Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS 1999), Phoenix, Arizona, December 1999.

Sharing Resources among Periodic and Aperiodic Tasks with Dynamic Deadlines

Marco Caccamo, Giuseppe Lipari, and Giorgio Buttazzo
Scuola Superiore S. Anna, Pisa, Italy
{caccamo,lipari,giorgio}@sssup.it

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. A tunable servicing technique is used to improve aperiodic responsiveness in the presence of resource constraints. The schedulability analysis is also extended to the case in which aperiodic deadlines vary at runtime. The results achieved in this paper can also be used for developing adaptive real-time systems, where task deadlines or periods can change to conform to new load conditions.

1. Introduction

In many real-time control applications, tasks cooperate through global resources (e.g., shared memory buffers) accessed in mutual exclusion to ensure data consistency of all common data structures. If the application timing constraints need to be guaranteed off-line, the maximum blocking delays introduced on tasks' execution by the concurrency control protocol used in the kernel must be taken into account in the schedulability analysis. Classical semaphores are not suited for enforcing mutual exclusion in real-time systems, since they may introduce unbounded priority inversion which can cause high priority tasks to miss their deadlines. To bound the priority inversion phenomenon and allow an off-line schedulability analysis, several approaches have been proposed in the literature.

In [9], resource contention is solved off line by constructing a static schedule, which is stored in a table and enforced at runtime using a time-driven approach. A different approach is used in the Spring kernel [16], where the schedule is constructed using a heuristic function which may integrate timing, resource, and precedence constraints [22]. Sha, Rajkumar, and Lehoczky, in [15], proposed two concurrency control protocols, the Priority Inheritance Protocol (PIP) and the Priority Ceiling Protocol (PCP), to bound the priority inversion phenomenon in fixed-priority systems scheduled by the Rate Monotonic (RM) algorithm. In [7], Jeffay described a method, the Dynamic Deadline Modification (DDM) protocol, for scheduling sporadic tasks with shared resources under the Earliest Deadline First (EDF) scheduling Algorithm. In [1], Baker proposed a general resource access protocol, the Stack Resource Policy (SRP), which can be used under fixed as well as dynamic priority assignments.

All the papers cited above assume that resources can be shared among hard tasks, which can be periodic or sporadic, but no soft tasks are considered in the model. In other works, soft aperiodic task management is integrated with periodic task scheduling, both under the RM algorithm [10, 17, 11, 21] and the EDF algorithm [6, 19, 3], however no resource constraints are considered in the schedulability analysis.

Such a lack in the scheduling theory is probably due to the fact that, when aperiodic tasks are handled by a server mechanism, their execution can be broken into many different chunks (this is especially true for a Sporadic Server), whose start time and duration depends on the actual system load and server capacity. In these conditions, if aperiodic tasks share resources with periodic tasks, bounding the maximum blocking times of periodic tasks to achieve feasible schedules is not easy.

On the other hand, real world control applications often consist of tasks with different criticality and activation characteristics which cooperate through shared resources. Thus, investigating the problem of scheduling hard periodic and soft aperiodic tasks under timing and resource constraints has certainly an important impact on real-time systems development.

In [12], this problem has been solved in the case in which periodic tasks are scheduled by EDF, aperiodic tasks are handled by the Total Bandwidth Server (TBS), and shared resources are managed by the SRP.

In this paper, we extend the analysis to deal with dynamic deadline modifications, in order to use the tunable TB* server [3], for improving aperiodic responsiveness in the presence of resource constraints. The results achieved in this paper can also be used in adaptive real-time systems, where task deadlines or periods can change to conform to new load conditions.

The rest of the paper is organized as follows. Section 2 briefly recalls the Total Bandwidth Server (TBS) algorithm and its tunable extension (TB*) for assigning optimal deadlines to soft aperiodic tasks. Section 3 presents the main results obtained on the scheduling analysis of hybrid (hard periodic and soft aperiodic) task sets when soft aperiodic tasks share resources with periodic tasks. Section 4 extends the analysis to the case in which aperiodic tasks can change their deadlines and illustrates the TB^* usage in the presence of resource constraints. Section 5 presents our conclusions and future works.

2. The Total Bandwidth Approach

In this section we briefly recall the Total Bandwidth Server (TBS) algorithm and its tunable extension (TB*) for assigning optimal deadlines to soft aperiodic tasks to improve their responsiveness.

2.1. The Total Bandwidth Server

The Total Bandwidth Server was proposed by Spuri and Buttazzo [18, 19] to improve the response time of soft aperiodic requests in a dynamic real-time environment, where tasks are scheduled according to EDF [13].

This is done by assigning a suitable deadline to each request, which is scheduled according to EDF together with the periodic tasks in the system. When the k-th aperiodic request arrives at time $t = r_k$, it receives a deadline

$$d_k = \max(r_k, d_{k-1}) + \frac{C_k^a}{U_s},$$

where C_k^a is the execution time of the request and U_s is the server utilization factor (i.e., its bandwidth). By definition $d_0 = 0$. The request is then inserted into the ready queue of the system and scheduled by EDF, as any periodic or sporadic instance.

