Computing the Minimum EDF Feasible Deadline in Periodic Systems

Hoai Hoang[1], Giorgio Buttazzo[2], Magnus Jonsson[1], Stefan Karlsson
[1]CERES - Centre for Research on Embedded Systems, Halmstad University, Halmstad, Sweden
Email: Hoai.Hoang@ide.hh.se, Magnus.Jonsson@ide.hh.se
[2]Scuola Superiore S. Anna, Pisa, Italy Email: Giorgio@sssup.it

Abstract

In most real-time applications, deadlines are artifices that need to be enforced to meet different performance requirements. For example, in periodic task sets, jitter requirements can be met by assigning suitable relative deadlines and guaranteeing the feasibility of the schedule.

This paper presents a method (called `minD`*) for calculating the minimum EDF-feasible deadline of a real-time task. More precisely, given a set of periodic tasks with hard real-time requirements, which is feasible under EDF, the proposed algorithm allows computing the shortest deadline that can be assigned to an arbitrary task in the set, or to a new incoming task (periodic or aperiodic), still preserving the EDF feasibility of the new task set. The algorithm has a pseudo polynomial complexity and handles arbitrary relative deadlines, which can be less than, equal to, or greater than periods.*

1. Introduction

Most real-time applications involve the execution of periodic activities to perform data sampling, sensory processing, action planning, actuation, and control. The stability and the performance of a control system is then influenced by a number of timing variables, including the sampling periods, the input-output delays, and the sampling jitter [2]. Although task periods can be precisely enforced by a real-time operating system, input-output delays depend on the processor speed and on the specific task interactions, whereas jitter is mainly induced by scheduling.

The effect of jitter on real-time control applications has been extensively studied in the literature [11, 17] and several techniques have been proposed to cope with it. Marti el al. [18] proposed a compensation technique according to which control actions are properly computed depending on the temporal distance between successive samples. Di Natale and Stankovic [19] proposed the use of simulated annealing to find the optimal configuration of task offsets that minimizes jitter, according to some user defined cost function. Cervin et al. [12] presented a method for finding an upper bound of the input-output jitter of each task by estimating the worst-case and the best-case response time under EDF scheduling, but no method is provided to reduce the jitter by shortening task deadlines. Rather, the concept of *jitter margin* is introduced to simplify the analysis of control systems and guarantee their stability when certain conditions on jitter are satisfied.

Another way of reducing the jitter is to limit the execution interval of each task by setting a suitable relative deadline. Working on this line, Baruah et al. [4] proposed two methods for assigning shorter relative deadlines to tasks and guaranteeing the schedulability of the task set. The first method is based on task utilizations and runs in polynomial time, whereas the second method has a pseudo-polynomial complexity since it is based on the processor demand criterion [6]. However, none of them can be used to compute the shortest possible deadline that minimizes jitter under schedulability constraints.

Brandt et al. [7] also addressed the problem of reducing the deadline of a periodic task, but their approach is based on the processor utilization, hence it cannot find the shortest possible deadline. Moreover their method is used to compute the minimum feasible period, while this paper discusses how to compute the minimum feasible deadline, without modifying the period.

Shin et al. [23] presented a method for computing the minimum deadline of a newly arrived task, assuming the existing task set is feasibly schedulable by EDF; however, their approach is tailored for distributed applications and requires some off-line computation.

Buttazzo and Sensini [9] also presented an on-line algorithm to compute the minimum deadline to be assigned to a new incoming task in order to guarantee feasibility under EDF. However, their approach only applies to aperiodic requests that have to be executed in a periodic environment.

A similar problem has been independently addressed in [3] using a slightly different approach, but the basic idea for computing the minimum relative deadline for a new incom-

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE

ing periodic task was originally proposed in [15], although no proofs or simulations were presented.

In this paper, the problem is formally presented and generalized to find the minimum deadline of arbitrary tasks with periodic, aperiodic, or sporadic activation pattern, and with deadlines less than, equal to, or greater than periods. The complexity of the algorithm for computing the minimum deadline is the same as the classical acceptance test for deadlines different from periods.

The importance of this method in real-time applications is that it can be effectively used to reduce the response time of specific control activities or limit their input-output jitter, with the purpose of meeting performance and stability requirements. As another motivation, the proposed method can also be very useful to reduce the end-to-end deadlines associated with real-time transactions in multi-hop networks. In fact, given a multi-hop network with an arbitrary topology, any real-time transaction that has to traverse m links to reach its destination within a deadline D_a, can be considered as a chain of m real-time messages, each with its local (node-to-node) deadline: $D_{a,1}, D_{a,2}, \ldots, D_{a,m}$, such that $D_a = \sum_{k=1}^{m} D_{a,k}$. Under this situation, the `minD` algorithm can be used to assign the minimum EDF feasible deadline to each local message, in order to maximize the *slack* available for the transaction and improve system responsiveness.

The rest of the paper is organized as follows. Section 2 describes the system model and precisely introduces the problems to be addressed. Section 3 illustrates the deadline minimization algorithm. Section 4 presents some experimental results and compares the proposed algorithm with another deadline minimization approach. Finally, Section 5 states our conclusions and future work.

