Optimal deadline assignment for periodic real-time tasks in dynamic priority systems

Patricia Balbastre, Ismael Ripoll and Alfons Crespo*
Department of Computer Engineering
Technical University of Valencia, Spain
{patricia,iripoll, alfons}@disca.upv.es

Abstract

Real-time systems are often designed using a set of periodic tasks. Task periods are usually set by the system requirements, but deadlines and computation times can be modified in order to improve system performance. Sensitivity analysis in real-time systems has focused on changes in task computation times, using fixed priority analysis. Only a few studies deal with the modification of deadlines in dynamic-priority scheduling. The aim of this work is to provide a sensitivity analysis for task deadlines in the context of dynamic-priority, pre-emptive, uniprocessor scheduling. In this paper, we present a deadline minimisation method that achieves the maximum reduction. As undertaken in other studies concerning computation times, we also define and calculate the critical scaling factor for task deadlines. Our proposal is evaluated and compared with other works in terms of jitter. The deadline minimisation can be used to strongly reduce jitter of control tasks, in a real-time control application.

1 Introduction

1.1 Motivation

Real-time systems are often designed using a set of periodic activities running on top of a real-time operating system. For instance, in control systems, the correct behaviour of the closed-loop controller requires that the system meet timing constraints such as periods and latencies, which are expressed as deadlines. Widely established scheduling policies, such as Rate [18] or Deadline Monotonic [16] for fixed priorities and Earliest Deadline First (EDF) for dynamic priorities assign priorities according to timing parameters, sampling periods and deadlines respectively.

*This work was supported by the Spanish Government Research Office (CICYT) under grant DPI2005-09327-C02-02

Both timing parameters, periods and deadlines, are susceptible to adjustment during the design phase of the control system. While modifications in the periods may require the redesigning of the controller in order to adjust it to the new rate, deadline adjustments do not. A modification in the deadline of a control activity can, for instance, increase or decrease the task priority if Deadline Monotonic criteria is used in a fixed priority scheme. When using the EDF scheduling policy, a deadline modification does not have a direct effect on the task priority as in the case of fixed priority scheduling. The effect is acknowledged in a variation in the task response time throughout the task activations. In both cases, this effect can be measured through the output jitter of the task.

In control systems, the influence of jitter on performance quality is not always easy to analyse. From a control perspective, sampling jitter and latency jitter can be interpreted as disturbances acting upon the control system. The input-output latency decreases the stability margin and limits the performance of the system. If the jitter and the latency are small, then these can be overlooked. Otherwise, they should be accounted for in the control design or, if possible, compensated for at run-time. Further information regarding these issues is detailed in [9]. Output jitter has in general a negative effect on the control performance. In general, output jitter has a negative effect on the control performance. This can be significant depending on the term known as control effort [1]. The control effort measures how sensitive a control task is to time delays. The reduction of the output jitter and the subsequent improvement in the control performance, is directly associated to the reduction of task deadlines.

In other real-time applications, as in multimedia systems, a bound of output jitter is needed in order to have an acceptable quality of service. Delay jitter can be eliminated by buffering at the receiver, but the amount of buffer space required can be reduced if the network provides some guarantees about jitter [23].

The output jitter reduction can be regarded as a more

general question related to deadline minimisation. Questions such as: "By how much can a task deadline be reduced?", or, "Which is the maximum deadline reduction factor needed to maintain the schedulability of a set of tasks?, can be of interest for a set of applications. Most of the work on feasibility analysis in real-time systems provides a yes or no answer, in other words, whether the system is feasible or not. However, given a schedulable set of tasks, is it possible to determine the smallest deadline or period, or the largest computation time, which will still render the system schedulable? This kind of analysis is referred to in the literature as *sensitivity analysis*.

1.2 Related work

Sensitivity analysis has focused on permissible changes on tasks WCET, mainly because this has been applied to fault tolerance design. In this sense, [15, 24, 20] define the *Critical scaling factor*, as the largest possible scaling factor for task computation times while the task set to remain schedulable. These works assume Rate Monotonic priority assignment and deadlines equal to periods.

Sensitivity analysis for computation times using EDF scheduling has been performed in [2]. Moreover, the *Optional Computation Window* (OCW) is defined as the possibility that a task can execute in n activations in a window of m consecutive invocations with an increased computation time than the initially assigned. Deadlines less than or equal to periods are assumed.

Sensitivity analysis for task periods can be found in [6]. In this paper, periods are modeled as springs with given elastic coefficients and minimum lengths. Requested variations in task execution rates or overload conditions are managed by changing the rates based on the spring elastic coefficients. This work assumes deadlines equal to periods and EDF scheduling. This technique is very useful in multimedia applications, since in control systems a modification in the period requires to redesign the controller.

Regarding deadline sensitivity analysis, as far as the authors are aware, there are no studies which focus on this topic, that is, to find the minimum deadlines while maintaining the schedulability of the system. However, there are some papers which assign new deadlines to tasks in order to achieve a secondary objective. In [4] the goal is to minimise output jitter and in [8] new deadlines for control tasks are calculated in order to guarantee close-loop stability of real-time control systems.

Regarding output jitter, several studies use scheduling solutions in order to reduce this. In [17] the difference between two consecutive finishing times of the same task is required to be bounded by a specific value. In [11] a new scheduling approach is presented to minimise jitter in distributed real-time systems. In [19, 14], high priority tasks are used to reduce the jitter. In [13] new deadlines are assigned to tasks scheduled under EDF, using an integer linear program. It can be noted that, in all these studies dealing with deadline reduction, this reduction is not optimal in the sense that deadlines can be minimised any further.

In order to minimise control jitter, other strategies have been used. In [10, 7, 3] subtask partition is proposed. Furthermore, in [10] and [3] control jitter is both reduced and determined off-line the variation range, enabling control designers to take decisions before executing the control system in order to achieve a better performance.