Figure 1 shows an example of schedule produced with a TBS. The first aperiodic request, arrived at time $t = 6$, is assigned a deadline $d_1 = r_1 + \frac{C_1}{U_s} = 6 + \frac{1}{0.25} = 10$, and since d_1 is the earliest deadline

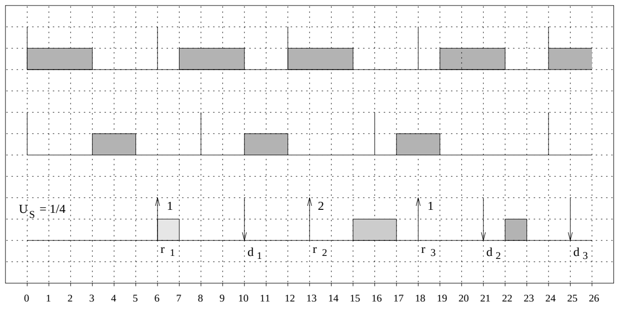

Figure 1. Total Bandwidth Server example.

in the system, the aperiodic activity is executed immediately. Similarly, the second request receives the deadline $d_2 = r_2 + \frac{C_2}{U_s} = 21$, but it is not serviced immediately, since at time $t = 13$ there is an active periodic task with a shorter deadline (18). Finally, the third aperiodic request, arrived at time $t = 18$, receives the deadline $d_3 = \max(r_3, d_2) + \frac{C_3}{U_s} = 21 + \frac{1}{0.25} = 25$ and is serviced at time $t = 22$.

Intuitively, the assignment of the deadlines is such that in each interval of time the fraction of processor time allocated by EDF to aperiodic requests never exceeds the server utilization U_s. As a consequence, the schedulability of a periodic task set in the presence of a TBS can simply be tested by verifying the following condition:

$$U_p + U_s \leq 1,$$

where U_p is the utilization factor of the periodic task set. This results is formally proved in [19].

2.2. The TB* algorithm

The key idea of the TB* algorithm is to assign each aperiodic request a deadline shorter than that given by the TBS rule.

The algorithm works in the following way. Each time an aperiodic task is eligible for execution (that is, as soon as it is released when there are no aperiodic tasks to execute, or all the previously arrived aperiodic tasks complete) it is first assigned a deadline d_k according to a TBS with a bandwidth $U_s = 1 - U_p$. Then, the algorithm tries to shorten this deadline as much as possible to enhance aperiodic responsiveness, still maintaining the periodic tasks schedulable. The feasibility of the shortening process is guaranteed by the following theorem [3]:

Theorem 1 (Buttazzo and Sensini, 97) *Let σ be a feasible schedule of task set $\mathcal{T}$, in which an aperiodic job J_k is assigned a deadline d_k, and let f_k be the finishing time of J_k in σ. If d_k is substituted with*

$d'_k = f_k$*, then the new schedule* σ*' produced by EDF is still feasible.*

The result stated in Theorem 1 is a direct consequence of the EDF optimality [5]. The process of shortening the deadline can recursively be applied to each new deadline, until no further improvement is possible, given that the schedulability of the periodic task set must be preserved. If d_k^s is the deadline assigned to the aperiodic request J_k at step s, and f_k^s is the corresponding finishing time in the current EDF schedule (achieved with d_k^s), the new deadline d_k^{s+1} is set at time f_k^s. At each step, the schedulability of the task set is guaranteed by Theorem 1. The algorithm stops either when $d_k^s = d_k^{s-1}$ or after a maximum number of steps defined by the system designer for bounding the complexity.

The possibility of bounding the number of shortening steps is a very important feature in dynamic systems, since it allows the designer to balance the performance of the algorithm versus its complexity. In the following, TB(N) will denote a Total Bandwidth server which performs at most N shortening steps in the deadline assignment rule. In particular, the plain TBS is equivalent to a TB(0). Moreover, TB* will denote a Total Bandwidth server which continues to shorten the deadline until $d_k^s = d_k^{s-1}$.

It is worth noting that the exact evaluation of f_k^s would require the development of the entire schedule up to the finishing time of request J_k, scheduled with d_k^s. However, there is no need to evaluate the exact value of f_k^s to shorten the deadline. Rather, an upper bound $\tilde{f}_k^s$ can be used to simplify the computation. It is defined as follows.

Definition 1 *When deadline* d_k^s *is assigned to job* J_k*,* $\tilde{f}_k^s$ *is defined as the time at which the aperiodic job* J_k *and all the periodic instances with deadline less than* d_k^s *complete execution.*

Based on its definition, $\tilde{f}_k^s$ can be computed as follows:

$$\tilde{f}_k^s = t + C_k^a + \mathcal{D}(t, d_k^s - 1), \qquad (1)$$

where t is the current time (corresponding to the release time r_k of request J_k or to the completion time of the previous request), C_k^a is the worst-case computation time required by J_k, and $\mathcal{D}(t, d_k^s - 1)$ is the interference on J_k due to the periodic jobs in the interval $[t,\ d_k^s)$. Notice that a periodic job with deadline equal to d_k^s does not interfere with J_k, because deadlines ties are broken in favor of aperiodic jobs.

It is worth observing that $\tilde{f}_k^s$ is an upper bound for f_k^s because the interference $\mathcal{D}$ is computed in $[t,\ d_k^s)$, rather than in $[t,\ f_k^s)$. As a consequence, $\tilde{f}_k^s$ could be the finishing time of a periodic instance. In general, $\tilde{f}_k^s \geq f_k^s$.

