Improving Feasibility of Fixed Priority Tasks using Non-Preemptive Regions

Marko Bertogna, **Giorgio Buttazzo**
Scuola Superiore Sant'Anna, Pisa, Italy
{marko, giorgio}@sssup.it

Gang Yao
University of Illinois at Urbana-Champaign, USA
gangyao@illinois.edu

Abstract

Preemptive schedulers have been widely adopted in single processor real-time systems to avoid the blocking associated with the non-preemptive execution of lower priority tasks and achieve a high processor utilization. However, under fixed priority assignments, there are cases in which limiting preemptions can improve schedulability with respect to a fully preemptive solution. This is true even neglecting preemption overhead, as it will be shown in the paper.

In previous works, limited-preemption schedulers have been mainly considered to reduce the preemption overhead, and make the estimation of worst-case execution times more predictable. In this work, we instead show how to improve the feasibility of fixed-priority task systems by executing the last portion of each task in a non-preemptive fashion. A proper dimensioning of such a region of code allows increasing the number of task sets that are schedulable with a fixed priority algorithm. Simulation experiments are also presented to validate the effectiveness of the proposed approach.

1 Introduction

A common misconception in the scheduling of Fixed Priority (FP) sporadic tasks with implicit deadlines on a single processor system is that the best scheduler one can adopt is fully preemptive Rate Monotonic (RM). Indeed, RM is an optimal priority assignment for periodic task systems scheduled with FP in a fully-preemptive way. However, there can be cases in which limiting preemptions could improve the schedulability. In other words, there exist sporadic task systems that are not schedulable with preemptive RM, but can be scheduled using a limited preemptive policy. While this is a rather straightforward outcome when preemption overhead is considered in each task's WCET, somewhat more surprising is the observation that limited preemptive methods can be superior even when the preemption overhead is neglected.

A first example showing the dominance of limited preemptive methods over fully preemptive and non-preemptive scheduling in fixed priority systems was shown by Wang and Saksena [17], using Preemption Thresholds. A particularly interesting problem is how to exploit limited preemptive scheduling to maximize the system schedulability. In fact, for many task sets with total utilization less than or equal to one that are unfeasible under preemptive RM, a simple modification in the scheduler can lead to a performance comparable to EDF. Burns [7] formulated a conjecture stating that "For any task set with total utilization less than or equal to 100% there exists a dual priority assignment that will meet all deadlines." But such a result has never been formally proved and still remains an open problem.

In this paper, we consider the non-preemptive execution of selected regions of code of FP task systems, in order to increase the feasibility over fully preemptive and non-preemptive methods. In particular, if the last part of a task is executed non-preemptively, the interference from higher priority tasks can decrease significantly, so reducing its response time. If the non-preemptive region is well dimensioned, this might be sufficient to achieve the task set schedulability. However, note that the interference is reduced only when executing non-preemptively the *final* portion of the task, since the higher priority requests are thus postponed after the finishing time of the task. Instead, the non-preemptive execution of other portions of code, different from the final one, does not influence the task response time, since the interference of the higher priority tasks is not avoided, but simply postponed to a later instant, before its completion.

Contribution of the paper. In this paper, we address the problem of improving the schedulability of sporadic task systems scheduled with a Fixed Priority (FP) algorithm, identifying a limited-preemption strategy that allows extending the feasibility limits of preemptive schedulers. To do that, we exploit recent advancements in the theory of limited-preemption scheduling. In particular, we determine the longest duration of the final non-preemptive chunk of each task that guarantees a feasible schedule, if there exists one. The length of such a final part can also be used as a bound for any non-preemptive region inside the same task.

Note that, in a previous work [19], the same problem of identifying the longest non-preemptive region of each task was addressed under the assumptions that the task set was feasible under fully preemptive scheduling and tasks had constrained deadlines. Although these assumptions allow simplifying the complexity of the computation, restricting

1052-8725/11 $26.00 © 2011 IEEE
DOI 10.1109/RTSS.2011.30

the analysis to the first job of each task after a critical instant, they prevent taking advantage of the potential of limited preemption scheduling to find a schedulable solution when a task set is not feasible preemptively. In this paper, we relax the assumptions of preemptive feasibility and constrained deadlines, and show how to dimension the last non-preemptive region of each task to maximize schedulability.

Structure of the paper. The rest of the paper is organized as follows. Section 2 presents the system model and the terminology adopted in the paper. Section 3 discusses the related work, while Section 4 briefly recalls the existing results for deferred preemption scheduling. Section 5 derives an alternative schedulability test for deferred preemption systems. Section 6 illustrates the analysis for maximizing the length of the last non preemptive region of each task, thus improving the system schedulability. Section 7 illustrates some simulation experiments aimed at evaluating the average performance of the proposed solution against other approaches. Finally, Section 8 states our conclusions and future work.

2 System model and background

We consider a set τ composed of n sporadic tasks [1] $\tau_1, \tau_2, \ldots, \tau_n$ executing upon a single processor platform with preemption support. Each sporadic task τ_i ($1 \leq i \leq n$) is characterized by a worst-case execution time (WCET) C_i, a relative deadline D_i and a minimum inter-arrival time T_i, also referred to as period. Relative deadlines can be smaller than, equal to, or greater than periods. All parameters are assumed in $\mathbb{R}^+$. Each task generates an infinite sequence of jobs, with the first job arriving at any time and subsequent arrivals separated by at least T_i units of time.

Without loss of generality, we assume that tasks are indexed in decreasing priority order (i.e., if $0 < i < j \leq n$, then τ_i has higher priority than τ_j).

When a task τ_i is executed with deferred preemptions, q_i^{max} and q_i^{last} denote the length of the largest and of the last non-preemptive region of τ_i, respectively. Unless otherwise stated, we assume q_i^{max} and q_i^{last} to be in $\mathbb{R}^+$.

The objective of this work is to determine, for each task τ_i, the largest value of q_i^{last} that guarantees the schedulability of the task set. Note that a large q_i^{last} may decrease the response time of τ_i, reducing the interference it may suffer from higher priority tasks. However, q_i^{last} cannot be arbitrarily large, to limit the blocking time imposed to higher priority tasks.