1.3 Contributions of this paper

As explained in the prevoius section, although there are works that deal with deadline reduction, there are not works that focus on finding the minimum deadline of a task while preserving feasibility in dynamic priority systems. The aim of this work is to provide a sensitivity analysis for task deadlines in the context of dynamic-priority, preemptive, uniprocessor scheduling. Two different kind of analysis are carried out:

- Finding the maximum reduction for one task deadline. Note that the maximum reduction implies that a lower deadline makes the system not feasible. We will demonstrate that, in this case, both output jitter and worst case response times of tasks are known off-line. Therefore, there is no need to calculate them.

- Finding the maximum reduction for all task deadlines, that is, the critical scaling factor for task deadlines. This implies that a lower scaling factor makes the system not feasible.

This deadline reduction can be used subsequently to reduce output jitter, control jitter and response times of tasks.

1.4 Outline

The remainder of this paper is organised as follows: Section 2 introduces the computational model and the assumptions used. Section 3 describes the deadline minimisation algorithm. In section 4 the critical scaling factor for task deadlines is presented. An example to illustrate how the deadline minimisation works is presented in Section 5. We have implemented the deadline minimisation algorithm, and the critical scaling factor calculation, and we have conducted some experiments in order to assess the validity and effectiveness of our proposals. The results and the comparison with other related works are presented in Section 6. Finally, in Section 7 some conclusions are highlighted and future lines of research are discussed.

2 System model and notation

Let $\tau = \{T_1, ..., T_n\}$ be a periodic task system scheduled under EDF. Each task $T_i \in \tau$ has the following temporal parameters $T_i = (C_i, D_i, P_i)$. Where P_i is the period; D_i is the deadline relative to the start time; and C_i is the worst case execution time, $C_i \leq D_i$. In this work, we consider that $D_i \leq P_i$.

Each task T_i produces a sequence of jobs $J_{i,j} = (r_{i,j}, C_i, d_{i,j})$, where $r_{i,j}$ denotes the arrival time (or request time) of the j^{th} job of the task T_i, and $d_{i,j} = r_{i,j} + P_i$ is the absolute deadline. The sequence of jobs $J_{i,j}$ are arranged by absolute deadline in ascending order. Subsequently, a task set can be seen as an infinite number of activations.

Definition 1 *Let* $G_\tau(t) = \sum_{i=1}^{n} C_i \left\lceil \frac{t}{P_i} \right\rceil$. *It denotes the maximum cumulative execution time requested by activations of* τ *whose arrival times are before time* t.

Note that $G_\tau(t)$ function does not depend on task deadlines. Changing the deadlines of the tasks have no impact on the value of this function.

Definition 2 *[5] Let* $H_\tau(t) = \sum_{i=1}^{n} C_i \left\lfloor \frac{t+P_i-D_i}{P_i} \right\rfloor$. *It denotes the maximum cumulative execution time requested by jobs of* τ *whose absolute deadlines are less than or equal to* t.

Feasibility test for EDF consists of checking that $\forall t \leq \mathcal{R} \quad H_\tau(t) \leq t$, where $[0, \mathcal{R})$ is the ICI of the task set τ. ICI stands for *Initial Critical Interval* and it is the first interval where the processor is not idle [21]. $\mathcal{R}$ is the smallest $t > 0$ such that $t \leq G_\tau(t)$. In fact, there is no need to confirm that $H_\tau(t) \leq t$ applies for all $t \leq \mathcal{R}$ to determine whether τ is feasible, except in the designated scheduling points [15].

Definition 3 *The set* $\mathcal{S}_\tau$ *of scheduling points for* τ *is defined by:*

$$\mathcal{S}_\tau = \{d_{i,j} \quad / \quad 1 \leq i \leq n\,, \quad 1 \leq j \leq k_i \ \wedge \ d_{i,j} \leq \mathcal{R}\}$$

where $k_i = \left\lceil \frac{\mathcal{R}}{P_i} \right\rceil$ *denotes the number of activations of* T_i *in* $[0, \mathcal{R})$.

Definition 4 *Let* $\alpha_i = 1 - \frac{D'_i}{D_i}$ *be the deadline reduction factor of* T_i *where* $C_i \leq D'_i \leq D_i$

Definition 5 *Let* $\alpha_i^{max} = 1 - \frac{C_i}{D_i}$ *be the maximum deadline reduction factor of* T_i

Definition 6 *Let* Δ^* *be the* critical scaling factor *of* τ. *It denotes the smallest scaling factor for task deadlines, above which the task set remains schedulable for all* $i \quad / \quad T_i = (C_i, \Delta D_i, P_i)$, *where* $\Delta \geq \Delta^*$.

If τ is a schedulable task set, then $\Delta^* \leq 1$, otherwise $\Delta^* > 1$. Note that $\alpha_i = 1 - \Delta^*$.

3 Deadline minimisation method

This section describes theory and algorithm used to reduce a task deadline until its minimum value is reached. In other words if we continue to reduce it, the system becomes unfeasible.

The theorems presented in this section are based on the $H_\tau(t)$ function throughout the ICI. Note that if a task deadline is modified, the ICI does not change (this depends on task periods and computation times) and, whenever $H_\tau(t) < t$ in $[0, \mathcal{R})$, system feasibility is assured. Moreover, while calculating the minimum deadline of a task there is no need to re-calculate $H_\tau(t)$ function.

First of all, we will calculate the minimum deadline for the first job of a task and next, the minimum deadline of the task itself.

Theorem 1 *Let* $\tau = \{T_1, ...T_i, ...T_n\}$ *be a feasible set of* n *periodic tasks. Let* $J_{i,1}$ *be the first job of* T_i. *If there exists a time* t_1 *such that* $C_i \leq t_1 \leq D_i$ *and* $H_\tau(t_1) = H_\tau(D_i - \epsilon) > t_1 - C_i$ *then the sequence of jobs* $J_{1,1}, .., J'_{i,1}, .., J_{n,k_n}$ *with* $J'_{i,1} = \{0, C_i, H_\tau(t_1) + C_i\}$ *is schedulable.*

Proof Let $d'_{i,1} = H_\tau(t_1) + C_i$ be the absolute deadline of $J'_{i,1}$. We need to look at in which scheduling points $H_\tau(t)$ changes. In Figure 1 t_1, D_i and $d'_{i,1}$ are represented for a better understanding of the theorem.