The periodic interference $\mathcal{D}(t, d_k^s - 1)$ in equation (1) can be expressed as the sum of two terms, $\mathcal{D}_a(t, d_k^s - 1)$ and $\mathcal{D}_f(t, d_k^s - 1)$, where $\mathcal{D}_a(t, d_k^s - 1)$ is the interference due to the currently active periodic instances with deadlines less than d_k^s, and $\mathcal{D}_f(t, d_k^s - 1)$ is the future interference due to the periodic instances activated after time t with deadline before d_k^s. Hence:

$$\mathcal{D}_a(t, d) \;=\; \sum_{\tau_i \ active,\ d_i \leq d} c_i(t) \qquad (2)$$

and

$$\mathcal{D}_f(t, d) \;=\; \sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{d - next\_r_i(t)}{T_i} \right\rfloor\right) C_i, \qquad (3)$$

where $next\_r_i(t)$ identifies the time greater than t at which the next periodic instance of task τ_i will be activated.

Since $\mathcal{D}_a$ and $\mathcal{D}_f$ can be computed in $O(n)$, the overall complexity of the deadline assignment algorithm is $O(Nn)$, where N is the maximum number of steps performed by the algorithm to shorten the initial deadline assigned by the TBS.

It can be shown that $\tilde{f}_k^s$ is a tight upper bound if it coincides with the deadline d_k^s.

2.2.1 An example

The following example illustrates the TB* deadline assignment algorithm. The task set consists of two periodic tasks, τ_1 and τ_2, with periods 3 and 4, and computation times 1 and 2, respectively. A single aperiodic job J_k arrives at time $t = 2$, requiring 2 units of computation time. The periodic utilization factor is $U_p = 5/6$, leaving a bandwidth of $U_s = 1/6$ for the aperiodic tasks.

When the aperiodic request arrives at time $t = 2$, it receives a deadline $d_k^0 = r_k + C_k^a/U_s = 14$, according to the TBS algorithm. The schedule produced by EDF using this deadline assignment is shown in Figure 2.

By applying equations (2) and (3) we have:

$$\begin{aligned} \mathcal{D}_a(2, 13) &= c_2(2) = 1 \\ \mathcal{D}_f(2, 13) &= 3C_1 + 2C_2 = 7 \end{aligned}$$

and, by equation (1), we obtain:

$$d_k^1 \;=\; \tilde{f}_k^0 \;=\; t + C_k^a + \mathcal{D}_a + \mathcal{D}_f \;=\; 12.$$

In this case, it can easily be verified that the aperiodic task actually terminates at $t = 12$. This happens because the periodic tasks do not leave any idle time to

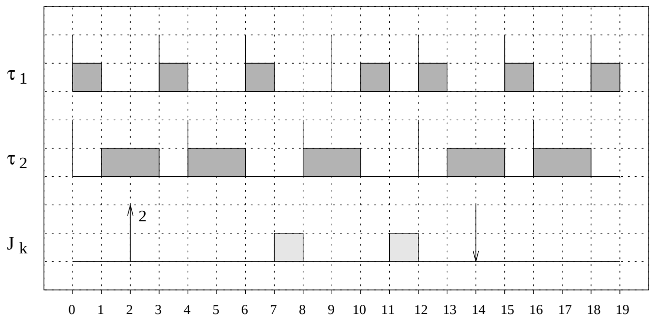

Figure 2. Schedule produced by EDF with $d_k^0 = 14$.

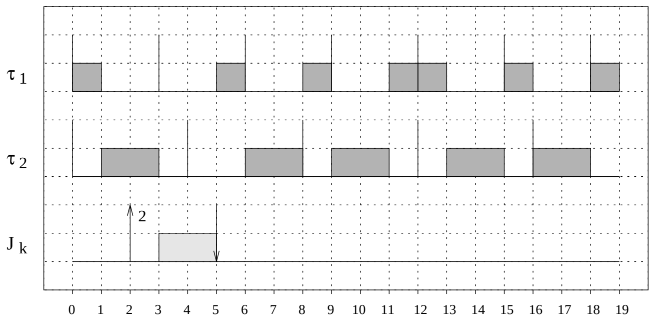

Figure 3. Schedule produced by EDF with $d_k^* = 5$.

the aperiodic task, which is thus compelled to execute at the end. Table 1 shows the subsequent deadlines evaluated at each step of the algorithm. In this example, six steps are necessary to find the shortest deadline for the aperiodic request.

step	d_k^s	$\tilde{f}_k^s$
0	14	12
1	12	9
2	9	8
3	8	6
4	6	5
5	5	5

Table 1. Deadlines and finishing times computed by the TB* algorithm.

The schedule produced by EDF using the shortest deadline $d_k^* = d_k^5 = 5$ is shown in Figure 3. Notice that at $t = 19$ the first idle time is reached, showing that the whole task set is schedulable.

