption levels. With our reclaiming rule, this problem does not arise since $(d_k - e_k)$ is constant.

4.4. Other server mechanisms

What about using other aperiodic server mechanisms in the presence of resource constraints? Except for the TBS, dynamic aperiodic servers can mainly be divided into two classes: fixed capacity servers (e.g., Polling, Deferrable [9] or Sporadic Server [23]), and slack stealing techniques (e.g., EDL and IPE [23]).

Using slack stealing methods, resource constraints could be taken into account by computing slack intervals based on preemption of periodic tasks. However, extending this kind of servers is out of the scope of this paper.

If a fixed capacity server is used to handle aperiodic requests, under the SRP protocol, a suitable preemption level must be assigned to the server, even though aperiodic tasks do not share resources with hard tasks. For example, to verify Property 1, we could assign the server a preemption level inversely proportional to its period. In this case, however, if aperiodic tasks share resources with hard tasks, the blocking factors of hard periodic tasks cannot easily be bounded. For example, if an aperiodic job is executed within two instances of the server, it could enter a critical section during the first instance and release the semaphore during the second instance. As a result, the estimation of the maximum preemption times would be too pessimistic. Such a situation is depicted in Fig. 3, where two periodic tasks, τ_1 ($C = 2, T = 8$) and τ_2 ($C = 3, T = 12$), interact with two aperiodic jobs, J_1 and J_2, both having execution time $C_a = 2$ and release times $r_1 = 0$ and $r_2 = 1$. In particular, τ_1 and J_2 use the same resource during all their execution (they are dark shaded). Fig. 3(a) shows the schedule produced by a polling server with $C_s = 3$ and $T_s = 6$, for a total utilization of $(C_s/T_s) = 0.5$. Note that task τ_1 could experience a blocking time of $B_1 = 5$. Fig. 3(b) shows the schedule obtained by a TBS with $U_s = 0.5$. In this case, τ_1 cannot be blocked on the resource, since its preemption level is less than the preemption level assigned to job J_2. Thus, under the TBS, the

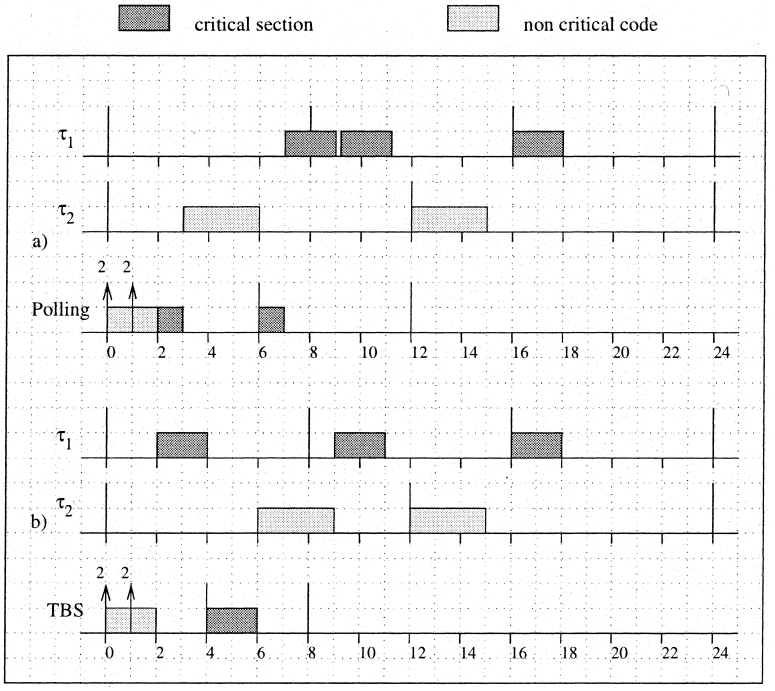

Fig. 3. Example of a periodic task which shares a resource with an aperiodic task scheduled by a polling server (a) and by a TBS (b).

blocking factor of τ_1 is $B_1 = 0$; under the polling server it is $B_1 = 5$, whereas with background it is $B_1 = 2$.