We assume tasks to be independent (i.e., interacting with non blocking primitives) and assume a negligible preemption overhead. Note that, although such an assumption can be considered unrealistic, a secondary target of this work is to reduce the number of preemptions as much as possible, thus decreasing the cache related preemption delays and making task WCETs smaller and more predictable. Hence, providing the largest possible non-preemptive region of each task is a key factor for reducing the number of preemptions and their related cost. Moreover, the results derived in this paper can be easily extended to a model that takes preemption costs into account, seamlessly integrating with previously proposed techniques that allow determining an optimal set of fixed preemption points to minimize the overall preemption cost [3, 4].

For any sporadic task τ_i and any non-negative number t, the **request bound function** $\text{RBF}_i(t)$ denotes the maximum sum of the execution requests that could be generated by jobs of τ_i arriving within a contiguous time-interval $[a, b)$ of length t. It has been shown [12] that the request bound function for a sporadic task τ_i is:

$$\text{RBF}_i(t) \stackrel{\text{def}}{=} \left\lceil \frac{t}{T_i} \right\rceil C_i. \tag{1}$$

We also find it useful to define a *modified* request bound function $\text{RBF}_i^*(t)$ that considers all the execution requests in a time-interval $[a, b]$ of length t, including the right extreme b. Then:

$$\text{RBF}_i^*(t) \stackrel{\text{def}}{=} \left(\left\lfloor \frac{t}{T_i} \right\rfloor + 1 \right) C_i. \tag{2}$$

Note that $\text{RBF}_i(t)$ and $\text{RBF}_i^*(t)$ are identical, except for multiples of T_i, where $\text{RBF}_i^*(t)$ includes a further contribution C_i. The cumulative execution request of all tasks with priority greater than τ_i over any interval $[a, b)$ of length t is given by:

$$W_i(t) \stackrel{\text{def}}{=} \sum_{j=1}^{i-1} \text{RBF}_j(t). \tag{3}$$

Similarly, using a closed interval $[a, b]$ of length t, we define:

$$W_i^*(t) \stackrel{\text{def}}{=} \sum_{j=1}^{i-1} \text{RBF}_j^*(t). \tag{4}$$

It is possible to prove [6] that, for any instant of time t, there exists an arbitrarily small $\epsilon > 0$ such that

$$W_i^*(t - \epsilon) = W_i(t). \tag{5}$$

3 Related work

The schedulability analysis of preemptive Fixed Priority task systems has been established in the early ages of the real-time scheduling theory [14, 9, 12]. The main interest in preemptive schedulers was motivated by the better schedulability performance that preemption support guarantees when compared to non-preemptive strategies that are not able to achieve a high utilization due to the large blocking imposed to high priority jobs. Among preemptive FP schedulers, Deadline Monotonic (DM) has been proved in [13] to be an optimal priority assignment for sporadic task systems with constrained deadlines, i.e., with deadlines less than or equal to periods[1]. That means that if a task set

[1] DM is not optimal for sporadic task systems with arbitrary deadlines, as proved in [11].

according to the mentioned model can be positively scheduled with a static priority scheduler, then it is also schedulable with DM.

More recently, hybrid preemption strategies have been considered for two main reasons:

1. To reduce the preemption overhead imposed by fully preemptive schedulers, simplifying at the same time the WCET analysis of the task system;
2. To improve the schedulability of fully preemptive systems, reducing the interference due to preemptions from higher priority tasks.

The first target has been considered in [8, 6, 2, 18, 19, 3], where different limited-preemption methods have been explored to limit the context switch overhead, without imposing an excessive blocking to higher priority tasks. In [2, 18], a preemption model is considered that does not specify the exact location of the preemption points, which are assumed to be "floating" within the task code. Under this floating model, a method is proposed to compute, for each task scheduled with EDF [2] or FP [18], an upper bound on the maximum non-preemptive region that preserves the schedulability of the task system. When a task set is not preemptively feasible, the method fails, because no further improvement can be made when no information is available on the location of the non-preemptive regions.

The second target has been addressed in [16], proposing the *preemption threshold* scheduling. In this model, each task is assigned a nominal priority and a preemption threshold. A preemption will take place only if the preempting task has a nominal priority greater than the preemption threshold of the executing task. An exact schedulability analysis for FP with preemption thresholds has been presented in [10][2].

4 Deferred preemption scheduling

In [8], the *deferred preemption* model is proposed (also called *cooperative scheduling*), according to which each task is composed of a sequence of non-preemptive subjobs, separated by a preemption point. Since a preemption can take place only at subjob boundaries, the WCET analysis is simplified, allowing an easier computation of the context switch overhead. Moreover, the worst-case response time of a task can be smaller than in the preemptive case, since higher priority requests arriving during the execution of the last subjob of the considered task are postponed after its completion, potentially reducing the interference.

An exact schedulability analysis of the deferred preemption model has been presented in [6], noting that the largest response time of a task τ_i is found when (i) all higher priority tasks are released simultaneously with τ_i, and (ii) the longest subjob among the lower priority tasks starts executing an arbitrarily small amount of time earlier. This particular configuration of task releases is often called "critical instant" of task τ_i. However, the largest response time is not necessarily found in the first instance of τ_i after a critical instant, but can occur in later instances contained within the *level-i active period*, defined as follows.

Definition 1. A *level-i active period* is an interval $[a, b)$ such that the level-i pending workload is positive for all $t \in (a, b)$, and it is null in a and b.

Definition 2. The *level-i pending workload* $W(t)$ at time t is the amount of processing that still needs to be performed at time t due to jobs released before t by tasks with priority higher than or equal to τ_i's.

The maximum blocking B_i that a task τ_i can suffer due to lower priority tasks is equal to the length of the longest subjob among lower priority tasks[3]:

$$B_i = \max_{j>i} \left\{ q_j^{max} \right\}. \tag{6}$$

The length L_i of the largest level-i active period can be computed using the following recurrent relation, with initial value $L_i^{(0)} = B_i + C_i$:

$$L_i^{(\ell)} = B_i + \sum_{j=1}^{i} \mathrm{RBF}_j \left(L_i^{(\ell-1)} \right). \tag{7}$$

In particular, L_i is the smallest value for which $L_i^{(\ell)} = L_i^{(\ell-1)}$. The number of jobs of τ_i that are released in this interval is given by

$$K_i = \left\lceil \frac{L_i}{T_i} \right\rceil. \tag{8}$$

This means that the response time of τ_i must be computed for all jobs $\tau_{i,k}$ with $k \in [1, K_i]$.