2.2.2 Optimality

As far as the actual execution time of tasks is equal to the worst-case one, the TB* deadline assignment algorithm achieves optimality, in that it minimizes the response time of each aperiodic task among all scheduling algorithms which meet all periodic task deadlines, assuming that aperiodic requests are processed in FIFO order and that deadlines ties are broken in favor of aperiodic tasks. The result is summarized in the following theorem [3]:

Theorem 2 (Buttazzo and Sensini, 97) *For any periodic task set and any aperiodic arrival stream processed in FIFO order, the TB* algorithm minimizes the response time of every aperiodic task among all scheduling algorithms which meet all periodic task deadlines.*

3. TBS with resources

When soft aperiodic tasks share resources with periodic tasks, the duration of their critical sections must be taken into account in the feasibility analysis. In order to bound the maximum blocking time of each task and analyze the schedulability of hybrid task sets, we assume that shared resources are accessed using the Stack Resource Policy (SRP) [1].

According to this protocol, every 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

$$ceil(R_k) = \max_i \{\pi_i \mid \tau_i \;\; needs \;\; R_k\}. \qquad (4)$$

Finally, 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

[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.

"a task is not allowed to start executing until its priority is the highest among the active tasks 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 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.

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

$$B_i = \max\{s_{jh} \mid (D_i < D_j) \ \wedge \ \pi_i \leq \mathrm{ceil}(\rho_{jh})\}, \quad (5)$$

where s_{jh} is the worst-case execution time of the h-th critical section of task τ_j, D_j is its relative deadline and, ρ_{jh} is the resource accessed by the critical section s_{jh}. Given these definitions, the feasibility of a task set with resource constraints (when only periodic and sporadic tasks are considered), can be tested by the following sufficient condition [1]:

$$\forall i, \ 1 \leq i \leq n \quad \sum_{k=1}^{i} \frac{C_k}{T_k} + \frac{B_i}{T_i} \ \leq \ 1, \qquad (6)$$

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

In [12], the analysis has been extended to provide a tighter schedulability test, using a processor demand approach [2, 8]. The result is stated in the following theorem:

Theorem 3 (Lipari and Buttazzo, 99) *Let $\mathcal{T}^{\mathcal{P}}$ be a set of n hard periodic tasks ordered by decreasing preemption level (so that $\pi_i \geq \pi_j$ only if $i < j$). Then, $\mathcal{T}^{\mathcal{P}}$ is schedulable by EDF+SRP if*

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \ \leq \ 1,$$

$$\forall\, i, \quad 1 \leq i \leq n \ \ \forall\, L, \ T_i \leq L \ < \ T_n$$
$$L \geq \ \sum_{j=1}^{i} \left\lfloor \frac{L}{T_j} \right\rfloor C_j + \max\{0, B_i - 1\}.$$

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

To use the SRP along with a TBS, aperiodic tasks must be assigned a suitable preemption level. In particular, each aperiodic request J_k is assigned the following preemption level:

$$\pi_k = \frac{U_s}{C_k^a}. \qquad (7)$$

Notice that $\frac{C_k^a}{U_s} = [d_k - e_k]$ 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 equation (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 computed using equation (5):

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

where now j ranges over the whole task set, including both periodic and aperiodic tasks, and, if τ_j is an aperiodic task, $D_j = \frac{C_j}{U_s}$. Then, the schedulability of the hybrid task set can be guaranteed based on the following theorem [12]:

Theorem 4 (Lipari and Buttazzo, 99) *Let $\mathcal{T}^{\mathcal{P}}$ be a set of n hard periodic tasks ordered by decreasing preemption level (so that $\pi_i \geq \pi_j$ only if $i < j$), and let $\mathcal{T}^{\mathcal{A}}$ be a set of aperiodic tasks scheduled by a TBS with utilization U_s. Then, set $\mathcal{T}^{\mathcal{P}}$ is schedulable by EDF+SRP+TBS if*

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + U_s \ \leq \ 1. \qquad (8)$$

$$\forall\, i, \quad 1 \leq i \leq n \ \ \forall\, L, \ T_i \leq L \ < \ T_n$$
$$L \geq \ \sum_{j=1}^{i} \left\lfloor \frac{L}{T_j} \right\rfloor C_j + \max\{0, B_i - 1\} + LU_s. \ (9)$$

4. TB* with resources

In this section we extend the analysis to the case in which tasks can change their deadlines and analyze the TB^* usage in the presence of resource constraints. Unfortunately, using the TB^* algorithm with SRP protocol is not trivial. In fact, whenever the deadline of job J_k is shortened, the preemption level associated with J_k increases, and the blocking factors of all the tasks in the set change at each shortening step. When an aperiodic task is served by TB^*, its fictitious deadline is computed on line depending on the current work-load, so we cannot decide off line the *actual preemption level* $\pi_k = 1/D_k$ that will be assigned to aperiodic task J_k at run-time. When the task set has to be guaranteed off line, it's necessary to know the *minimum relative deadline* D_k^{min} that can be assigned to request J_k by TB*. By fixing D_k^{min}, we can assign each aperiodic task J_k a *maximum preemption level* $\pi_k^{max} = 1/D_k^{min}$ that will be used to compute off line the ceiling of every resource. If D_k is the relative deadline of aperiodic task J_k assigned on line by TB^*, the following inequality holds:

$$\forall k,\ \ \pi_k^{max} \geq \frac{1}{D_k}.$$

Note that, in our model an aperiodic task J_k starts its execution only if its priority is the highest among the active tasks (it is assumed that deadlines ties are broken in favor of aperiodic jobs) and only if its *actual preemption level* $\pi_k = 1/D_k$ is greater than the system ceiling. In order for the SRP protocol to be correct, every resource R_i is assigned a static[2] ceiling $ceil(R_i)$ (we assume binary semaphores) that is equal to the highest preemption level of the tasks that could be blocked on R_i when the resource is busy. Hence, $ceil(R_i)$ can be computed as follows:

$$\begin{aligned} ceil(R_i) = \quad & \max[\{\pi_j \mid \tau_j \text{ periodic needs } R_i\} \cup \\ & \{\pi_k^{max} \mid J_k \text{ aperiodic needs } R_i\}]. \end{aligned} \tag{10}$$

It is easy to note that the ceiling of a resource, computed by equation (10), is greater than or equal to the one computed using the preemption level of periodic tasks and the actual preemption level of aperiodic tasks (see equation (4)).