2. Terminology and Assumptions

We consider a set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of periodic tasks that have to be executed on a uniprocessor system. Each task τ_i consists of an infinite sequence of jobs, or task instances, having the same worst-case execution time (WCET), the same relative deadline, and the same interarrival period. We allow tasks to start at different times, although the guarantee test is performed in the worst-case scenario, occurring for synchronous activations [20]. All tasks are fully preemptive.

The following notation is used throughout the paper:

$\tau_{i,j}$ denotes the j-th job of task τ_i, with $j \in \mathcal{N}$.

C_i denotes the worst-case execution time (WCET) of task τ_i, that is, the WCET of each job of τ_i.

T_i denotes the period of task τ_i, or the minimum interarrival time between successive jobs.

D_i denotes the relative deadline of task τ_i, that is, the maximum finishing time allowed for any job, relative to its activation time.

$d_{i,j}$ denotes the absolute deadline of job $\tau_{i,j}$, that is the maximum absolute time, before which job $\tau_{i,j}$ must complete.

U_i denotes the utilization of task τ_i, that is, the fraction of cpu time used by τ_i ($U_i = C_i/T_i$).

U denotes the total utilization of the task set, that is, the sum of all tasks utilizations ($U = \sum_{i=1}^{n} U_i$).

$h_i(t)$ denotes the *processor demand* of task τ_i in $[0, t]$, that is the sum of WCETs of the jobs $\tau_{i,j}$ with arrival time and absolute deadline in $[0, t]$.

$h(t)$ denotes the total processor demand of the task set in $[0, t]$, that is the sum of the individual demands $h_i(t)$ of the tasks in the set.

H denotes the hyperperiod of the task set, that is the minimum time interval after which the schedule repeats itself. For a set of periodic tasks with zero offset, it is equal to the least common multiple of all the periods ($H = lcm(T_1, ..., T_n)$).

It is important to recall that periodic tasks can have arbitrary deadlines, which can be less than, greater than, or equal to periods. We assume that tasks are scheduled by the Earliest Deadline First (EDF) algorithm [16], according to which jobs are assigned priorities inversely proportional to their absolute deadlines: the shorter the deadline, the higher the priority.

2.1. Problem statement

Deadlines of tasks' are artifices often used by the designer to enforce performance requirements (e.g., response times, communication delays, or input-output jitter) which affect the behavior of a computer controlled system. The algorithm proposed in this paper allows us to find the shortest deadline that can be assigned to a task, without jeopardizing the task set schedulability. More specifically, the proposed method can be used to solve the following problems:

1. **Problem 1**
   Given a set $\mathcal{T}$ of n periodic tasks, with $U < 1$, that is feasible under EDF, find the minimum relative deadline D_k^{min} that can be assigned to an arbitrary task τ_k, such that the EDF-feasibility of the task set is preserved.
2. **Problem 2**
   Given a set $\mathcal{T}$ of n periodic tasks, with $U < 1$, that is feasible under EDF, find the minimum relative deadline of each task, following a given arbitrary order, such that the EDF-feasibility of the task set is preserved.

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE

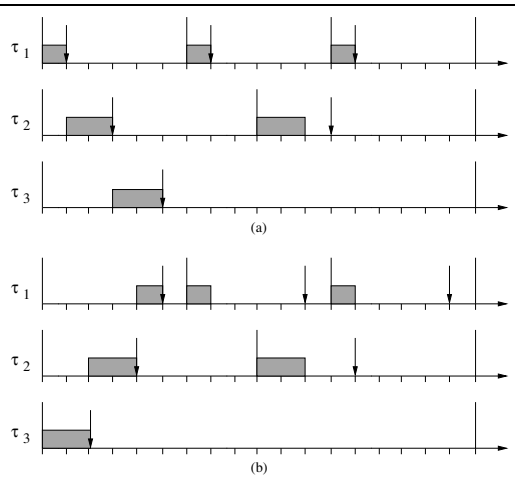

Figure 1. Task set with different deadline assignments.

3. **Problem 3**
   Given a set $\mathcal{T}$ of n periodic tasks, with $U < 1$, that is feasible under EDF, and given a new soft aperiodic job J_a with arrival time r_a, no deadline, and computation time C_a, find the minimum relative deadline D_a^{min} that can be assigned to J_a such that EDF still generates a feasible schedule.

As far as Problem 2 is concerned, it is worth noticing that minimizing the deadline of the one task does not mean consuming all the available processor bandwidth and does not prevent the other deadlines to be reduced. Clearly, the specific order in which the process is applied can make a big difference on the deadline improvement that can be achieved on a task, but this is part of the strategy that can be applied by the user. Figure 1 shows an example in which a set of three tasks is scheduled by EDF with two deadline assignments: starting from the case in which deadline are equal to periods, deadlines were minimized in increasing order for case (a) and in decreasing order for case (b). Notice that, although the selected order can make a significant difference in the deadline assignments, minimizing the deadline of the first tasks can still leave a significant space for reducing the deadlines of the remaining tasks. As another remark, it should be obvious that, if some tasks may tolerate a given amount of jitter, their deadlines can be fixed at predefined values and the deadline minimization process can be applied only to those tasks whose deadlines need to be minimized.

For the sake of clarity, the deadline minimization algorithm will be explained for the case of Problem 1, and then extended for the other two problems.