5. Deadlines different than periods

The schedulability analysis presented above can be extended to the case of periodic task sets with deadlines less than periods. To do that, we use the "*processor demand criterion*" adopted in Refs. [3,10] for testing the feasibility of periodic or sporadic task sets.

Theorem 6. *Let $\mathscr{T} = T^{\mathscr{P}} \cup T^{\mathscr{A}}$ be a hybrid set of tasks, where $\mathscr{T}^{\mathscr{P}}$ is a set of n hard sporadic tasks, and $\mathscr{T}^{\mathscr{A}}$ is a set of soft aperiodic tasks scheduled by a TBS with utilization U_s, that share resources under the SRP. Then, $\mathscr{T}$ is schedulable if, for all $L > 0$ and for all i, $1 \leqslant i \leqslant n$,*

$$L \geqslant \sum_{j=1}^{i} \left\{ \left\lfloor \frac{L - D_j}{T_j} \right\rfloor + 1 \right\} C_j + \max\{0, B_i - 1\} + S(i) L U_s. \tag{9}$$

Proof. The proof is an extension of proof of Theorem 1 and it is not reported here. It can be found in Ref. [16]. □

Although Eq. (9) should be tested for all $L > 0$, we can restrict the number of points in which they have to be verified to a finite set of elements. It is easy to show that Eq. (9) can be checked only for L equal to the deadlines of the jobs in $\mathscr{T}^{\mathscr{P}}$. Assuming that tasks in $\mathscr{T}^{\mathscr{P}}$ are ordered by decreasing preemption levels, let U_i be

$$U_i = \sum_{j=1}^{i} \frac{C_j}{T_j}.$$

Note that, if $U_i + U_s > 1$, there is an overload and task set is not schedulable. In the following, we assume that, for all i, $1 \leqslant i \leqslant n$, $U_i + U_s < 1$. This is a realistic assumption in practical systems. Let Θ_i be defined as follows

$$\Theta_i = \frac{U_i \max_{k=1,\ldots,i}(T_k - D_k) + \max\{0, B_i - 1\}}{1 - U_i - S(i) U_s},$$

where $S(i)$ is defined as in Eq. (8). Finally, we now prove that for all $L > \Theta_i$, the ith Eq. (9) is verified.

Theorem 7. *Let $\mathscr{T} = T^{\mathscr{P}} \cup T^{\mathscr{A}}$ be a hybrid set of tasks, where $\mathscr{T}^{\mathscr{P}}$ is a set of n hard sporadic tasks, and $\mathscr{T}^{\mathscr{A}}$ is a set of soft aperiodic tasks scheduled by a TBS with utilization U_s, that share resources under the SRP. Let P_i be*

$$P_i = \{ d_{j,k} \mid d_{j,k} = kT_j + D_j, d_{j,k} \leqslant \Theta_i, k \geqslant 0, 1 \leqslant j \leqslant i \}.$$

Then, $\mathscr{T}$ is schedulable if and only if, for all i, $1 \leqslant i \leqslant n$, and for all $L \in P_i$

$$\sum_{k=1}^{i} \left\{ \left\lfloor \frac{L - D_k}{T_k} \right\rfloor + 1 \right\} C_k + \max\{0, B_i - 1\} + S(i) L U_s \leqslant L. \tag{10}$$

Proof. The ith equation can be rewritten as

$$g(L) = \sum_{k=1}^{i} \left\{ \left\lfloor \frac{L - D_k}{T_k} \right\rfloor + 1 \right\} C_k + \max\{0, B_i - 1\} + S(i) L U_s - L \leqslant 0$$

and we note that

$$\begin{aligned} g(L) &\leqslant L \sum_{k=1}^{i} \frac{C_k}{T_k} - \sum_{k=1}^{i} \frac{D_k - T_k}{T_k} C_k + \max\{0, B_i - 1\} \\ &\quad + S(i) L U_s - L \leqslant L(U_i + S(i) U_s - 1) \\ &\quad + \max_{k=1,\ldots,i} (T_k - D_k) U_i + \max\{0, B_i - 1\}. \end{aligned}$$