Furthermore, we need to change the definition of blocking time for a periodic task. In fact, in computing the blocking time for a periodic task, we need to take into account the duration of the critical section of an aperiodic task without considering its relative deadline, because it is assigned on-line and cannot be known beforehand. Hence:

$$\begin{aligned} B_i = \quad & \max\{s_{j,h} \mid \pi_i \leq ceil(\rho_{jh}) \wedge \\ & ((\tau_j \text{ is periodic with } D_i < D_j) or \\ & (\tau_j \text{ is aperiodic}))\}. \end{aligned} \tag{11}$$

These modifications do not change any property of the SRP and permit to keep a static ceiling for the resources even when the fictitious deadlines of aperiodic jobs are computed on line by the TB^* server.

Now we need to compute a bandwidth U_s to safely assign an initial deadline to an aperiodic task: then this deadline can be advanced by the TB* algorithm. We also need a way to assign each aperiodic task a maximum preemption level π_k^{max}.

In the next section we propose a method for assigning these variables such that the periodic tasks' schedulability is not jeopardized.

4.1. Off-line analysis

Condition (8) imposes that $U_s \leq 1 - \sum_{i=1}^{n} \frac{C_i}{T_i}$. If aperiodic tasks do not use resources, then the blocking

```
Algorithm bandwidth_reservation
{
    $x = 0$;
    $U_s^{def} = U_s^{inf}(x) = 0$
    $U_s^{sup}(x) = 1 - \sum_{i=1}^{n} \frac{C_i}{T_i}$
    while ($U_s^{sup}(x) - U_s^{inf}(x) \geq$ ERR) {
        $U_s(x) = \frac{U_s^{sup}(x) + U_s^{inf}(x)}{2}$
        for each aperiodic task $J_k$ $\pi_k^{max}(x) = \frac{U_s(x)}{C_k}$
        for each resource $\rho$ compute ceil($\rho$)
        for each periodic task $\tau_i$ compute $B_i$
        if (9) is verified {
            $U_s^{def} = U_s(x)$
            $U_s^{sup}(x+1) = U_s^{sup}(x)$
            $U_s^{inf}(x+1) = U_s(x)$
        }
        else {
            $U_s^{sup}(x+1) = U_s(x)$
            $U_s^{inf}(x+1) = U_s^{inf}(x)$
        }
        $x = x + 1$
    }
}
```

Figure 4. Algorithm for reserving a bandwidth U_s to TBS.

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

time B_i do not depend on the aperiodics. In this simple case, U_s can be computed as:

$$U_s = \min\{\left(1 - \sum_{i=1}^{n} \frac{C_i}{T_i}\right) \cup \min_{i,L}\left(\frac{L - \sum_{j=1}^{i}\left\lfloor \frac{L}{T_j}\right\rfloor C_j - max\{0, B_i - 1\}}{L}\right)\}.$$

In the case in which aperiodic tasks use resources, the problem is complicated by the fact that the resource ceilings depend upon the maximum preemption levels assigned to aperiodic tasks. The following algorithm computes the bandwidth U_s for the server and a first approximation for the maximum preemption levels for the aperiodic tasks.

We assume that, for $U_s = 0$, the system is schedulable. In fact, this is equivalent to schedule the aperiodic tasks in background: if the system is not schedulable with $U_s = 0$, then it cannot be schedulable with $U_s > 0$.

The pseudo-code for the algorithm is shown in Figure 4. The algorithm precision depends on the value of the ERR constant, which must be carefully chosen to balance precision against computational complexity (which is $\mathcal{O}(mnlog\left(\frac{U_s^{sup}(0) - U_s^{inf}(0)}{ERR}\right)))$.

At the end of the previous algorithm we have in U_s^{def} the bandwidth used to assign the initial deadline to the aperiodic tasks, and a first approximation for the maximum preemption levels; in fact each aperiodic task is assigned a initial value of maximum preemption level $\pi_k^{max} = \frac{U_s^{def}}{C_k}$. However, these can be further improved.

If an aperiodic task J_k does not use resources, we can assign $\pi_k^{max} = \infty$, because the relative deadline D_k can be advanced as much as possible by the TB^*. In fact, neither the resource ceilings nor the periodic blocking times depend upon the maximum preemption level of this task: hence the schedulability of the system does not change. If, on the contrary, J_k uses resources, we must proceed by successive refinements.