3. Deadline minimization algorithm

To explain the deadline minimization algorithm proposed in this paper, we assume to have an EDF-feasible task set $\mathcal{T}$ consisting of n periodic tasks with arbitrary periods and deadlines, and total utilization $U < 1$. Without loss of generality, let us assume we have to minimize the relative deadline of task τ_x.

The algorithm is iterative and starts by assigning τ_x the minimum possible relative deadline, which is clearly $D_x = C_x$. Then, the feasibility of the task set is checked by using the Processor Demand Criterion [6, 8]. If the task set is feasible with $D_x = C_x$, then the minimum feasible deadline of τ_x has been found, otherwise D_x is incremented by a suitable amount, and the feasibility is checked again. The process of incrementing D_n and checking for feasibility continues until the task set is found to be schedulable. At this point, the current relative deadline assigned to τ_x is the minimum deadline D_x^{min} that guarantees the feasibility of the task set.

Before presenting the algorithm in detail, the following section briefly recalls the EDF guarantee test used to check the schedulability of the task set.

3.1. EDF feasibility test

Since we consider the general case of periodic tasks with deadlines less than, equal to, or greater than periods, the feasibility test is performed using the processor demand criterion, which provides a necessary and sufficient condition for the schedulability of the task set under EDF. For a set of periodic tasks simultaneously activated at time $t = 0$ (i.e., with no activation offset), the processor demand $h(t)$ in an interval $[0, t]$ is the amount of processing time requested by those jobs activated in $[0, t]$ and with deadline less than or equal to t. Then, the feasibility of a task set is guaranteed if and only if, in *any* interval of time, the processor demand does not exceed the available time, that is, if and only if

$$\forall t > 0 \quad h(t) \leq t. \tag{1}$$

Baruah, Rosier, and Howell [6] showed that $h(t)$ can be computed as follows:

$$h(t) = \sum_{i=1}^{n} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i. \tag{2}$$

Baruah, Mok, and Rosier [5] showed that the time instants at which the test has to be performed correspond to those deadlines within the hyperperiod H not exceeding the value

$$L_a = \max \left\{ D_1, \ldots, D_n, \frac{\sum_{i=1}^{n} (T_i - D_i) U_i}{1 - U} \right\}. \tag{3}$$

Hence, the feasibility test for EDF can be summarized by the following theorem.

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE

Theorem 1 *A set of periodic tasks simultaneously activated at time $t = 0$ is schedulable by EDF if and only if $U < 1$ and*

$$\forall t \in \mathcal{S} \quad \sum_{i=1}^{n} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i \leq t. \tag{4}$$

where $\mathcal{S}$ is the set of all task absolute deadlines not exceeding $t_{max} = \min\{L_a, H\}$, that is,

$$\mathcal{S} = \{d_k : d_k \leq \min\{L_a, H\}\}. \tag{5}$$

The complexity of such a feasibility test is pseudo-polynomial. A different upper bound on the number of deadlines that must be checked for feasibility can be determined using the busy period approach. A *busy period* is any interval of time in which the processor is not idle. It is worth observing that an idle time interval can have zero length if the last executed job completes at the same time a new job is released.

In general, a schedule can have several busy periods in the first hyperperiod. However, a set of periodic tasks simultaneously activated at time $t = 0$ is schedulable by EDF if and only if no deadline is missed in the first busy period $[0, L_b]$, which is also the longest one [20, 22].

The value of L_b can be computed using a recursive procedure, which recursively compares the cumulative workload $W(t)$ in the interval $[0, t)$ with the length of the interval. Then, the first busy period length L_b is given by the smallest positive t such that $W(t) = t$. Practically, the cumulative workload in $[0, t)$ is the computation time requested by all the jobs released before t and can be computed as:

$$W(t) = \sum_{i=1}^{n} \left\lceil \frac{t}{T_i} \right\rceil C_i. \tag{6}$$

Hence, the busy period length L_b can be computed by the following recurrent equation, which is stopped when $L^{(k)} = L^{(k-1)}$:

$$\begin{cases} L^{(0)} = \sum_{i=1}^{n} C_i \\ L^{(k)} = W(L^{(k-1)}). \end{cases} \tag{7}$$

Note that, if $U < 1$, then $L_b < H$, but nothing can be said with respect to L_a. Hence, the test can be performed only for those absolute deadlines not exceeding $t_{max} = \min\{L_a, L_b\}$. In conclusion, the EDF feasibility of a periodic task test with arbitrary deadlines and simultaneous activations can be tested by the following theorem.

Theorem 2 *A set of periodic tasks simultaneously activated at time $t = 0$ is schedulable by EDF if and only if $U < 1$ and*

$$\forall t \in \mathcal{S} \quad \sum_{i=1}^{n} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i \leq t. \tag{8}$$

EDF_feasibility_test($\mathcal{T}$)
  $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$

  $\mathcal{D} = \bigcup_{i=1}^{n} \{mT_i + D_i;\ m = 1, 2, \ldots\}$
    $= \{d_1, d_2, \ldots\}$

  $L_a = \max\left\{D_1, \ldots, D_n, \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1-U}\right\}$
  L_b = busy_period()
  $t_{max} = \min\{L_a, L_b\}$

  $\mathcal{S} = \{d_k \in \mathcal{D} : d_k \leq t_{max}\}$

  if $(U > 1)$ **then**
    return ("Unfeasible")
  end if

  for each $(t \in \mathcal{D})$
    $h(t) = \sum_{i=1}^{n} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i$
    if $(h(t) > t)$ **then**
      return ("Unfeasible")
    end if
  end for
  return ("Feasible")
end

Figure 2. Pseudo-code of the EDF feasibility test algorithm.