For a generic job $\tau_{i,k}$, an upper bound on the start time $s_{i,k}$ of the last subjob can be computed considering:

- the maximum blocking time imposed to τ_i: B_i;
- the computation time of the preceding $(k-1)$ jobs: $(k-1)C_i$;
- the computation time of the subjobs of $\tau_{i,k}$, excluding the last one: $C_i - q_i^{last}$;
- the interference from higher priority tasks until the start of the last subjob of $\tau_{i,k}$, i.e., in $[0, s_{i,k}]$: $W_i^* (s_{i,k})$.

[2] The original analysis in [17] was flawed and has been corrected in [15], which in its turn has been improved by [10].

[3] To be precise, the blocking is an infinitesimal amount smaller than this value, because a lower priority subjob must start an arbitrarily small amount of time earlier than τ_i in order to block it.

Following [6], we distinguish the computation of $s_{i,k}$ into two cases, depending on whether the blocking B_i is null or not:

$$\begin{cases} s_{i,k}^{(\ell)} = B_i + kC_i - q_i^{last} + W_i\left(s_{i,k}^{(\ell-1)}\right), & \text{if } B_i > 0 \\ s_{i,k}^{(\ell)} = kC_i - q_i^{last} + W_i^*\left(s_{i,k}^{(\ell-1)}\right), & \text{if } B_i = 0. \end{cases} \tag{9}$$

Note that the blocking is null for the lowest priority task, as well as for each task that has all lower priority tasks executing in a fully preemptive way. When instead B_i is not null, the blocking time given by Equation (6) is an arbitrarily small amount larger than the real blocking imposed to τ_i. This infinitesimal difference is compensated in the corresponding term of Equation (9) by adopting W_i instead of W_i^* for the cumulative execution requests of higher priority tasks in $[0, s_{i,k}^{(\ell-1)}]$.

The start time $s_{i,k}$ of the last subjob of a generic job $\tau_{i,k}$ can then be computed as the fixed point of Equation (9), using $s_{i,k}^{(0)} = (k-1)T_i + C_i - q_i^{last}$ as initial value. Since, once started, the last subjob cannot be preempted, the finishing time $f_{i,k}$ can be computed as

$$f_{i,k} = s_{i,k} + q_i^{last}. \tag{10}$$

Hence, the response time of task τ_i is given by

$$R_i = \max_{k \in [1, K_i]} \{f_{i,k} - (k-1)T_i\}. \tag{11}$$

Once the response time of each task is computed, the task set is feasible if and only if

$$\forall i = 1, \ldots, n: \quad R_i \leq D_i. \tag{12}$$

5 Schedulability analysis

In order to compute the maximum length of the final subjob of each task that guarantees schedulability, we propose an alternative formulation of the schedulability test for deferred preemption systems.

As mentioned in Section 4, the schedulability of a task τ_i can be checked examining all jobs $\tau_{i,k}$ in the largest level-i active period of length L_i. Instead of computing the start time $s_{i,k}$ of the last subjob using the iterative method described in Section 4, the following lemma adopts a different technique.

Lemma 1. *A task τ_i is feasible if and only if $\forall k \in [1, K_i], \exists t \in ((k-1)T_i, (k-1)T_i + D_i - q_i^{last}]$, such that*

$$t \geq \begin{cases} B_i + kC_i - q_i^{last} + W_i(t), & \text{if } B_i > 0 \\ kC_i - q_i^{last} + W_i^*(t), & \text{if } B_i = 0. \end{cases} \tag{13}$$

Proof. The proof is identical for both $B_i > 0$ and $B_i = 0$. Consider the critical instant configuration. The schedulability of the first job of τ_i is guaranteed if the last subjob of $\tau_{i,1}$ can start its execution at least q_i^{last} time-units prior to its deadline at D_i. This happens if and only if Equation (13) is verified, with $k = 1$, for some $t \in (0, D_i - q_i^{last}]$. This is because at such a time the processor will have completed (i) the blocking B_i imposed by lower priority tasks, (ii) the execution request of τ_i up to the start of the last subjob of $\tau_{i,1}$, and (iii) the maximum cumulative execution requests of higher priority tasks. Moreover, since $t \leq D_i - q_i^{last}$, the last subjob will have sufficient time to complete before the deadline.

If the level-i active period is not over, the next job $\tau_{i,2}$ is considered. This can be done replacing τ_i with a task having computation time $2C_i$ and deadline $T_i + D_i$. This modified task will have sufficient time to start its last subjob before $T_i + D_i - q_i^{last}$ if and only if Equation (13) is verified, with $k = 2$, for some $t \in (0, T_i + D_i - q_i^{last}]$. The lemma follows applying the same procedure to each job $\tau_{i,k}$ until the end of the level-i active period, replacing τ_i with a job having a computation time of kC_i and a deadline $(k-1)T_i + D_i$. □

Note that it is possible to significantly simplify the test by observing that the only discontinuity points of $W_i(t)$ and $W_i^*(t)$ in Equation (13) coincide with release times of higher priority tasks. Let $\Pi_{i,k}$ be the set of release times of tasks $\tau_{j \leq i}$ that are contained in $((k-1)T_i, (k-1)T_i + D_i - q_i^{last}]$, including as well the point at the end of the interval:

$$\Pi_{i,k} \stackrel{\text{def}}{=} \left((k-1)T_i, (k-1)T_i + D_i - q_i^{last}\right] \cap \{hT_j, \forall h \in \mathbb{N}, j \leq i\} \cup \left\{(k-1)T_i + D_i - q_i^{last}\right\}.$$

Hence, we now reformulate the schedulability test on a reduced set of points.