Consider an aperiodic task J_k that uses resources, and define at step $x = 0$, $\pi_k^{max}(0) = \frac{U_s^{def}}{C_k}$. Now suppose that at the generic step x, this maximum preemption level is between the preemption levels of two periodic tasks, i.e.:

$$\pi_{i+1} \leq \pi_k^{max}(x) < \pi_i.$$

What happens if, at next step, we increase $\pi_k^{max}(x+1)$ such that it becomes equal to π_i? First, note that only the ceilings of the resources that are used by J_k can increase. In any case, it's easy to see that for any periodic task τ_j, with $j < i$ or $j \geq i+1$, the blocking time B_j does not change. Only for task τ_i the blocking time can increase. Thus we have only to re-calculate the resource ceilings and the blocking time B_i, and check whether the i-th equation of (9) still holds:

$$\forall L, \qquad T_i \leq L < T_n$$
$$L \geq \sum_{j=1}^{i}\left\lfloor \frac{L}{T_j}\right\rfloor C_j + \max\{0, B_i - 1\} + LU_s.$$

Notice that U_s is fixed and it is obtained by the previous algorithm. If so, then we can set $\pi_k^{max}(x+1) = \frac{1}{T_{i-1}}$ and repeat the procedure. If $i = 1$, then $\pi_k^{max} = \infty$ and we can stop. Instead, if the previous condition does not hold, we set $\pi_k^{max} = \frac{1}{T_i+1}$ and stop the algorithm. We can repeat the same algorithm for every aperiodic task that uses resources. It can be shown that:

- the maximum preemption level obtained in this way is the maximum that can be achieved off-line for J_k;
- the maximum preemption level for task J_k is not influenced by the maximum preemption level of other aperiodic tasks.

Proofs are not reported for space limitation.

4.2. Algorithm description

The TB^* algorithm in the presence of resource constraints works in a similar way as the TB^* algorithm for independent tasks described in Section 2.2. In fact, each time an aperiodic task is eligible for execution, it is first assigned a deadline d_k according to a TBS with a bandwidth U_s. Then, the algorithm tries to shorten this deadline as much as possible to enhance aperiodic responsiveness, still maintaining the periodic tasks schedulable. Since tasks may share resources, the upper bound $\tilde{f}_k^s$ on the aperiodic response time has to be computed to consider the blocking time of the aperiodic task to be served.

Let $S(t, d_k^s)$ be the set of all the tasks with deadline greater than or equal to d_k^s that are active and inside a critical section at time t, that is:

$$S(t, d_k^s) = \{\tau_{i,j} | (\tau_{i,j} \text{ is active and inside a critical section at time } t) \wedge (d_{i,j} \geq d_k^s)\}. \quad (12)$$

Moreover, we define $D^{a^{max}}(t, d_k^s)$ to be the maximum relative deadline between tasks which create interference in $[t, d_k^s)$, that is the interference due to the currently active (at time t) instances with deadline less than d_k^s, and the future interference due to the instances activated after time t with deadline before d_k^s. That is:

$$D^{a^{max}}(t, d_k^s) = \{0\} \cup max_{i,j}\{D_{i,j} | (\tau_{i,j} \text{ creates interference in } [t, d_k^s)\}. \quad (13)$$

Assuming that $s_{i,r}(t)$ is the residual computation time (at time t) that τ_i has to spend inside the critical section on resource r, we introduce the notion of *"actual aperiodic blocking time"* $B^a(t, d_k^s)$ of an aperiodic task. This function represents the amount of time that a task J_k (eligible at time t) with fictitious deadline d_k^s must be blocked according to the SRP protocol. More formally, $B^a(t, d_k^s)$ can be defined as follows:

Definition 2 *Considering interval $[t, d_k^s]$, if $\tau_{\overline{m}}$ is the task (if any) with minimum deadline among the tasks in $S(t, d_k^s)$, we define the* "actual aperiodic blocking time" *$B^a(t, d_k^s)$ of aperiodic task J_k (eligible at time t) with deadline d_k^s, as:*

$$B^a(t, d_k^s) = \{0\} \cup max_r\{s_{\overline{m},r}(t) | ceil(r) \geq \frac{1}{max\{D^{a^{max}}(t, d_k^s), d_k^s - t\}}\}. \quad (14)$$

Note that the max_r in equation (14) is needed only in the presence of nested critical sections. In the presence of resource constraints, the computation of $\tilde{f}_k^s$ (which still represents the time at which the aperiodic job J_k and all the periodic instances with deadline less than d_k^s complete their execution) has to be changed as follows to take the actual aperiodic blocking time $B^a(t, d_k^s)$ into account:

$$\tilde{f}_k^s = t + C_k^a + \mathcal{D}(t, d_k^s - 1) + B^a(t, d_k^s), \quad (15)$$

where t is the current time (corresponding to the eligible time e_k of request J_k), C_k^a is the worst-case computation time required by J_k, $\mathcal{D}(t, d_k^s - 1)$ is the interference on J_k due to the periodic jobs in the interval $[t,\ d_k^s)$ and $B^a(t, d_k^s)$ is the blocking time as defined above. Notice that a periodic job with deadline equal to d_k^s does not interfere with J_k, but it can contribute to block J_k, because deadlines ties are broken in favor of aperiodic jobs.

The following theorem sets the basis for our method.