There are no scheduling points in the time interval $[t_1, D_i)$ since $H(t)$ is a non-decreasing step function that changes its value only at scheduling points, and $H_\tau(t_1) = H_\tau(D_i - \epsilon)$. Both $H_{\tau'}(t)$ (the new $H_\tau(t)$ function with $J'_{i,1}$ instead of $J_{i,1}$) and $H_\tau(t)$ are the same function for $t \leq t_1$ and $t > D_i$ due to the fact that the set of scheduling points remains the same except for job $J'_{i,1}$ which changes from D_i to $d'_{i,1}$.

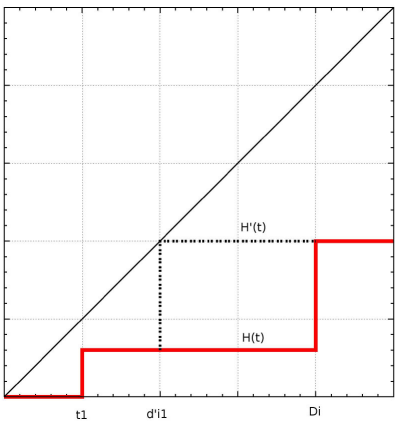

Figure 1. Graphical representation of theorem 1

Then, the system is feasible in $[0, t_1] \cup [D_i, \mathcal{R}]$. It is also feasible in $[t_1, d'_{i,1}) \cup (d'_{i,1}, D_i]$ since the condition $H_\tau(t_1) = H_\tau(D_i - \epsilon)$ guarantees that there are no more scheduling points. It only remains to prove that $J'_{i,1}$ is schedulable in $t = d'_{i,1}$.

The new H_τ function in $d'_{i,1}$ will be

$$H'_\tau(d'_{i,1}) = H_\tau(t_1) + C_i$$

Since $d'_{i,1} = H_\tau(t_1) + C_i$, it follows that

$$\begin{aligned} H'_\tau(d'_{i,1}) &= d'_{i,1} + C_i - C_i \\ &= d'_{i,1} \end{aligned}$$

∎

Corollary 1 *If it does not exists such t_1 then $d'_{i,1} = C_i$ and the systems is schedulable.*

Proof This situation can occur when:

- There are not scheduling points in $[0, D_i)$. In this case, clearly C_i is a valid deadline for $J_{i,1}$.
- Or it exists a scheduling point with enough slack, that is $\exists t_2 \quad / \quad H_\tau(t_2) = H_\tau(D_i - \epsilon) \leq t_2 - C_i$ In this case, τ is schedulable in t_2 since $H'_\tau(t_2) = H_\tau(t_2) + C_i \leq t_2$. It is also schedulable in C_i since there are no scheduling points in $[0, C_i)$ and $H_\tau(C_i) = C_i$. ∎

Corollary 2 *$d'_{i,1}$ is the minimum deadline for job $J_{i,1}$.*

Proof Clearly, if $d'_{i,1} = C_i$, this is the minimum deadline for $J_{i,1}$. Else if $d'_{i,1} = H_\tau(t_1) + C_i$, then $d'_{i,1} - \epsilon$ will render the set of jobs unfeasible. We need to look at schedulability in $t_{min} = d'_{i,1} - \epsilon$.

$$H'_\tau(t_{min}) = H_\tau(t_1) + C_i$$

Since $t_{min} = d'_{i,1} - \epsilon = H_\tau(t_2) + C_i - \epsilon$ and substituting $H_\tau(t_1)$ in the previous equation:

$$\begin{aligned} H'_\tau(t_{min}) &= t_{min} + C_i - C_i + \epsilon \\ &= t_{min} + \epsilon \end{aligned}$$

and then, τ is not schedulable in t_{min}. ∎

The previous theorems show how to calculate the minimum deadline for the first job of a task. However, it does not imply that all the job deadlines can be reduced the same amount. Next definition extends the calculation of $d'_{i,1}$ for any job $J_{i,j}$.

Definition 7 *Let $d'_{i,j} = H_\tau(t_1) + C_i$*
where $r_{i,j} + C_i \leq t_1 \leq d_{ij}$ and $H_\tau(t_1) = H_\tau(d_{ij} - \epsilon) > t_1 - C_i$

If such t_1 does not exists, then $d'_{i,j} = C_i$

It is important to note that $d'_{i,j}$ (when $1 < j \leq k_i$) is not the minimum deadline of $J_{i,j}$. Although 1 is valid in the sense that $H_\tau(t) \leq t$, schedulability of τ is not guaranteed. This is due to the fact that $H(t)$ is defined for periodic tasks, whose job deadlines have the same value in each activation. However, by theorem 1 (extended for a job $J'_{i,j} = \{r_{i,j}, C_i, d'_{i,j}\}$ instead of $J'_{i,1}$), we can assure that $H_\tau(d'_{i,j}) = d'_{i,j}$. We will base the next theorem on this result to calculate the minimum deadline of a task.

Theorem 2 *Let $\{J'_{i,1}, .., J'_{i,k_i}\}$ be an ordered sequence of jobs of T_i in $[0, \mathcal{R})$, where $J'_{i,j} = \{r_{i,j}, C_i, d'_{i,j}\}$. Then, $\tau' = \{T_1, .., T'_i, .., T_n\}$ with $T'_i = \{C_i, max_{j=1}^{k_i}(d'_{i,j} - r_{i,j}), P_i\}$ is schedulable.*

Proof Suppose that jobs belonging to T_i are arranged in ascending order by its relative deadline ($d'_{i,j} - r_{i,j}$). This results in $max_{j=1}^{k_i}(d'_{i,j} - r_{i,j}) = d'_{i,k_i} - r_{i,k_i}$. As job deadlines are calculated in accordance with theorem 1, then in every $t_j = d'_{i,j}$, it can be assured that $H_\tau(t_j) = t_j$.