Theorem 1. *A fixed-priority task set τ with arbitrary deadlines and deferred preemptions is feasible if and only if for every task $\tau_i \in \tau$, $\forall k \in [1, K_i], \exists t \in \Pi_{i,k}$, such that*

- *when $B_i > 0$: $B_i + kC_i - q_i^{last} + W_i(t) \leq t$*
- *when $B_i = 0$, one of the following conditions holds:*
  1. *$kC_i - q_i^{last} + W_i(t) < t$;*
  2. *For $\hat{t} = (k-1)T_i + D_i - q_i^{last}$: $kC_i - q_i^{last} + W_i^*(\hat{t}) \leq \hat{t}$.*

Proof. From Lemma 1, a necessary and sufficient schedulability condition for task τ_i is that $\forall k \in [1, K_i], \exists t \in ((k-1)T_i, (k-1)T_i + D_i - q_i^{last}]$ such that Equation (13) is satisfied. Consider the k-th job in the largest level-i active period of task τ_i. Note that $W_i(t)$ and $W_i^*(t)$ in Equation (13) are both non-decreasing functions of t, whose only discontinuity points are those in $\Pi_{i,k}$. We treat separately the cases with $B_i > 0$ and $B_i = 0$.

Case $B_i > 0$. Equation (13) becomes

$$B_i + kC_i - q_i^{last} + W_i(t) \leq t. \tag{14}$$

The "*if*" part of the theorem is trivially satisfied, noting that all points in $\Pi_{i,k}$ are contained in $((k-1)T_i, (k-1)T_i + D_i - q_i^{last}]$. Therefore, if there is a point in $\Pi_{i,k}$ that satisfies Condition (14), then the schedulability is guaranteed by Lemma 1.

To prove the "*only if*" part of the theorem, we will show that if there is a point $t' \notin \Pi_{i,k}$ that satisfies Condition (14) and that is contained in the considered interval $((k-1)T_i, (k-1)T_i + D_i - q_i^{last}]$, then the condition is also satisfied at a point $\in \Pi_{i,k}$.

Take the smallest point $t'' \in \Pi_{i,k}$ such that $t'' > t'$. Since $\Pi_{i,k}$ includes all discontinuity points of $W_i(t)$, including the end of the considered interval, and $W_i(t)$ is a non-decreasing function that is left-continuous, then $W_i(t'') = W_i(t')$. Therefore,

$$B_i + kC_i - q_i^{last} + W_i(t'') = \\ B_i + kC_i - q_i^{last} + W_i(t') \leq t' < t'',$$

proving the statement.

Case $B_i = 0$. Equation (13) becomes

$$kC_i - q_i^{last} + W_i^*(t) \leq t. \tag{15}$$

Since $W_i^*(t)$ is not left-continuous, we cannot immediately apply the technique used in the previous case. Instead, we prove that if there is a point $t' \in ((k-1)T_i, (k-1)T_i + D_i - q_i^{last}]$ that satisfies Condition (15), then one of the following conditions is satisfied as well (the "*only if*" part of the theorem):

1. there is a point $t'' \in \Pi_{i,k}$ that satisfies

$$kC_i - q_i^{last} + W_i(t'') < t''; \tag{16}$$

2. $t = (k-1)T_i + D_i - q_i^{last}$ satisfies Equation (15).

If $t' = (k-1)T_i + D_i - q_i^{last}$, the second condition is trivially satisfied. Otherwise, let t'' be the smallest point $\in \Pi_{i,k}$ such that $t'' > t'$. Since $\Pi_{i,k}$ includes all discontinuity points of $W_i^*(t)$, the following relation is verified for an arbitrarily small $\epsilon > 0$: $W_i^*(t'' - \epsilon) = W_i^*(t')$. Moreover, from Equation (5), $W_i^*(t'' - \epsilon) = W_i(t'')$. Therefore,

$$kC_i - q_i^{last} + W_i(t'') = \\ kC_i - q_i^{last} + W_i^*(t') \leq t' < t'',$$

and the first condition is satisfied, proving the statement.

It remains to prove the "*if*" part of the theorem for the case $B_i = 0$, i.e., that Condition 1. and 2. are also sufficient for schedulability. The sufficiency of Condition 2. trivially follows from Lemma 1, being $\hat{t} \in ((k-1)T_i, (k-1)T_i + D_i - q_i^{last}]$. To prove that Condition 1. is also sufficient, consider a point $t'' \in \Pi_{i,k}$ that satisfies Equation (16). We prove that there is also a point $t' \in ((k-1)T_i, (k-1)T_i + D_i - q_i^{last}]$ that satisfies Equation (15), so that the schedulability follows from Lemma 1. Let $t' = t'' - \epsilon$, for an arbitrarily small $\epsilon > 0$. Since t'' belongs to the left-open interval $((k-1)T_i, (k-1)T_i + D_i - q_i^{last}]$, then also $t' \in ((k-1)T_i, (k-1)T_i + D_i - q_i^{last}]$. Moreover, since Equation (16) has a strict inequality, t' can be chosen such that

$$kC_i - q_i^{last} + W_i(t'') < t' < t'';$$

From Equation (5), we have $W_i^*(t') = W_i^*(t'' - \epsilon) = W_i(t'')$. Then,

$$kC_i - q_i^{last} + W_i^*(t') = \\ kC_i - q_i^{last} + W_i(t'') < t',$$

proving the statement.

Repeating the same argument for all jobs $\tau_{i,k}$: $k \in [1, K_i]$ and for all tasks $\tau_i \in \tau$, the theorem follows. □

6 Improving the schedulability

Whenever the locations of the non-preemptive regions of each task are not given a priori, but can be freely decided at design time, it is possible to decrease the response time of a task by properly selecting the length of its last subjob. If a task is not feasible when executed preemptively, it might be the case that executing the last chunk of that task in a non-preemptive fashion might lead to a reduction in the worst-case response time, moving the interference from higher priority jobs after the completion of the task. In this way, the response time might decrease enough to avoid a deadline miss. However, care should be taken when selecting the length of the final subjob of a task, in order to avoid an excessive blocking to higher priority tasks.

In this section, we show how to compute, for each task τ_i, the length of the last subjob that maximizes the schedulability. To do that, we first show that the response time of a given task is minimized when the last subjob is as long as possible. Then, we compute an upper bound of the length of such a subjob, in order to avoid an excessive blocking to higher priority tasks. Finally, we derive an algorithm that computes the optimal length of the last subjob of each task to maximize the schedulability of the whole task set.