---

where $\mathcal{S}$ is the set of all task absolute deadlines not exceeding $t_{max} = \min\{L_a, L_b\}$, that is,

$$\mathcal{S} = \{d_k : d_k \leq \min\{L_a, L_b\}\}. \tag{9}$$

The pseudo-code for the EDF feasibility test is shown in Figure 2.

3.2. Computing deadline increments

A simple but inefficient method to find the minimum deadline for τ_x would be to start by setting $D_x = C_x$, and then incrementing D_x by one tick at each iteration, until the task set is found to be schedulable. This algorithm is inefficient because it requires a large number of steps to terminate. The method used in this paper is more efficient since deadline increments are computed based on the processor demand evaluated at a deadline miss. The method is illustrated in the following section using a simple example.

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE

Task	C_i	T_i	D_i
τ_1	10	20	16
τ_2	1	6	3
τ_3	2	6	100

Table 1. Task set parameters.

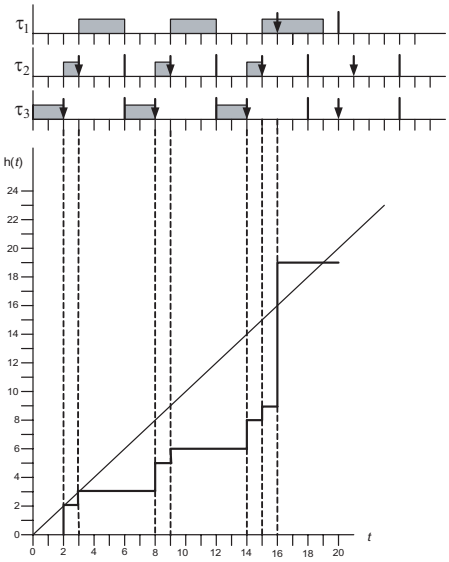

Figure 3. Processor demand for the task set of Example 1 when $D_3 = 2$.

3.3. Example

Consider a set of three periodic tasks whose parameters are given in Table 1. The task set is feasible under the EDF scheduling algorithm, but we want to find the minimum deadline D_3^{min} for τ_3 that preserves feasibility.

The first step is to assign $D_3 = C_3 = 2$ and verify its schedulability by Theorem 2. To follow the reasoning used by the algorithm, Figure 3 illustrates the schedule produced by EDF and the corresponding demand function $h(t)$ when $D_3 = 2$.

As we can see from the figure, at time $t = 16$, the value of $h(t)$ exceeds the available time t. Therefore, the task set is not feasible and D_3 has to be increased to reduce the interference caused by τ_3. Let t^* be the first time at which the deadline miss is detected and let Δ be the amount of execution time exceeding the deadline. We have:

$$\Delta = h(t^*) - t^*. \tag{10}$$

In general, to avoid the deadline miss at time t^*, D_x has to be increased to a value D'_x such that a sufficient number of τ_x instances are removed from the interval $[0, t^*]$. Let K be the minimum number of jobs of task τ_x that must be removed from $[0, t^*]$ to avoid the deadline miss at time t^*. It is easy to see that

$$K = \left\lceil \frac{\Delta}{C_x} \right\rceil. \tag{11}$$

After the removal of K jobs, the new processor demand in $[0, t^*]$ becomes

$$h'(t^*) = h(t^*) - KC_x. \tag{12}$$

Notice that, to remove K instances from $[0, t^*]$, the new absolute deadline d'_x of τ_x must be strictly greater than t^*, but also greater than or equal to $h'(t^*) + C_x$, otherwise τ_x would miss its deadline. Since we are interested in the minimum deadline, d'_x is set exactly at the value $h'(t^*) + C_x$. Hence,

$$d'_x = h'(t^*) + C_x = h(t^*) - (K-1)C_x. \tag{13}$$

The relative deadline of τ_x can be computed as

$$D'_x = d'_x - r^*_x + (K-1)T_x \tag{14}$$

where r^*_x is the last release time of task τ_x before time t^*. It is given by

$$r^*_x = \left\lfloor \frac{t^*}{T_x} \right\rfloor T_x. \tag{15}$$

From equations (11) and (14) we have:

$$D'_x = h(t^*) + (K-1)(T_x - C_x) - r^*_x \tag{16}$$

and using equation (15) we finally have

$$D'_x = h(t^*) + (\left\lceil \frac{\Delta}{C_x} \right\rceil - 1)(T_x - C_x) - \left\lfloor \frac{t^*}{T_x} \right\rfloor T_x. \tag{17}$$

In the specific case of the example, the new relative deadline for τ_3 is found to be $D_3 = 11$. Figure 4 illustrates the schedule produced by EDF and the corresponding demand function $h(t)$ when $D_3 = 11$.

We have shown that, if task τ_x is assigned a relative deadline D'_x, the deadline miss at time t^* is avoided. However, to guarantee the schedulability of the task set, the feasibility test must be performed for all the remaining deadlines in the set $\mathcal{D}$. The pseudo-code of the deadline minimization algorithm, also referred to as the `minD` algorithm, is illustrated in Figure 5.