If a new deadline of task T_i is chosen as $D_i = d'_{i,k_i-1} - r_{i,k_i-1}$ so that $d'_{i,k_i-1} - r_{i,k_i-1} < d'_{i,k_i} - r_{i,k_i}$, let us determine if T_i is schedulable at its scheduling points:

- If $j = k_i - 1$ then $H_\tau(t_j) = t_j$.
- If $j < k_i - 1$ then $H_\tau(t_j) = t_j + d'_{i,k_i-1} - d'_{i,j} - r_{i,k_i-1} + r_{i,j}$. As $d'_{i,k_i-1} - r_{i,k_i-1} > d'_{i,j} - r_{i,j}$, then $H_\tau(t_j) < t_j$.
- If $j = k_i$ then $H_\tau(t_j) = t_j + d'_{i,k_i-1} - d'_{i,k_i} - r_{i,k_i-1} + r_{i,k_i}$. As $d'_{i,k_i-1} - r_{i,k_i-1} < d'_{i,j} - r_{i,j}$, then $H_\tau(t_j) > t_j$.

As a consequence, the maximum relative deadline for jobs, calculated as described in theorem 1, has to be chosen, otherwise the system will not be schedulable. ∎

Corollary 3 *Let $WCRT_i$ be the worst case response time of task T_i. Then,*

$$WCRT_i = D_i^{min}$$

where $D_i^{min} = max_{j=1}^{k_i}(d'_{i,j} - r_{i,j})$.

Proof We know that $H_\tau(D_i^{min}) = D_i^{min}$. Then, we can assure that in $t = D_i^{min}$ a certain task will finish its execution at its deadline. The only case in which this does not occur is when two or more tasks have their deadline arrival at D_i^{min}, that is, when a multiple scheduling point exists. Let's see that if $D_i^{min} = max_{j=1}^{k_i}(d'_{i,j} - r_{i,j})$, in $t = D_i^{min}$ then no other scheduling points are present.

Let t_j be a scheduling point of T_j, and let's suppose that in t_j there is enough slack time for T_i, so $t_j - H_\tau(t_j) = C_i$. Let t_3 be a scheduling point with $t_3 - H_\tau(t_3) \leq C_i$ and $t_3 \leq t_j$. By theorem 1, $t_{min} = D_i^{min} = H_\tau(t_3) + C_i$.

As t_3 and t_j are two consecutive scheduling points:

$$H_\tau(t_j) - H_\tau(t_3) \geq C_i$$

Therefore, $H_\tau(t_3) < H_\tau(t_j)$, since $C_i \neq 0$. Then

$$t_j \geq H_\tau(t_j) + C_i$$
$$t_j > H_\tau(t_3) + C_i$$
$$t_j > t_{min}$$

Hence, it is not possible that the resulting D_i^{min} coincides with another existing scheduling point. This means that, as in D_i^{min} there is no slack time, the worst case response time of T_i is known and equal to the minimum deadline of T_i. ∎

Under fixed priority scheduling (without offsets), the problem of finding the worst case response time of a task is equivalent to the problem of calculating the response time of that task of the first activation. This is not longer true in EDF. Instead, the worst case response time calculation requires the analysis of several different scenarios, which has pseudo-polynomial time complexity [22]. For this reason, we want to remark the importance of these results. The method presented in this section does calcualte the minimum deadline for a task and also it solves the problem of computing the WCRT under EDF.

Based on the previous theorems, an algorithm to minimise the deadline of a periodic task T_i (Table 1) has been developed. The algorithm starts at time $t = jP_i + D_i$ and computes the slack time in the immediate previous scheduling point of t. If the slack is greater than C_i, the algorithm continues backwards to the next scheduling point; else $d'_{i,j}$ of $J_{i,j}$ is calculated. The procedure is repeated for all jobs of T_i whose deadline is in the interval $[0, \mathcal{R})$. Once the deadline minimisation algorithm is applied to a task T_i, its deadline reduction factor is:

$$\alpha_i = 1 - \frac{D_i^{min}}{D_i}$$

This algorithm has pseudo-polynomial time complexity in the size of the problem instance. However, as stated in [4], this bound is quite reasonable in practice whenever the system has a relative small utilization. Moreover, minor changes on some periods may reduce drastically the complexity to polynomial time [12].

The deadline minimisation algorithm shows how to minimise the deadline of one task in the set. When the goal is to minimise the deadline of more than one task, two alternatives are proposed:

- When all the tasks have the same importance, deadlines of all tasks will be reduced with the same ratio. Thus, we must calculate the *critical scaling factor*, which is the central issue which is discussed in detail in section 4.

Table 1.

1.	**Deadline minimisation algorithm:**
2.	input: τ, T_i;
3.	$\mathcal{R} \leftarrow Calculate\_ICI(\tau)$;
4.	$H_\tau(t) = \sum_{j=1}^{n} C_j \left\lfloor \frac{t+P_j-D_j}{P_j} \right\rfloor$;
5.	$deadline \leftarrow C_i$
6.	$k_i \leftarrow \left\lceil \frac{\mathcal{R}}{P_i} \right\rceil$
7.	$D_i^{min} \leftarrow 0$
8.	for s in $0..(k_i - 1)$ loop
9.	$t \leftarrow (sP_i) + D_i$;
10.	deadline = C_i;
11.	while $t > sP_i + C_i$
12.	if $t = (\lceil \frac{t}{P_j} \rceil - 1)P_j + D_j$ then
13.	$slack = t - H_\tau(t)$
14.	if $slack < C_i$
15.	$deadline = H_\tau(t) + C_i - sP_i$;
16.	exit-while;
17.	end-If;
18.	end-If;
19.	$t \leftarrow t - 1$;
20.	end-while;
21.	$D_i^{min} = max(D_i^{min}, deadline)$;
22.	end-loop;
23.	$D_i = D_i^{min}$;

- When there are some tasks more important than others, these tasks are selected for reducing their deadlines.