6.1 Minimizing the response time

The next theorem shows that the response time of a task is minimized when the last subjob is as long as possible.

Theorem 2. *Decreasing the length q_i^{last} of the last subjob of a task τ_i in a system scheduled with* FP *cannot decrease the response time of τ_i, when all other tasks' parameters remain the same.*

Proof. The proof is by contradiction. Suppose that a task τ_i has a smaller response time when decreasing q_i^{last} to $\hat{q_i}^{last} = q_i^{last} - \Delta q$, with $q_i^{last} \geq \Delta q > 0$. Let $\tau_{i,k}$ be

the job corresponding to the largest response time of τ_i after a critical instant, when the last subjob is of length q_i^{last}.

Assume $B_i > 0$. According to Equation (9), the start time $s_{i,k}$ of the last subjob of $\tau_{i,k}$ can be derived as the fixed point of the following relation

$$s_{i,k}^{(\ell)} = B_i + kC_i - q_i^{last} + W_i\left(s_{i,k}^{(\ell-1)}\right),$$

using $s_{i,k}^{(0)} = (k-1)T_i + C_i - q_i^{last}$ as initial value. A similar relation can be used to derive the start time $\hat{s}_{i,k}$ of the last subjob of $\tau_{i,k}$ when decreasing q_i^{last} to $\hat{q_i}^{last}$.

We first prove that

$$\hat{s}_{i,k} \geq s_{i,k} + \Delta q. \tag{17}$$

To do that, we induct over $s_{i,k}^{(\ell)}$ and $\hat{s}_{i,k}^{(\ell)}$.

Base case: $\hat{s}_{i,k}^{(0)} \geq s_{i,k}^{(0)} + \Delta q$.
Note that

$$s_{i,k}^{(0)} = (k-1)T_i + C_i - q_i^{last}$$

and

$$\hat{s}_{i,k}^{(0)} = (k-1)T_i + C_i - (q_i^{last} - \Delta q) = s_{i,k}^{(0)} + \Delta q,$$

proving the base statement.

Induction step. If $\hat{s}_{i,k}^{(\ell)} \geq s_{i,k}^{(\ell)} + \Delta q$, then $\hat{s}_{i,k}^{(\ell+1)} \geq s_{i,k}^{(\ell+1)} + \Delta q$.
Using Equation (9), we get

$$s_{i,k}^{(\ell+1)} = B_i + kC_i - q_i^{last} + W_i\left(s_{i,k}^{(\ell)}\right),$$

and

$$\hat{s}_{i,k}^{(\ell+1)} = B_i + kC_i - (q_i^{last} - \Delta q) + W_i\left(\hat{s}_{i,k}^{(\ell)}\right).$$

Note that $W_i(t)$ is a non-decreasing function of t that depends only on the execution requests of higher priority tasks. Since $\hat{s}_{i,k}^{(\ell)} \geq s_{i,k}^{(\ell)} + \Delta q$ by induction hypothesis, it follows that

$$W_i\left(\hat{s}_{i,k}^{(\ell)}\right) \geq W_i\left(s_{i,k}^{(\ell)}\right).$$

Therefore,

$$\hat{s}_{i,k}^{(\ell+1)} \geq B_i + kC_i - q_i^{last} + W_i\left(s_{i,k}^{(\ell)}\right) + \Delta q = s_{i,k}^{(\ell)} + \Delta q,$$

proving the statement, as well as Equation (17).

Using Equations (10) and (17), we get

$$\hat{f}_{i,k} = \hat{s}_{i,k} + q_i^{last} - \Delta q \geq s_{i,k} + q_i^{last} = f_{i,k}.$$

Therefore, the response time of $\tau_{i,k}$ when decreasing the last subjob length to $\hat{q}_i^{last}$ cannot decrease, reaching a contradiction.

When $B_i = 0$, the above considerations apply identically, using $W_i^*(t)$ instead of $W_i(t)$. □

According to the above theorem, the response time of a task is minimized when the last subjob is as long as possible, i.e., when maximizing the non-preemptive execution at the end of the task. Unfortunately, the length of the final non-preemptive region cannot be arbitrarily large, due to the limits imposed by the higher priority tasks, which cannot be blocked more than a given tolerance. In the next section, we show how this tolerance can be computed.

6.2 Computing the blocking tolerance

The *blocking tolerance* β_i of a task τ_i is defined as the maximum blocking that can be imposed to τ_i without missing any of its deadlines. We next show how to compute the blocking tolerance of a task that has the last subjob of length q_i^{last}.

We define $\beta_{i,k}$ as the blocking tolerance of the k-th job of τ_i after a critical instant. Using Theorem 1, the schedulability of job $\tau_{i,k}$ can be checked using the following condition, whenever $B_i > 0$:

$$\exists t \in \Pi_{i,k} : B_i \leq t - kC_i + q_i^{last} - W_i(t). \tag{18}$$

Rephrasing the terms, we obtain

$$B_i \leq \max_{t \in \Pi_{i,k}} \left\{ t - kC_i + q_i^{last} - W_i(t) \right\}.$$

The blocking tolerance of job $\tau_{i,k}$ is therefore

$$\beta_{i,k} = \max_{t \in \Pi_{i,k}} \left\{ t - kC_i + q_i^{last} - W_i(t) \right\}. \tag{19}$$

The blocking tolerance of task τ_i can be computed by Theorem 1, selecting the minimum blocking tolerance among the first K_i jobs of τ_i in the level-i active period after a critical instant:

$$\beta_i = \min_{k \in [1, K_i]} \beta_{i,k}. \tag{20}$$

A first problem is computing K_i using Equation (8). In fact, the length L_i of the largest level-i active period depends on the blocking imposed to τ_i, as shown in Equation (7). But B_i, that is set to the blocking tolerance of τ_i, is not yet known. To avoid this circular dependency, we can use an upper bound on the blocking tolerance β_i, given by the blocking tolerance of the first job $\tau_{i,1}$ after a critical instant. From Equation (20), we have $\beta_{i,1} \geq \beta_i$. Therefore, an upper bound on the largest level-i active period is given by the first fixed point of the following recurrent relation:

$$\hat{L}_i^{(\ell)} = \beta_{i,1} + \sum_{j=1}^{i} \mathrm{RBF}_j\left(\hat{L}_i^{(\ell-1)}\right), \tag{21}$$

with initial value $\hat{L}_i^{(0)} = \beta_{i,1} + C_i$. An upper bound on the number of jobs of τ_i that need to be checked to compute the blocking tolerance β_i is then given by

$$\hat{K}_i = \left\lceil \frac{\hat{L}_i}{T_i} \right\rceil. \tag{22}$$

The blocking tolerance of τ_i can then be computed as

$$\beta_i = \min_{k \in [1, \hat{K}_i]} \beta_{i,k}. \quad (23)$$

Extending the minimum to $\hat{K}_i \geq K_i$ jobs does not influence the correctness of the computed β_i, but only the number of steps required to compute this value. From Equation (19), it is easy to see that the blocking tolerances $\beta_{i,k}$ do not depend on B_i, which can be set to $\beta_{i,1}$ without affecting the analysis.