3.4. Extensions

The `minD` algorithm illustrated in Figure 5 computes the minimum relative deadline of τ_x that preserves the feasibility of the task set. Since no particular order has been assumed for the tasks, the algorithm can be used to minimize

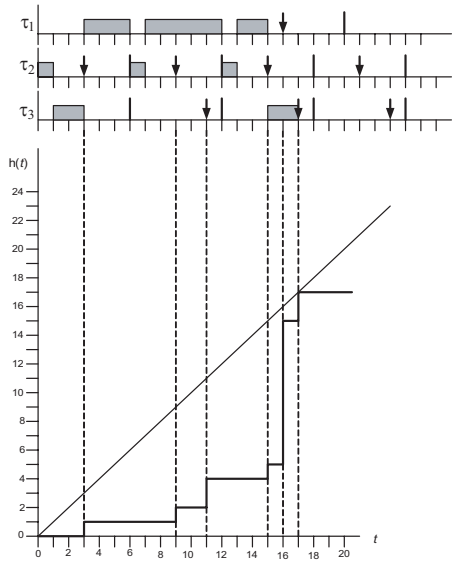

Figure 4. Processor demand for the task set of Example 1 when $D_3 = 11$.

minD_algorithm($\mathcal{T}$, x)
$D_x = C_x$
$\mathcal{S} = \{d_k \in \mathcal{D} : d_k \leq t_{max}\}$

for each $(t \in \mathcal{S})$
$h(t) = \sum_{i=1}^{n} \left\lfloor \frac{t+T_i-D_i}{T_i} \right\rfloor C_i$

if $(h(t) > t)$ **then**
$K = \left\lceil \frac{h(t)-t}{C_x} \right\rceil$
$r = \left\lfloor \frac{t}{T_x} \right\rfloor T_x$
$D_j = h(t) + (K-1)(T_x - C_x) - r$
update_$\mathcal{S}$()
end if
end for

$D_x^{min} = D_x$
return(D_x^{min})
end

Figure 5. Pseudo-code of the deadline minimization algorithm.

the relative deadline of an arbitrary task, as stated in Problem 1.

To solve Problem 2, it is sufficient to iteratively apply the `minD` algorithm to all tasks in the set, in a given order. If V is the ordered array that specifies the desired sequence of task indices, Problem 2 can simply be solved by the following algorithm:

minD_set(V) $\quad \% \; V = ordering\_vector()$

for $(i = 1$ to $n)$
$x = V[i]$
minD_algorithm($\mathcal{T}$, x)
end for
end

Solving Problem 3, that is, finding the minimum relative deadline of an aperiodic job J_a, is also very straightforward. In fact, the only difference with respect to Problem 1 is that the processor demand function $h(t)$ must be computed taking into account the computation time of the aperiodic job. Let r_a be the job arrival time, C_a its computation time, and d_a its absolute deadline, initially set equal to $r_a + C_a$. Clearly, in any interval of time $[0, t]$, the total processor demand is:

$$h(t) = \begin{cases} \sum_{i=1}^{n} \left\lfloor \frac{t+T_i-D_i}{T_i} \right\rfloor C_i + C_a & \text{if } d_a \leq t \\ \sum_{i=1}^{n} \left\lfloor \frac{t+T_i-D_i}{T_i} \right\rfloor C_i & \text{otherwise} \end{cases} \quad (18)$$

When a deadline miss is detected at time t^*, since the periodic task set was originally feasible, the exceeding time $\Delta = h(t) - t$ cannot be larger than C_a. Hence, the deadline miss at time t^* can be avoided by simply setting the absolute deadline of the aperiodic task at $d_a = h(t)$, which means $D_a = d_a - r_a$. Then, the feasibility test must proceed for all deadlines in the set $\mathcal{S}$, where $\mathcal{S}$ must also include d_a. The pseudo-code of the resulting algorithm is illustrated in Figure 6.

4. Simulation results

This section describes a set of simulation experiments that have been conducted to evaluate the behavior and the complexity of the proposed algorithm on different application scenarios, generated through synthetic task sets with random parameters within given ranges and distributions.

To generate a feasible task set of n periodic tasks with given utilization $U_d < 1$, we first generated n random utilizations uniformly distributed in (0,1) and then normalized them to have

$$\sum_{i=1}^{n} U_i = U_d.$$

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE

minD_aperiodic($\mathcal{T}$, J_a)
$d_a = r_a + C_a$
$\mathcal{S} = \{d_k \in \mathcal{D} : d_k \leq t_{max}\}$
$\mathcal{S} = \mathcal{S} \cup \{d_a\}$

for each ($t \in \mathcal{S}$)
$h(t) = \sum_{i=1}^{n} \left\lfloor \frac{t+T_i-D_i}{T_i} \right\rfloor C_i$
if ($t \geq d_a$) **then**
$h(t) = h(t) + C_a$
end if

if ($h(t) > t$) **then**
$d_a = h(t)$
update_$\mathcal{S}$()
end if
end for

$D_a^{min} = d_a - r_a$
return(D_a^{min})
end

Figure 6. Pseudo-code of the algorithm for minimizing the deadline of an aperiodic job.