Theorem 5 *Let σ be a feasible schedule produced by EDF + SRP for a task set $\mathcal{T}$ composed by periodic tasks and an aperiodic task J_k, having deadline d_k and maximum preemption level π_k^{max}. Let $\tilde{f}_k$ be an upper bound on the finishing time of J_k, computed as follows:*

$$\tilde{f}_k = e_k + C_k^a + \mathcal{D}(e_k, d_k - 1) + B^a(e_k, d_k),$$

(where e_k is the eligible time of aperiodic task J_k). If $\pi_k^{max} \geq \frac{1}{\tilde{f}_k - e_k}$, then the new task set $\mathcal{T}'$ obtained substituting d_k with $d'_k = \tilde{f}_k$ is still schedulable and the schedule σ' obtained by EDF+SRP is feasible.

Proof.
See [4]. □

Theorem 5 ensures that, if the task set $\mathcal{T}$ is schedulable by assigning J_k a deadline d_k, it will be also schedulable by advancing d_k up to the finishing time $\tilde{f}_k$, as performed by the TB^*.

Comparing the TB^* in the presence of resource constraints to the one for independent tasks, we notice that Theorem 5 introduces a constraint when the algorithm tries to shorten an aperiodic deadline. In fact the following condition has to be verified for each new deadline:

$$\forall s, k \ \frac{1}{d_k^s - t} \leq \pi_k^{max}. \quad (16)$$

The algorithm stops either when $d_k^s = d_k^{s-1}$, or after a maximum number of steps defined by the system designer for bounding the complexity, or when equation (16) does not hold.

4.2.1 An example

The following example illustrates the TB* deadline assignment algorithm in the presence of resource constraints. The task set consists of three periodic tasks τ_1, τ_2, τ_3, and an aperiodic task J_k which share two resources R_1 and R_2; in particular τ_1 and τ_2 share the resource R_1, while τ_3 and J_k share the resource R_2. The task set parameters are shown in Table 2. It is assumed that the maximum preemption level of aperiodic task J_k is $\pi_k^{max} = 1/C_k^a = 0.5$; hence, according to the result of Theorem 4, a bandwidth $U_s = 1/7$ can be safely assigned to the aperiodic task. In this example, a single aperiodic job J_k arrives at time $t = 2$, requiring 2 units of computation time.

When the aperiodic request arrives at time $t = 2$, it receives a deadline $d_k^0 = r_k + C_k^a/U_s = 16$, according to the TBS algorithm.

Table 3 shows the subsequent deadlines evaluated at each step of the algorithm. In this example, the TB^* algorithm performs three iterations to assign J_k a deadline $d_k^3 = 7$.

task	C_i	T_i	R_1	R_2
J_k	2	-	-	2
τ_1	1	4	1	-
τ_2	4	10	4	-
τ_3	4	20	-	4

Table 2. Parameters of the task set.

step	d_k^s	$\tilde{f}_k^s$	$B^a(t, d_k^s)$
0	16	9	0
1	9	8	3
2	8	7	3
3	7	7	3

Table 3. Deadlines and finishing times computed by the TB* algorithm.

When the process of shortening the aperiodic deadline occurs, at every step the actual aperiodic blocking time $B^a(t, d_k^s)$ of J_k has to be computed. At the first step, the actual aperiodic blocking time is $B^a(2, 16) = 0$, then, at the second step, the task τ_2 blocks J_k, so the actual aperiodic blocking time becomes $B^a(2, 9) = 3$, and the same occurs at the next steps; thus, we have:

$$B^a(2,16) \;=\; 0.$$

$$B^a(2,9) \;=\; B^a(2,8) \;=\; B^a(2,7) \;=\; 3.$$

The schedule produced by EDF using the deadline $d_k^* = d_k^3 = 7$ is shown in Figure 5. Notice that at $t = 19$ the first idle time is reached, showing that the whole task set is schedulable.

In section 2.2.2 is shown that the TB^* for independent tasks is an optimal algorithm. Unfortunately, in the presence of resource constraints, the TB^* loses its optimality. In fact, Figure 6 shows that the optimal deadline for J_k is $d_k^{opt} = 4$, where $d_k^{opt} < d_k^* = 7$.

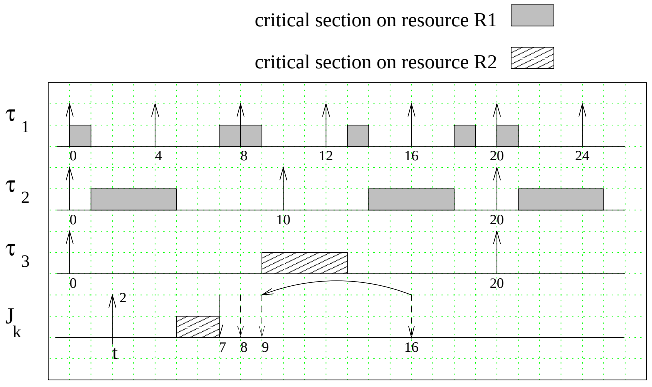

Figure 5. Schedule produced by EDF+SRP with $d_k^* = 7$.