Note that in the above method we assumed $B_i > 0$. The procedure is therefore correct only when the returned blocking tolerance is strictly positive. Whenever instead one of the blocking tolerances $\beta_{i,k}$ computed with Equation (19) is null or negative, further rules are needed. In particular, if $\beta_{i,k}$ is negative, the algorithm can stop, declaring the task not schedulable. In fact, $\beta_{i,k} < 0$ implies

$$\forall t \in \Pi_{i,k} : t - kC_i + q_i^{last} - W_i(t) < 0,$$

Since $W_i^*(t) \geq W_i(t)$, it follows

$$\forall t \in \Pi_{i,k} : kC_i - q_i^{last} + W_i^*(t) > t.$$

and task τ_i has a deadline miss even when all lower priority tasks execute preemptively without imposing any blocking.

If instead there is a $\beta_{i,k} = 0$, job $\tau_{i,k}$ might be schedulable with $B_i = 0$, if Condition 1. or 2. of Theorem 1 is satisfied. Note that Condition 1. cannot be satisfied, because $\beta_{i,k} = 0$ implies

$$\forall t \in \Pi_{i,k} : kC_i - q_i^{last} + W_i(t) \geq t.$$

Instead, Condition 2. is satisfied if and only if

$$\hat{t} - kC_i + q_i^{last} - W_i^*(\hat{t}) = 0,$$

for $\hat{t} = (k-1)T_i + D_i - q_i^{last}$. In this case, the blocking tolerance of job $\tau_{i,k}$ is zero. If all the remaining jobs of τ_i in the considered level-i active period are also schedulable, task τ_i can be scheduled when all lower priority tasks are executed in a fully preemptive fashion, i.e., $q_j^{\max} = 0, \forall j < i$.

The procedure COMPUTEβ to compute the blocking tolerance of a task τ_i with $q_i^{last} > 0$ is presented in Figure 1.

When instead a task τ_i is scheduled fully preemptively ($q_i^{last} = q_i^{\max} = 0$), the schedulability can be checked using the classical condition derived in [11]:

$$\forall k \in [1, K_i], \exists t \in \Pi_{i,k} : B_i \leq t - kC_i - W_i(t). \quad (24)$$

In that case, the blocking tolerance is easily derived as

$$\beta_i = \min_{k \in [1, K_i]} \max_{t \in \Pi_{i,k}} \{t - kC_i - W_i(t)\}, \quad (25)$$

and the task is schedulable when $\beta_i \geq 0$.

JOBTOLERANCE(i, k)

1 $\beta_{i,k} \leftarrow \max_{t \in \Pi_{i,k}} \{t - kC_i + q_i^{last} - W_i(t)\}$
2 **if** $(\beta_{i,k} = 0)$ {
3 $\quad \hat{t} = (k-1)T_i + D_i - q_i^{last}$
4 $\quad \beta_{i,k} = \hat{t} - C_i + q_i^{last} - W_i^*(\hat{t})$ }
5 **return** $(\beta_{i,k})$

COMPUTE$\beta\left(i, q_i^{last} > 0\right)$

1 $\beta_{i,1} \leftarrow$ JOBTOLERANCE$(i, 1)$
2 **if** $(\beta_{i,1} < 0)$ **return** (negative value)
$\quad \triangleright$ Compute $\hat{L}_i$ using $\beta_{i,1}$ as an upper bound on B_i
3 $\hat{L}_i^{(0)} \leftarrow \beta_{i,1} + C_i$
4 **while** $\left(\hat{L}_i^{(\ell+1)} \neq \hat{L}_i^{(\ell)}\right)$
5 $\quad \left\{ \hat{L}_i^{(\ell+1)} \leftarrow \beta_{i,1} + \sum_{j \leq i} \text{RBF}_j\left(\hat{L}_i^{(\ell)}\right) \right\}$
6 $\hat{K}_i \leftarrow \left\lceil \frac{\hat{L}_i}{T_i} \right\rceil$
$\quad \triangleright$ Compute $\beta_{i,k}$ for all jobs in $\hat{L}_i$
7 **for** $\left(k = \left\{2 \ldots \hat{K}_i\right\}\right)$ {
8 $\quad \beta_{i,k} \leftarrow$ JOBTOLERANCE(i, k)
9 $\quad$ **if** $(\beta_{i,k} < 0)$ **return** (negative value) }
10 $\beta_i = \min_{k \in [1, \hat{K}_i]} \{\beta_{i,k}\}$
11 **return** (β_i)

Figure 1. Compute τ_i's blocking tolerance.

6.3 Maximizing the schedulability

Now that we know how to compute the blocking tolerance for each task, Theorem 2 suggests a way to decrease the response time of a given task τ_i as much as possible without compromising the schedulability of higher priority tasks. To avoid an excessive blocking to higher priority tasks, the length q_i^{last} of the last subjob of τ_i cannot exceed the minimum blocking tolerance among the higher priority tasks:

$$\beta_i^{min} \stackrel{\text{def}}{=} \min_{j<i} \{\beta_j\}, \quad (26)$$

where $\beta_0 = \infty$ for completeness. Since β_i^{min} could be larger than the WCET of τ_i, an optimal selection of the length of the last subjob is

$$q_i^{last} \leftarrow \min\left\{C_i, \beta_i^{min}\right\}, \quad (27)$$

as proved in the following theorem.