Then, we generated n computation times as random variables uniformly distributed in $[C_{min}, C_{max}]$, and then calculated the period of each task as

$$T_i = \frac{C_i}{U_i}.$$

Relative deadlines have been computed to generate a feasible schedule as follows: each deadline was initially set at a value $D_i = C_i$ and then incremented by a random amount in $[0, T_i]$ until obtaining a feasible task set.

After generating the task set, we carried out a number of simulation experiments aimed at evaluating the behavior of the proposed algorithm in different application scenarios.

In a first set of experiments, we tested the complexity of the algorithm as a function of the number of tasks. To do that, we generated a task set with fixed utilization $U_d = 0.98$ and measured the number of elementary computational steps to find the minimum deadline of task τ_n. The number of tasks was varied from 10 to 30, and task execution times were randomly generated from 1 to 10. The results of this experiment are illustrated in Figure 7, which reports the average over 10,000 simulation runs.

We also tested the complexity of the algorithm as a function of the total processor utilization, for a fixed number of tasks. Figure 8 shows the number of elementary steps to find the minimum deadline of task τ_n for a set of 20 periodic tasks, having a total utilization ranging from 0.5 to 0.9.

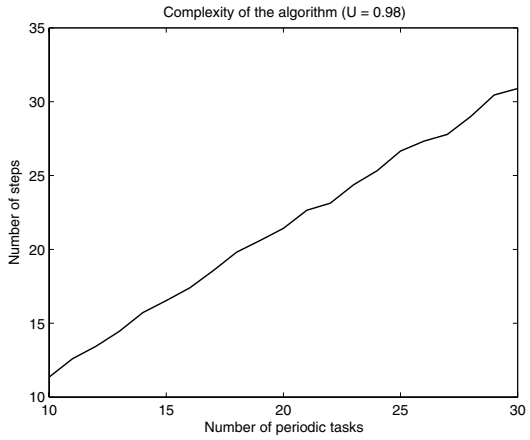

Figure 7. Number of steps of the minD algorithm as a function of the number of tasks ($U_d = 0.98$).

To evaluate the performance of the algorithm in reducing the deadline of a task, we tested the average deadline improvement as a function of the number of tasks. The improvement for a task τ_i is computed as the normalized ratio β_i of the actual deadline reduction and the maximum possible reduction, equal to $D_i - C_i$. That is

$$\beta_i = \frac{D_i - D_i^{min}}{D_i - C_i}.$$

Thus, $\beta_i = 0$ means that no reduction was achieved by the algorithm on task τ_i, so the task will run with its original deadline, whereas $\beta_i = 1$ means that the maximum possible reduction was achieved, so the task will run with a new relative deadline equal to its computation time. The results of this test are shown in Figure 9, for n ranging from 2 to 30 tasks.

In a second set of experiments, we compared the `minD` algorithm with the Improved Total Bandwidth Server (TB*), introduced by Buttazzo and Sensini [9] for computing the minimum deadline of aperiodic jobs. In all the cases, we measured the number of steps required by the two algorithms to find the minimum deadline of an aperiodic job with random arrival time r_a and given computation time C_a. The simulations have been done to test the dependency of the algorithm complexity from the number of tasks, the task set utilization, and the job execution time C_a.

We briefly recall that TB* starts assigning the aperiodic job an initial deadline given by the Total Bandwidth Server

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE

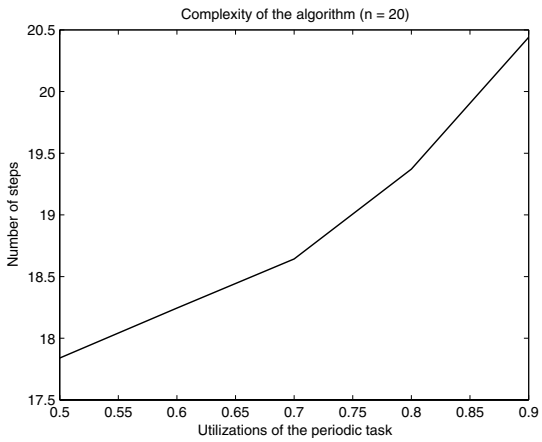

Figure 8. Number of steps to find MinD when the task set has 20 task while utilization is increased.

rule (TBS rule):

$$d_a = \max\{d^{prev}, r_a\} + C_a/(1-U) \qquad (19)$$

where d^{prev} is the absolute deadline assigned to the previous aperiodic request, and then tries to shorten it as much as possible, while preserving the feasibility of the task set. Comparing the complexity of the two approaches is interesting, because both algorithms are optimal (i.e., they find the shortest possible deadline that preserves feasibility) and run with pseudo-polynomial complexity, but they work in opposite directions. In fact, while `minD` starts with $D_a = C_a$ and increases D_a until a feasible schedule is found, TB* starts from a safe large deadline (given by the Total Bandwidth Server rule [21]) and tries to decrease it as much as possible. As a consequence, the two algorithms should exhibit a different average complexity for different task set scenarios. Identifying such scenarios is important, since it would allow a developer to select the most efficient method for a given real-time system, or even switch between the two algorithms if dealing with dynamic environments.