In the second case, the solution is to apply the deadline minimisation algorithm to the most important task, update its deadline to the minimum value obtained and repeat the previous steps to the next important task. The order in which tasks are chosen determines the amount of deadline reduction of each one. Clearly, if a task is minimised in second place, its deadline reduction (α_i) may be not that large as if it were minimised in first place. The task selection criteria should follow some semantic approach. In [1], a criteria for determining the importance of a control task based on a the control effort is used. The control effort measures the effort required for the controller to fulfil the control specification. A task with higher control effort is more sensible to the output jitter and can produce a performance degradation of the control response. In this case, the criteria consists on the selection of the task with higher control effort first.

3.1 Output jitter analysis

As stated in [4], *output jitter* refers to the variation between the inter-completion times of successive jobs of the same task. This section presents the output jitter analysis of a deadline minimised task. This analysis has the advantage that can be made off-line, due to the property $WCRT_i = D_i^{min}$. And, due to the same property, this analysis is not a bound but the exact value for output jitter.

The minimum and maximum separation between successive completions of T_i are:

$$p_i^{max} = P_i + D_i^{min} - C_i$$
$$p_i^{min} = P_i$$

Hence, the *absolute jitter* of T_i is:

$$\begin{aligned} AbsJitter(T_i) &= max(p_i^{max} - P_i, P_i - p_i^{min}) \\ &= max(D_i^{min} - C_i, 0) \\ &= D_i^{min} - C_i \end{aligned}$$

When the deadline minimisation algorithm achieves the maximum reduction factor ($\alpha_i = \alpha_i^{max}$) for T_i, we say that task T_i is *jitter-free*.

The *relative jitter* of T_i can be obtained as a fraction of its period:

$$RelJitter(T_i) = \frac{AbsJitter(T_i)}{P_i} = \frac{D_i^{min} - C_i}{P_i}$$

4 Critical scaling factor calculation

If we want to minimise all task deadlines, but we do not have any preference regarding which task deadline requires the greatest level of reduction, then we have to calculate the critical scaling factor Δ^*.

The method consists of calculating the point with the minimum slack. This slack is the maximum amount that a task deadline can be reduced. However, the same amount for all task deadlines will lead to different ratios. Hence, as a first approach Δ^* is chosen as the minimum ratio when new task deadlines are $D'_i = D_i - slack$. These steps are repeated until there is no slack available. Table 2 shows the algorithm.

Table 2.

1.	**Critical scaling factor algorithm:**
2.	input: τ;
3.	$\mathcal{R} \leftarrow Calculate\_ICI(\tau)$;
4.	$\Delta^* \leftarrow 1$
5.	$slackmin = min_0^{\mathcal{R}}(t - H_\tau(t))$;
6.	while $slackmin! = 0$
7.	$\Delta^* = min(1 - \frac{slackmin}{D_i})$;
8.	$D_i = \Delta^* D_i$;
9.	$slackmin = min_0^{\mathcal{R}}(t - H_\tau(t))$;
10.	end-while;
11.	return Δ^*;

It is important to note that now the minimisation is not optimal, in the sense that some tasks may reduce its deadline even more, but they are limited by other tasks. This way, the worst case response time of the tasks does not have to be equal to $\Delta^* D_i$. Hence, the output jitter analysis presented in the previous section is not applicable here.

5 Example

Now, we are going to illustrate the deadline minimisation method by means of an example. Consider a system of three tasks: $\tau = T_1 = (1, 7, 7)$, $T_2 = (3, 10, 10)$, $T_3 = (5, 20, 20)$. The minimisation sequence will be T_2, T_1, T_3. The ICI of this system is $[0, 10)$. Figure 2 depicts $H_\tau(t)$ function.

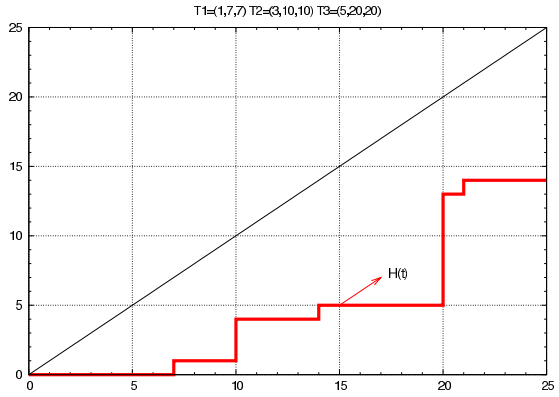

Figure 2. H_τ function

In order to apply the deadline minimisation algorithm to T_2, we have to calculate the number of jobs of T_2 within $[0, 10)$, that is, the k_2 parameter. For T_2, $k_2 = 1$. Then, from $t = D_2 = 10$, the algorithm searches backwards the next scheduling point. This point is $t = 7$. As can be seen in figure 2, the slack at time $t = 7$ is 6. Since $C_2 = 3 \leq 6$, we should continue searching backwards the next scheduling point, but the search ends because no more scheduling points before time 7 exists. Then, we can assign $D_2^{min} = C_2 = 3$.

Now, the new task system becomes $\tau' = \{T_1 = (1, 7, 7), T'_2 = (3, 3, 10), T_3 = (5, 20, 20)\}$, and the next task to minimise deadline is T_1. Figure 3 depicts $H_\tau(t)'$ function.

For T_1, $k_1 = 2$. Let's apply the deadline minimisation algorithm to each job:

- J_{11}: Algorithm starts at $t = 7$. The only scheduling point in $[0, 7)$ is in $t = 3$. Slack time in $t = 3$ is 0. Then $d'_{11} = H\tau'(3) + C_1 = 3 - 1 = 4$.

- J_{21}: Algorithm starts at $t = 14$. We have to search scheduling points from $t = 14$ to $t = P_1 + C_1 = 8$. As the only scheduling point in $[8, 14)$ is in $t = 13$ and in this time there is sufficient slack time, then $d'_{21} = 1$.