Theorem 3. *Setting q_i^{last} according to Equation* (27) *allows minimizing the response time of τ_i without affecting the schedulability of higher priority tasks.*

OPTSCHED(τ)

Initialize $\beta^{min} \leftarrow \infty$

▷ Check tasks with deferred preemptions

1 **for** $(i = 1; i \leq n; i{+}{+})$ $\{$

2 $\quad q_i^{last} \leftarrow \min\{C_i, \beta^{min}\}$

3 $\quad \beta_i \leftarrow$ COMPUTE$\beta\big(i, q_i^{last}\big)$

4 $\quad$ **if**$(\beta_i < 0)$ **return** ("`Infeasible`")

5 $\quad$ **if**$(\beta_i = 0)$ **break**

6 $\quad \beta^{min} \leftarrow \min\{\beta_i, \beta^{min}\}$ $\}$

▷ Check remaining tasks executed preemptively

7 **for** $(i = i + 1; i \leq n; i{+}{+})$ $\{$

8 $\quad q_i^{last} \leftarrow 0$

9 $\quad \beta_i \leftarrow \min\limits_{k \in [1, K_i]} \max\limits_{t \in \Pi_{i,k}} \{t - kC_i - W_i(t)\}$

10 $\quad$ **if**$(\beta_i < 0)$ **return** ("`Infeasible`")$\}$

11 **return** $\big($"`Feasible with` $\big\{q_i^{last}\big\}_{i=1}^{n}$"$\big)$

Figure 2. Procedure to maximize the task set schedulability by deferred preemptions.

Proof. When $C_i \leq \beta_i^{min}$, τ_i can be executed non-preemptively without affecting the schedulability of higher priority tasks. In this case, Theorem 2 guarantees that the response time of τ_i is minimized, since q_i^{last} is the maximum possible.

When instead $C_i > \beta_i^{min}$, q_i^{last} is set to β_i^{min}. Since the blocking tolerances are tight, a larger q_i^{last} would cause a deadline miss of at least one higher priority task, i.e., the one with the smallest blocking tolerance. Conversely, selecting a smaller q_i^{last} could not lead to a smaller response time of τ_i, according to Theorem 2. Therefore, the smallest response time that can be obtained for τ_i, without affecting the schedulability of higher priority tasks, is obtained setting q_i^{last} according to Equation (27). □

To maximize the chances of finding a feasible solution for the whole task set, an optimal strategy is to assign the q_i^{last} lengths according to Equation (27), proceeding in priority order, starting from the highest priority task. Figure 2 shows the pseudocode of procedure OPTSCHED(τ) for finding an optimal assignment of subjob lengths that maximizes the schedulability of a given task set τ. The *for* loop at line 1 evaluates all tasks in decreasing priority order, assigning q_i^{last} according to Equation (27). Whenever a task is found with a negative blocking tolerance, the procedure returns an infeasible result. When instead a null blocking tolerance is found, the task set might still be schedulable if all remaining lower priority tasks are executed in a fully preemptive way. The feasibility check for these tasks is executed in the *for* loop at line 7, using Equation (25).

7 Experimental results

This section presents some simulation experiments on randomly generated synthetic task sets aimed at evaluating how the schedulability of the system varies as a function of different task set parameters. The following scheduling algorithms have been considered in the comparison, and priorities were assigned according to the Deadline Monotonic algorithm.

- Fully preemptive scheduling (FPS), where preemption is allowed any time at arbitrary points.
- Non-preemptive scheduling (NPS), where preemption is completely disabled.
- Preemption threshold scheduling (PTS), using the optimal threshold assignment presented in [16][4].
- Limited preemptive scheduling (LPS), using the schedulability analysis presented in this paper. The task set is deemed schedulable if procedure OPTSCHED(τ) returns a feasible result.

Fully preemptive Earleast Deadline First (EDF) [14] has been also included in all the graphs to evaluate the difference with respect to an optimal solution. Each individual task set was generated as follows. The **UUniFast** algorithm [5] was used to generate a set of n tasks with total utilization equal to U_{tot}. Then, each computation time C_i was generated as a random integer uniformly distributed in a given interval $[100, 500]$, and the period was computed as $T_i = C_i/U_i$.

The performance of the algorithms was evaluated by comparing the ratio of feasible task sets, calculated as the number of feasible task sets divided by the total number of generated sets. In each experiment, 5000 task sets were randomly generated for each parameter configuration.

In the first experiment the number of tasks was set to $n = 10$ and the system utilization was varied from 0.6 to 1, with a step of 0.03. Figure 3 plots the feasible ratio when deadlines are equal to periods, whereas Figure 4 shows the results when relative deadlines are generated as a random integer in the range $[C_i + 0.5 \cdot (T_i - C_i),\ T_i]$.

It is interesting to observe that both LPS and PTS improve the schedulability level with respect to FPS, but our approach (LPS) is able to achieve a larger improvement, especially for large utilizations. For example, notice that in Figure 4 LPS is able to schedule 30% more task sets than FPS for U_{tot} around 0.9.

A second experiment has been carried out to test how schedulability is affected by the number of tasks. Here, the

[4] As shown by Wang and Saksena in [17], the RM and DM priority assignments may not be the optimal under PTS; however, to make the comparison fair, the same priority assignment was used for all the tested scheduling algorithms.

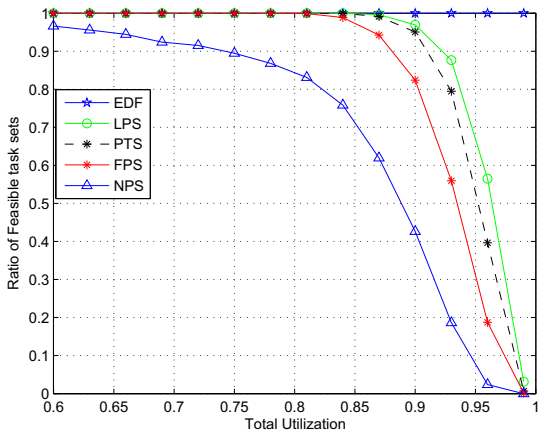

Figure 3. Feasible ratio versus utilization when $D = T$ and $n = 10$.