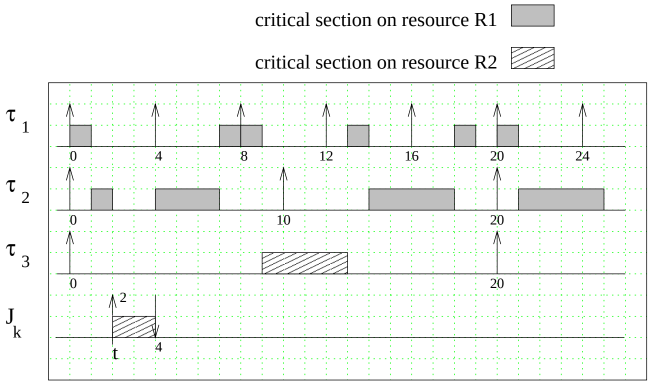

Figure 6. Optimal schedule produced by EDF + SRP with $d_k^{opt} = 4$.

5. Conclusions

In this paper, we addressed 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. The analysis has been carried out by considering that resources are accessed through the Stack Resource Policy and aperiodic tasks are serviced by the tunable TB^* server. This servicing technique is used to improve aperiodic responsiveness in the presence of resource constraints.

The off-line analysis proposed in this paper, can be used to perform off-line guarantee of critical periodic tasks and to reserve a bandwidth to serve aperiodic tasks on line.

In the presence of resource constraints we showed that the TB^* loses optimality. As the future work we plan to further investigate this issue and extend the analysis to periodic task set with elastic periods.

References

[1] T.P. Baker, "Stack-Based Scheduling of Real-Time Processes," *The Journal of Real-Time Systems* 3(1), 1991, pp. 67–100.

[2] S. K. Baruah, R. R. Howell, and L. E. Rosier, "Algorithms and Complexity Concerning the Pre-

emptive Scheduling of Periodic Real-Time Tasks on One Processor," The Journal of Real-Time Systems, 2, 1990.

[3] G. C. Buttazzo and F. Sensini. "Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time environments," *Proceedings of the 3rd IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'97)*, pages 39–48, Como, Italy, September 1997.

[4] M. Caccamo, G. Lipari, G. Buttazzo, "Sharing Resources with the TB* server" *Technical Report TR 04-99*, Retis Lab, Scuola Superiore S. Anna, April 1999.

[5] M.L. Dertouzos, "Control Robotics: the Procedural Control of Physical Processes," *Information Processing*, 74, North-Holland, Publishing Company, 1974.

[6] T.M. Ghazalie and T.P. Baker, "Aperiodic Servers In A Deadline Scheduling Environment," *The Journal of Real-Time Systems*, 9, 1995, pp. 21–36.

[7] K. Jeffay, "Scheduling Sporadic Tasks with Shared Resources in Hard Real-Time Systems," *Proceedings of IEEE Real-Time System Symposium*, pp. 89-99, December 1992.

[8] K. Jeffay and D.L. Stone, "Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems", *Proceedings of IEEE Real-Time System Symposium*, pp. 212-221, December 1993.

[9] H. Kopetz, A. Damm, Ch. Koza, M. Mulazzani, W. Schwabl, Ch. Senft, and R. Zainlinger, Distributed Fault-Tolerant Real-Time Systems : The MARS Approach, ACM Operating System Review, Vol. 23, No. 3, pp, 141-157, (July 1989).

[10] J.P. Lehoczky, L. Sha, and J.K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," *Proc. of Real-Time Systems Symposium*, 1987, pp. 261–270.

[11] J.P. Lehoczky and S. Ramos-Thuel, "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems," *Proc. of Real-Time Systems Symposium*, 1992, pp. 110–123.

[12] G. Lipari and G. Buttazzo, "Schedulability Analysis of Periodic and Aperiodic Tasks with Resource Constraints," to appear on the *Journal of Systems Architectures.*

[13] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard real-Time Environment," *Journal of the ACM* 20(1), 1973, pp. 40–61.

[14] A.K. Mok, *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*, Ph.D. Dissertation, MIT, 1983.

[15] L. Sha, L.R. Rajkumar, J.P. Lehoczky, Priority Inheritance Protocols: An Approach to Real-Time Synchronization, IEEE Transactions on Computers, 39(9), (1990).

[16] J.A. Stankovic and K. Ramamritham, The Spring Kernel: A New Paradigm for Real-Time Systems, *IEEE Software*, Vol. 8, No. 3, pp. 62-72, (May 1991).

[17] B. Sprunt, L. Sha, and J.P. Lehoczky, "Aperiodic Task Scheduling for Hard-Real-Time Systems," *The Journal of Real-Time Systems* 1, 1989, pp. 27–60.

[18] M. Spuri and G.C. Buttazzo, "Efficient Aperiodic Service under Earliest Deadline Scheduling", *Proc. of the IEEE Real-Time Systems Symposium*, San Juan, Portorico, December 1994.

[19] M. Spuri and G.C. Buttazzo, "Scheduling Aperiodic Tasks in Dynamic Priority Systems", *The Journal of Real-Time Systems*, 10(2), 1996.

[20] J.K. Strosnider, J.P. Lehoczky, and L. Sha, "The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", *IEEE Transactions on Computers*, 44(1), January 1995.

[21] T.S. Tia, J.W.S. Liu, and M. Shankar, "Algorithms and Optimality of Scheduling Aperiodic Requests in Fixed-Priority Preemptive Systems", *The Journal of Real-Time Systems*, 1995.

[22] Wei Zhao, K. Ramamritham, and J. Stankovic, "Scheduling Tasks with Resource Requirements in Hard Real-Time Systems", *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 5, May 1987, pp 564–677.