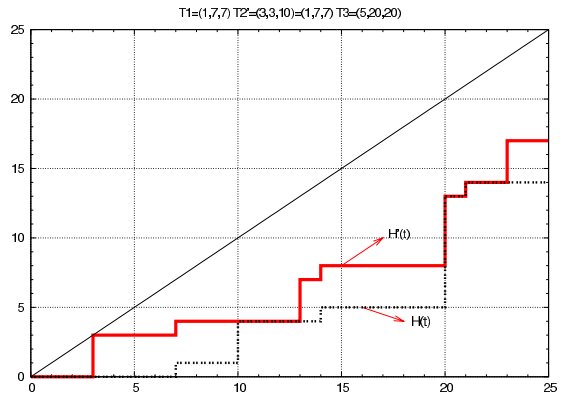

Figure 3. H_τ **and** H'_τ **functions**

Finally, $D_1^{min} = max(d'_{11}, d'_{21}) = 4$.

With the minimisation of T_1 deadline, the new task set is $\tau'' = T'_1 = (1,4,7)$, $T'_2 = (3,3,10)$, $T_3 = (5,20,20)$. Figure 4 depicts the new $H''_\tau(t)$ function.

For T_3, $k_3 = 1$, subsequently we have to find the scheduling points in $[5,20)$. Table 3 shows the scheduling points in this interval and the slack time associated. According to theorem 1, $D_3^{min} = d'_{3,1} = H''_\tau(4) + C_3 = 9$.

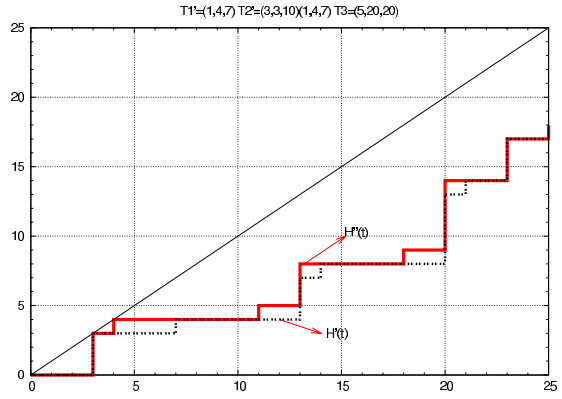

Figure 4. H'_τ **and** H''_τ **functions**

Table 3. Slack time in $[5,20)$

t	Slack = $t - H''_\tau(t)$
18	9
13	5
11	6
4	0

Therefore, the minimised deadline task set for order (T_2, T_1, T_3) is $\tau''' = T'_1 = (1,4,7)$, $T'_2 = (3,3,10)$, $T'_3 = (5,9,20)$. Table 4 shows the initial and final deadlines and the deadline reduction factors for the three tasks. This table also shows the calculation of Δ^*, and the deadlines associated with this parameter (rounded to the nearest upper integer).

Table 4. Results of deadline minimisation for order T_2, T_1, T_3

	D_i	D_i^{min}	α_i	α_i^{max}	Δ^*	$\Delta^* D_i$
T1	7	4	0.43	0.85	0.5	4
T2	10	3	0.70	0.70	0.5	5
T3	20	9	0.55	0.75	0.5	10

6 Experimental evaluation

In this section a series of simulations and comparisons with other works are presented to evaluate our deadline minimisation proposal.

A massive number of tests have been run, specifically, 1000 task sets have been generated for each utilisation. Each task was generated by randomly choosing the task period as an integer between 5 and 100, and then randomly selecting the task computation in such a way that the total system utilisation is approximately equal to the desired load.

The experiments are focused upon evaluating the level of the deadline reduction (α_i) in several conditions: *i)* when only the deadline of one task is minimised; *ii)* when all task deadlines are minimised with the same percentage (Δ^*); *iii)* as the deadline minimisation algorithm can be used to minimise jitter a third group of experiments is focused on the comparison with other works which propose a deadline scheme for reducing output jitter and control jitter.

6.1 Reducing the deadline of a task

Regarding the minimisation of one task deadline, experiments have been run with 5 and 10 tasks in each set.

As can be seen in Figure 5, with 10 tasks minimisation rises to 90% and the maximum deadline reduction is accomplished for all utilizations (reduced deadlines are equal or very close to task WCET). With 5 tasks, a smaller reduction is achieved, and it moves away from the maximum as the system utilisation increases. The main reason of this lower reduction is the experiment design. As the experiment generator tries to obtain a set of tasks for a specific utilisation, in general, the computation time of the tasks is greater than if the number of tasks is lower for the same periods. As a consequence, in a set of 5 tasks, a task is preempted for a shorter time period.