Thus, when $g(L) \leqslant 0$, that is, when

$$L \geqslant \frac{U_i \max_{k=1,\ldots,i}(T_k - D_k) + \max\{0, B_i - 1\}}{1 - U_i - S(i) U_s} = \Theta_i$$

Eq. (10) is automatically verified. □

What about the computational complexity of test Eq. (10)? Baruah et al. [3] proved that the problem of deciding the feasibility of synchronous periodic or sporadic task sets with arbitrary deadlines is NP-hard in the weak sense. They found an algorithm which tests feasibility in pseudo–polynomial time. Our test is based on the same result, and has complexity $O(n \max(\Theta_i))$ which is pseudo-polynomial in the input size.

6. Conclusions

In this paper, we have addressed the problem of integrating resource constraints in the execution of hybrid task sets including hard periodic and soft aperiodic tasks. Schedulability conditions have been derived for the Earliest Deadline First algorithm, which has been extended to handle periodic tasks with deadlines less than periods which share resources with soft aperiodic tasks. The analysis has been carried out by considering that resources are accessed through the Stack Resource Policy and aperiodic tasks are serviced by the Total Bandwidth Server.

The complexity of the analysis proposed in this paper, when hard task have deadlines different than periods, is pseudo-polynomial, because the problem has been shown to be NP-hard [3]. As a consequence, the resulting schedulability tests might be too costly for providing an on-line guarantee. Nevertheless, this method can be used to perform off-line guarantee of critical periodic and sporadic tasks and to reserve sufficient bandwidth to serve aperiodic tasks on line.

Finally, this approach permits the definition of timing and resource constraints, without explicitly imposing the release times, as typically done in off-line scheduling approaches. In this way, we take advantage of dynamic scheduling for handling aperiodic requests on-line and automatically reclaiming resources whenever possible.

7. For further reading

Refs. [4,6,11,14,21,24,25].

Acknowledgements

The authors wish to thank Kevin Jeffay for the useful hints he gave during the development of this work.

References

[1] N.C. Audsley, A. Burns, M. Richardson, K. Tindell, A. Wellings, Applying new scheduling theory to static priority preemptive scheduling, Software Engineering Journal 8 (5) (1993) 284–292.

[2] T.P. Baker, Stack-based scheduling of real-time processes, The Journal of Real-Time Systems 3 (1) (1991) 76–100.

[3] S.K. Baruah, L.E. Rosier, R.R. Howell, Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one process, The Journal of Real-Time Systems (1990) 2.

[4] G.C. Buttazzo, HARTIK: A real-time kernel for robotics applications, in: Proceedings of the 14th IEEE Real-Time Systems Symposium, Raleigh-Durham, 1993, pp. 201–205.

[5] M.I. Chen, J.K. Lin, Dynamic priority ceilings: A concurrency control protocol for real-time systems, Journal of Real-Time Systems (1990) 2.

[6] R.I. Davis, K.W. Tindell, A. Burns, Scheduling slack time in fixed priority preemptive systems, in: Proceedings of Real-Time Systems Symposium, 1993 pp. 222–231.

[7] H. Kopetz, A. Damm, Ch. Koza, M. Mulazzani, W. Schwabl, Ch. Senft, R. Zainlinger, Distributed fault-tolerant real-time systems: The MARS approach, ACM Operating System Review 23 (3) (1989) 141–157.

[8] G. Fohler, Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems, in: Proceedings of IEEE Real-Time System Symposium, 1995 pp. 152–161.

[9] T.M. Ghazalie, T.P. Baker, Aperiodic servers in a deadline scheduling environment, The Journal of Real-Time System (1995) 9.

[10] K. Jeffay, Scheduling sporadic tasks with shared resources in hard-real-time systems, in: Proceedings of IEEE Real-Time System Symposium, 1992, pp. 89–99.