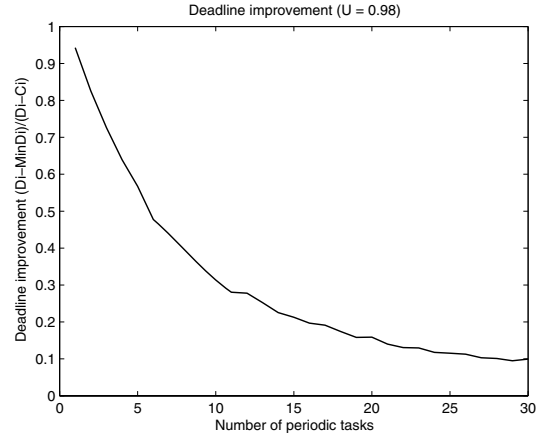

Figure 9. Average deadline improvement as a function of the number of tasks.

The graphs in Figure 10 show the number of steps required by the two algorithms as a function of the task set utilization, for a set of 20 periodic tasks and for a job with average execution time $C_a = 7$, which is the average execution time used for the periodic tasks (computation times of periodic tasks were generated in the range $[C_{min}, C_{max}]$, with $C_{min} = 2$ and $C_{max} = 12$). It is worth noting that `minD` is more efficient than TB* for low and medium task set utilizations ($U < 0.87$), whereas TB* becomes superior for high workloads. In fact, for low periodic utilizations, there is enough idle time in the schedule to allow aperiodic jobs to execute with a short relative deadline. Short deadlines are immediately found by `minD`, which starts searching from $D_a = C_a$. For high periodic utilizations, the optimal aperiodic deadline is much longer, and thus closer to the initial one used by TB*.

Figure 11 shows the behavior of the two algorithms as a function of the number of tasks, when the task set utilization is $U = 0.87$ (the crossing point in Figure 10) and aperiodic jobs have an average execution time $C_a = 7$. Again, we observe that, for less than 20 tasks, `minD` is more efficient than TB*. In fact, as the number of tasks increases, the idle time available in the schedule is fragmented into smaller pieces, and the optimal aperiodic deadline increases, getting closer to the initial value used by TB*.

Finally, Figure 12 shows the number of steps as a function of the aperiodic execution time C_a, for a set of 20 periodic tasks with total utilization $U_d = 0.87$. It is worth noting that TB* is very sensitive to the value of C_a, because increasing C_a also increases the initial deadline assignment given by the TBS rule, as stated by equation (19). Hence, for short jobs, the initial guess made by the TB* is close to the optimal value, while for long jobs the initial guess gets too far away. On the other hand, `minD` is not much affected by the value of C_a because, when C_a increases, its initial guess increases towards the optimal deadline.

In summary, the most interesting result derived from the simulation experiments is that none of the two algorithms dominates the other, but their runtime behavior depends on the task set parameters and size. For small task sets (i.e., less than 20 tasks) and low/medium processor workloads

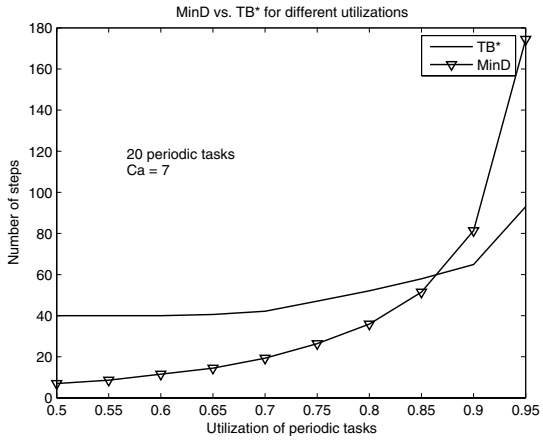

Figure 10. Average number of steps as a function of the task set utilization.

(i.e., $U < 0.87$), the `minD` algorithm is more efficient than TB* in computing the minimum deadline of an aperiodic job, whereas TB* is more effective for large task sets and high utilizations. Moreover, `minD` is also more efficient than TB* when the execution time of aperiodic jobs is larger than those of periodic tasks.

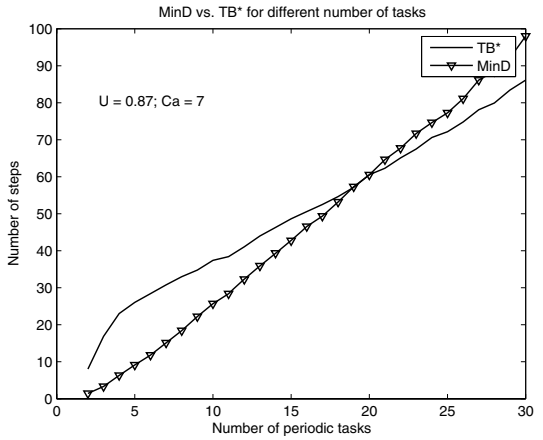

Figure 11. Average number of steps as a function of the number of periodic tasks.

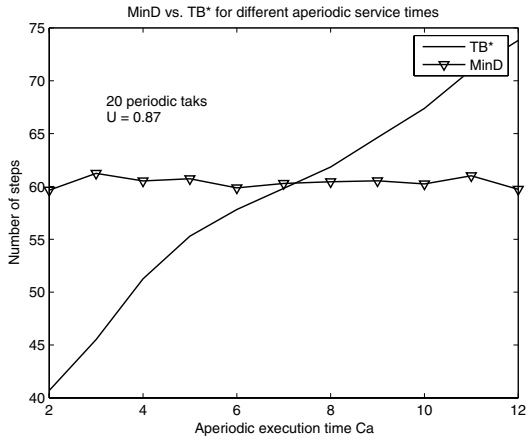

Figure 12. Average number of steps as a function of the aperiodic execution time.