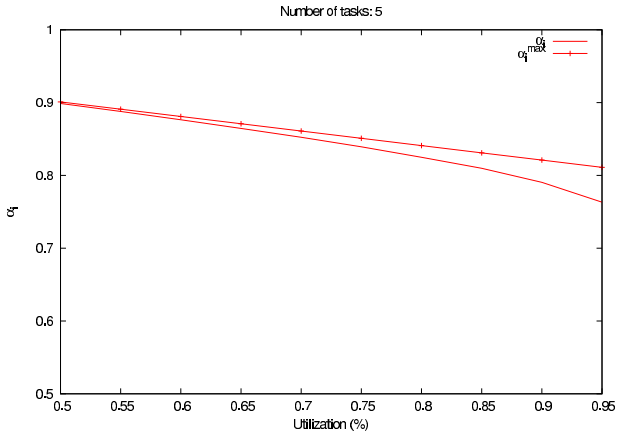

Figure 5. One task deadline minimisation

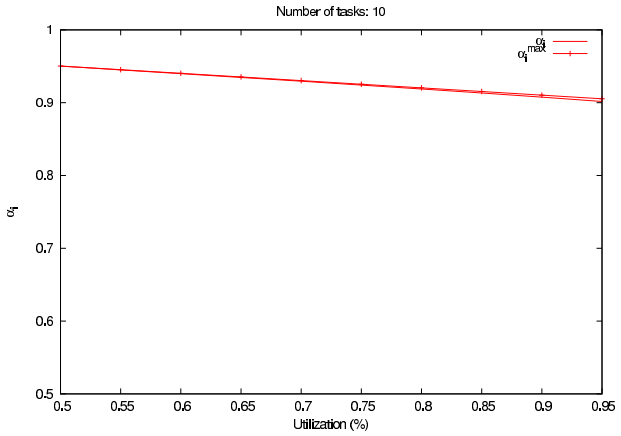

Figure 6. One task deadline minimisation

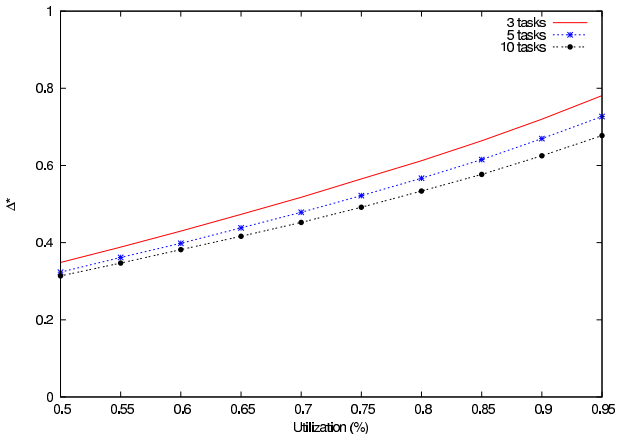

Figure 7. Critical scaling factor

6.2 Reducing all deadlines in a set of tasks

Figure 7 depicts the simulations obtained when Δ^* is calculated for tasks sets with 3, 5 and 10 tasks. As system load and number of tasks increase, the deadline reduction decreases (Δ^* approaches to 1).

6.3 Comparing the proposal with other works

This section describes the comparison with relevant works that deal with deadline reduction in order to: *i)* reduce output jitter; *ii)* reduce control jitter.

6.3.1 Output jitter reduction

Next simulations present the comparison with the work developed by Baruah et al. ([4]). This work proposed two methods for reducing the output jitter of a set of tasks: *Method 1* minimises jitter in polynomial time, whereas *Method 2* reduces it in pseudo-polynomial time. Figure 8 represents both methods for the reduction of the relative jitter over the complete task set and the relative jitter obtained with the deadline minimisation method presented in section 3. The tasks have been ordered by their period, so the first task chosen to apply the deadline minimisation algorithm is the task with the shortest period. As in the work presented in [4], simulations have been run with 5 and 10 task sets. Our proposal achieves better results because the deadline reduction is optimal, whereas the deadlines calculated in [4] can be reduce even more without jeopardising schedulability.

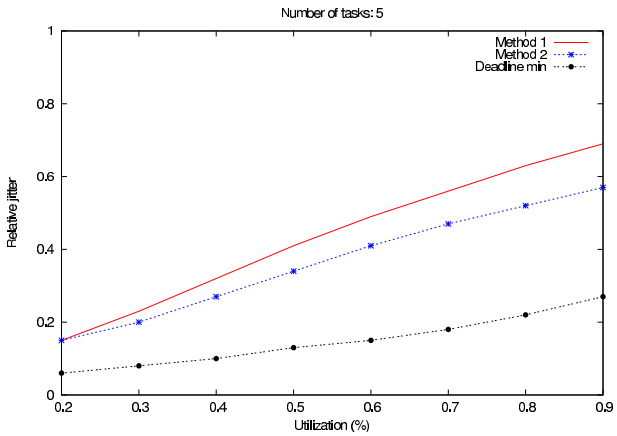

Figure 8. Output jitter comparison

6.3.2 Control jitter reduction

Finally, the deadline minimisation method is used to reduce the jitter of control tasks in real-time control applications.

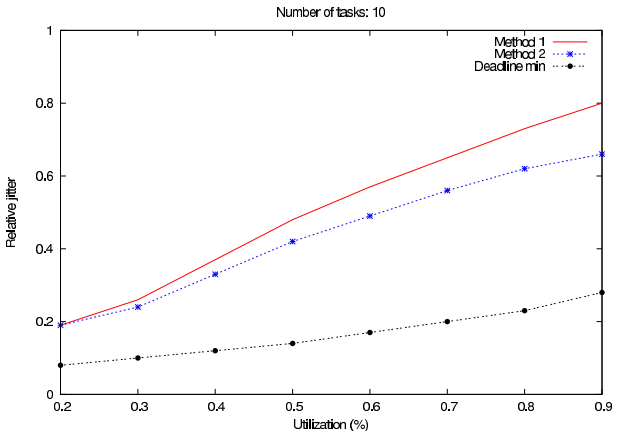

Figure 9. Output jitter comparison

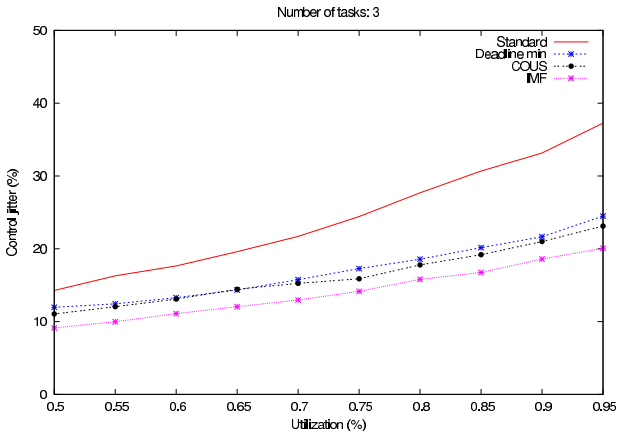

Figure 10. Control jitter comparison

Control jitter is defined as the variation in the response time of the task that sends the control action to the target system. Note the difference with *output jitter*, which is defined as the variation between successive periods, rather than over the lifetime of the system.