[11] K. Jeffay, D.L. Stone, Accounting for interrupt handling costs in dynamic priority task systems, in: Proceedings of IEEE Real-Time System Symposium, 1993, pp. 212–221.

[12] J.P. Lehoczky, L. Sha, J.K. Strosnider, Enhanced aperiodic responsiveness in hard real-time environments, in: Proceedings of IEEE Real-Time System Symposium, 1987, pp. 261–270.

[13] J.P. Lehoczky, S. Ramos-Thuel, An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems, in: Proceedings of IEEE Real-Time System Symposium, 1992.

[14] J. Leung, M. Merril, A Note on preemptive scheduling of periodic real-time tasks, Information processing Letters 12 9–12.

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

[16] G. Lipari, Resource constraints among periodic and aperiodic tasks, Technical Report, RETIS Lab, Scuola Superiore S. Anna, Pisa, Italy, 1997.

[17] L. Sha, L.R. Rajkumar, J.P. Lehoczky, Priority inheritance protocols: An approach to real-time synchronization, IEEE Transactions on Computers 39 (9) (1990).

[18] J.A. Stankovic, K. Ramamritham, The spring kernel: A new paradigm for real-time systems, IEEE Software 8 (3) (1991) 62–72.

[19] B. Sprunt, L. Sha, J.P. Lehoczky, Aperiodic task scheduling for hard real-time systems, The Journal of Real-Time Systems, 1 (1) 1989.
[20] M. Spuri, G.C. Buttazzo, Efficient aperiodic service under earliest deadline scheduling, in: Proceedings of IEEE Real-Time System Symposium, San Juan, Portorico, 1994.
[21] M. Spuri, J. Stankovic, How to integrate precedence constraints and shared resources in real-time scheduling, IEEE Transactions on Computers 43 (12) (1994) 1407–1412.
[22] M. Spuri, G.C. Buttazzo, F. Sensini, Robust aperiodic scheduling under dynamic priority systems, in: Proceedings of the 16th IEEE Real-Time System Symposium, 1995.
[23] M. Spuri, G.C. Buttazzo, Scheduling aperiodic tasks in dynamic priority systems, Real-Time Systems 10 (2) (1996).
[24] J.K. Strosnider, J.P. Lehoczky, L. Sha, The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments, IEEE Transactions on Computers 44 (1) (1995).
[25] T.S. Tia, J.W.S. Liu, M. Shankar, Algorithms and optimality of scheduling aperiodic requests in fixed-priority preemptive systems, The Journal of Real-Time Systems (1995).
[26] K. Tindell, A. Burns, A. Wellings, An extendible approach for analysing fixed priority hard real-time tasks, Journal of Real Time Systems 6 (2) (1994) 133–151.

Giorgio C. Buttazzo is an Assistant Professor of Computer Engineering at the Scuola Superiore S. Anna of Pisa, Italy. He received a B.E. degree in Electronic Engineering at the University of Pisa in 1985, a M.S. degree in Computer Science at the University of Pennsylvania in 1987, and a Ph.D. degree in Computer Engineering at the Scuola Superiore S. Anna of Pisa in 1991. During 1987, he also worked on active perception and real-time control at the G.R.A.S.P. Laboratory of the University of Pennsylvania. His main research interests include real-time computing, dynamic scheduling, multimedia systems, advanced robotics, and neural networks. Dr. Buttazzo has authored, and co-authored, three books on real-time systems and over sixty journal and conference papers.

Giuseppe Lipari is a Ph.D. student in Computer Engineering at the Scuola Superiore S. Anna of Pisa (Italy). In 1996, he graduated in Computer Engineering at University of Pisa. His reasearch activity is focused on the development and analysis of flexible scheduling algorithms in dynamic real-time systems for both hard predictable control applications and soft QoS-based applications. His research interests include real-time operating systems, scheduling algorithms, aperiodic service mechanisms, quality of service control, and multimedia applications.