5. Conclusions and future work

In this paper, we presented an algorithm for computing the shortest deadline of a task, while preserving the feasibility of the task set. The algorithm can be applied to periodic and aperiodic tasks and can be useful to reduce input-output jitter in control applications. Although the algorithm has a pseudo-polynomial complexity, extensive simulation experiments showed that it can be effectively used on-line in most practical situations. For the case of aperiodic jobs, the proposed approach has been compared with the Improved Total Bandwidth server introduced by Buttazzo and Sensini [9]. Simulation experiments showed that no algorithm dominates the other, rather their behavior significantly depends on the application parameters. In general, `minD` is more efficient than TB* in computing the minimum deadline of an aperiodic job when the task set is small, the load is not very high, and aperiodic jobs have long computation times.

In addition to the efficiency, however, the proposed algorithm is more general than existing techniques, since it applies to tasks with arbitrary deadlines and arbitrary activation patterns.

As a future work, we plan to adapt the algorithm to reduce the deadlines of several tasks at the same time, without following a predefined order. For example, a system could have different classes of real-time activities, and some class might require a uniform improvement with respect to the others. It would be interesting to find an efficient solution minimizing the deadlines of a task subset provided that deadlines have the same *scaling factor*.

We are also planning to implement the deadline minimization algorithm in the Shark real-time kernel [1, 14] for

Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)
0-7695-2676-4/06 $20.00 © 2006 IEEE

testing the actual performance improvement of real-world control applications.

Acknowledgement

This work has been partly funded by the CERES research profile grant from The Knowledge Foundation.

References

[1] Shark. http://shark.sssup.it.

[2] K. J. Astrom and B. Wittenmark. *Computer Controller Systems: Theory and Design*. Prentice-Hall, 1984.

[3] P. Balbastre, I. Ripoll, and A. Crespo. Optimal deadline assignment for periodic real-time tasks in dynamic priority systems. In *18th IEEE Euromicro Conference on Real-Time Systems*, 2006.

[4] S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari. Scheduling periodic task systems to minimize output jitter. In *IEEE International Conference on Real-Time Computing Systems and Applications*, pages 62–69, 1999.

[5] S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *IEEE Real-Time Systems Symposium*, pages 182–190, 1990.

[6] S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2:301–324, 1990.

[7] S. A. Brandt, S. A. Banachowski, C. Lin, and T. Bisson. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *IEEE Real-Time Systems Symposium*, 2003.

[8] G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications - Second Edition*. Springer, 2005.

[9] G. C. Buttazzo and F. Sensini. Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time environments. *IEEE Transactions on Computers*, 48(10):1035–1052, 1999.

[10] G. C. Buttazzo, M. Velasco, P. Martí, and G. Fohler. Managing quality-of-control performance under overload conditions. In *16th Euromicro Conference on Real-Time Systems (ECRTS 2004)*, pages 53–60, 2004.

[11] A. Cervin. *Integrated Control and Real-Time Scheduling*. PhD thesis, Department of Automatic Control, Lund University, Lund, Sweden, April 2003.

[12] A. Cervin, B. Lincoln, J. Eker, K.-E. Årzén, and G. Buttazzo. The jitter margin and its application in the design of real-time control systems. In *International Conference on Real-Time and Embedded Computing Systems and Applications*, 2004.

[13] D. Ferrari and D. C. Verma. A scheme for real-time channel establishement in wide-area networks. *IEEE Journal of Selected Areas in Communications*, 8(3):368–379, 1990.

[14] P. Gai, L. Abeni, M. Giorgi, and G. Buttazzo. A new kernel approach for modular real-time systems development. *13th Euromicro Conference on Real-Time Systems (ECRTS 2001)*, 2001.

[15] H. Hoang, S. Karlsson, and M. Jonsson. Minimum edf-feasible deadline calculation with low-time complexity. In *Real-time systems symposym, wip*, 2005.

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

[17] P. Marti. *Analysis and Design of Real-Time Control Systems with Varying Control Timing Constraints*. PhD thesis, July 2002.

[18] P. Marti, J. Fuertes, G. Fohler, and K. Ramamritham. Jitter compensation for real-time control systems. In *Real-Time Systems Symposium*, pages 39–48, 2001.

[19] M. D. Natale and J. A. Stankovic. Scheduling distributed real-time tasks with minimum jitter. *IEEE Transactions on Computers*, 49(4):303–316, 2000.

[20] M. Spuri. Analysis of deadline schedule real-time systems. Technical Report 2772, Inria, France, 1996.

[21] M. Spuri and G. C. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Real-time Systems*, 10(2):179–210, 1996.

[22] J. A. Stankovic, K. Ramamritham, and M. Spuri. *Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms*. Kluwer Academic Publishers, 1998.

[23] Q. Zheng and K. G. Shin. On the ability of establishing real-time chennals in point-to-point packet-switched networks. *IEEE Transactions on Communications*, 42(2/3/4):1096–1105, 1994.