We have compared our method with the task models presented in [7] and [10]. Figure 10 shows the average control jitter (relative jitter of the task that sends the control action to the target system) of the following task models:

- Standard model: This is the original task set consisting of 3 control periodic tasks.
- COUS model: This model was proposed in [7]. Each control task is split into two subtasks: *Calculate Output* subtask (CO) and *Update State* subtask (US). New deadlines are calculated for each subtask. Control jitter is associated to CO subtask. This model produces a task set with 6 tasks.
- IMF model: This is the model presented in [10] and it splits the system in three subtasks (Initial, Mandatory and Final). As in the COUS model, new deadlines are obtained for initial and final subtasks, and a fixed offset is assigned to final tasks. Here, control jitter is associated to Final task. The original task set of 3 tasks, becomes 9 tasks.
- Deadline min: This is the standard model to which the deadline minimisation method has been applied where tasks have been sorted by period.

As figure 10 illustrates, the worst jitter is, as it is expected, for the standard model. The best results are obtained with the IMF model, whereas the COUS and the deadline minimisation model are very similar in terms of control jitter. This is also expected, because the more partitioning is made, the lesser computation time for final tasks, and hence, the lower jitter. However, some important remarks:

- IMF and COUS models make a task partitioning which implies more context switches than in a non partitioning task model. As Deadline minimisation model and COUS have similar jitter, it is better to choose our proposal that has less tasks.
- IMF applies a fixed delay to the final task. This means that the control action is delayed to achieve a very small jitter. This may cause a bad performance depending on the controlled system. For this reason, it is sometimes better to have a slightly bigger jitter with no fixed delay. In these situations, we would choose the deadline minimisation model.

7 Conclusions and future work

The implementation of multi-loop control systems implies the definition of a set of tasks running under timing constraints. The task scheduling is a fundamental issue in real-time control algorithm implementation to obtain the desired performance of the control system. Task parameters, such as computation times, periods and deadlines, are susceptible to adjustment during the design phase of the control system. Modifications in the periods may require the redesign of the controller in order to adjust it to the new rate. As a consequence, sensitivity analysis has mainly focused upon changes in task computation times. Although there are some works that deal with deadline modification, we have not found any sensitivity analysis of task deadlines, in other words, to find the minimum deadline of a task without jeopardising system feasibility, either for fixed or dynamic priority scheduling.

This paper proposes a sensitivity analysis for periodic task deadlines, using EDF scheduling. The deadline minimisation algorithm reduces a task deadline to the limit, while the critical scaling factor calculation establishes the same reduction ratio for all task deadlines within the system. The first method should be applied when it is important to reduce one task deadline in particular, this being more important than the others. While the second method is more useful when all tasks in the system have the same significance.

The sensitivity analysis presented in this paper has several advantages. As the simulations have shown, output jitter is strongly reduced. Moreover, in the case of deadline minimisation, the deadlines obtained are equal to the worst case response times of a task. Note that the calculation of WCRT in EDF is quite complex. Using our algorithm, there is no need to calculate the WCRT of a task.

As future lines of research, we can combine the deadline minimisation method with the control effort concept. The control effort of a task expresses the sensitivity of the task to time delays, i.e, to output jitter. In order to improve system performance, tasks with higher control effort are prone to greater reductions in their deadlines than the rest of the tasks.

References

[1] P. Albertos, A. Crespo, I. Ripoll, M. Vallés, and P. Balbastre. Rt control scheduling to reduce control performance degrading. In *Proceedings of the 39th IEEE Conference on Decision and Control*, 2000.

[2] P. Balbastre, I. Ripoll, and A. Crespo. Schedulability analysis of window-constrained execution time tasks for real-time control. In *14th Euromicro Conference on Real-Time Systems.*, 2002.

[3] P. Balbastre, I. Ripoll, J. Vidal, and A. Crespo. A task model to reduce control delays. *Journal of Real-Time Systems*, 27(3):215–236, 2004.

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

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

[6] G. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *IEEE Real-Time Systems Symposium*, pages 286–295, December 1998.

[7] A. Cervin. Improved scheduling of control tasks. In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, 1999.

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

[9] A. Crespo, P. Albertos, K. Arzen, A. Cervin, M. Torgren, and Z. Hanzalek. Artist2 roadmap on real-time techniques in control system implementation. Technical report, Control for Embedded Systems Cluster. EU/IST IST-004527, 2005.

[10] A. Crespo, I. Ripoll, and P. Albertos. Reducing delays in rt control: the control action interval. *IFAC World Congress Beijing*, 1999.

[11] M. DiNatale and J. A. Stankovic. Scheduling distributed real-time tasks with minimum jitter. *IEEE Trans. Computers*, 49(4):303–316, 2000.

[12] M. Garey and D. Johnson. *Computer and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[13] T. Kim, H. Shin, and N. Chang. Deadline assignment to reduce output jitter of real-time tasks. In *16th IFAC Workshop on Distributed Computer Control Systems*, 2000.

[14] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers, 1993.

[15] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behaviour. In *IEEE Real-Time Systems Symposium*, pages 166–171, 1989.

[16] J. Leung and J. Whitehead. On the complexity of fixed-priority schedulings of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[17] K. Lin and A. Herkert. Jitter control in time-triggered systems. In *Proceedings of the 29th Hawaii International Conference on System Sciences*, 1996.

[18] C. Liu and J.W.Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *JACM*, 23:46–68, 1973.

[19] C. Locke. Software architecture for hard real-time applications: cyclic executives vs. priority executives. *Journal of Real-Time Systems*, 4(1):37–53, 1992.

[20] S. Punnekkat, R. Davis, and A. Burns. Sensitivity analysis of real-time task sets. In *Proceedings of the Conference of Advances in Computing Science*, pages 72–82, 1997.

[21] I. Ripoll, A. Crespo, and A. Mok. Improvement in feasibility testing for real-time tasks. *Journal of Real-Time Systems*, 11:19–40, 1996.

[22] M. Spuri. Analysis of deadline scheduled real-time systems. *Rapport de Recherche RR-2772, INRIA, France*, 1996.

[23] D. C. Verma, H. Zhang, and D. Ferrari. Delay jitter control for real-time communication in a packet switching network. In *Proceedings of the IEEE Conference on Communications Software*, pages 35–43, 1991.

[24] S. Vestal. Fixed-priority sensitivity analysis for linear compute time models. *IEEE Transactions on Software Engineering*, 20(4):308–317, April 1994.