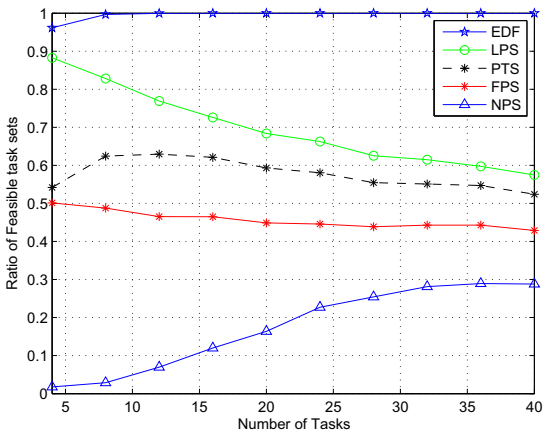

Figure 5. Feasible ratio as a function of n, when $D \leq T$ and $U_{tot} = 0.9$.

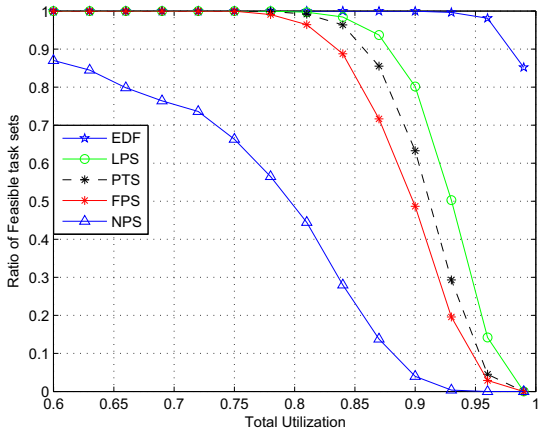

Figure 4. Feasible ratio versus utilization when $D \leq T$ and $n = 10$.

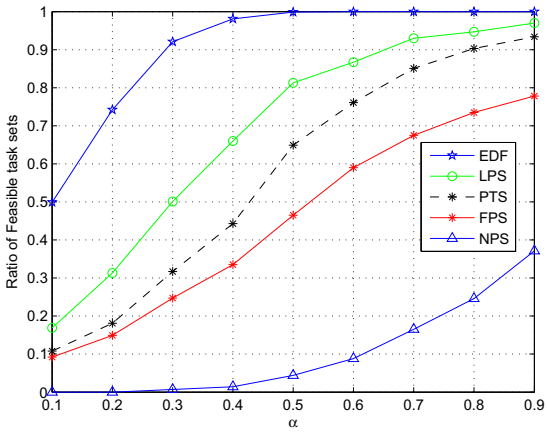

Figure 6. Feasible ratio versus deadline distribution when $D \leq T$, $n = 10$ and $U_{tot} = 0.9$.

total system utilization was set to $U_{tot} = 0.9$ and the number of tasks was varied from 4 to 40. The results are reported in Figure 5. Note that LPS always outperforms all the other fixed priority algorithms, although the improvement decreases for larger task sets. This can be explained observing that a large task set is more likely to have smaller blocking tolerances, due to the higher number of generated deadlines. When a task is generated with a small deadline, its blocking tolerance is also small. This limits the length of the non-preemptive regions of the lower priority tasks, so that LPS is not able to exploit its ability to improve the schedulability. On the other hand, the performance of NPS increases with n, because larger task sets tend to have smaller computation times, which introduce smaller blocking times in higher priority tasks.

A third experiment was performed to test the effect of relative deadline distribution on the task set feasibility. In this case, the task set included $n = 10$ tasks, with a fixed total utilization $U_{tot} = 0.9$, and each task deadline was generated as a random number in the range $[C_i + \alpha * (T_i - C_i), T_i]$. Results are reported in Figure 6. Note that increasing the range of task deadlines (i.e., reducing α), all algorithms, including EDF, degrade their performance, but LPS has still the best performance among all the fixed priority schemes.

In a final experiment, we changed the distribution range of task execution times and monitored how the system feasibility level was affected. The result is not reported here since no significant variation has been observed.

As a last remark, note that even if LPS has a better performance than PTS in every considered scenario, no dominance relation can be stated. In fact, there are task sets that are schedulable with PTS but not with LPS. However, very few such task sets have been found in our simulations (less than one out of a thousand generated sets).

8 Conclusions

This paper showed that limited preemptive scheduling is an effective method for improving the schedulability of fixed priority systems. In particular, the presented approach provides an algorithm for computing the longest non-preemptive region of each task, to be executed at the end of the code, to reduce its response time as much as possible, without jeopardizing the schedulability of the higher priority tasks.

Experimental results on synthetic task sets showed that limited preemptive scheduling is able to achieve an average schedulability level higher than preemption thresholds, for all task set utilizations.

As a future work, we intend to implement a policy that combines the benefits of deferred preemption scheduling and preemption thresholds. We believe this hybrid policy could potentially allow a larger least upper bound on the schedulable utilization of fixed priority systems, answering to the open problem on the achievable utilization of dual priority assignments [7].

References

[1] S. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium (RTSS'90)*, Orlando, Florida, 1990.

[2] M. Bertogna and S. Baruah. Limited preemption EDF scheduling of sporadic task systems. *IEEE Transactions on Industrial Informatics*, 6(4):579–591, 2010.

[3] M. Bertogna, G. Buttazzo, M. Marinoni, G. Yao, F. Esposito, and M. Caccamo. Preemption points placement for sporadic task sets. In *Proceedings of the 22nd Euromicro Conference on Real-Time Systems (ECRTS'10)*, Brussels, Belgium, June 2010.

[4] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, and G. Buttazzo. Optimal scheduling with variable preemption overhead. In *Proceedings of the 23rd Euromicro Conference on Real-Time Systems (ECRTS'11)*, Porto, Portugal.

[5] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Syst.*, 30(1-2):129–154, 2005.

[6] R.J. Bril, J.J. Lukkien, and W.F.J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. *Real-Time Systems: The International Journal of Time-Critical Computing*, 42(1-3):63–119, 2009.

[7] A. Burns. Dual priority scheduling: Is the processor utilisation bound 100%? In *Proceedings of 1st International Real-Time Scheduling Open Problems Seminar (RTSOPS)*, Brussels, Belgium.

[8] A. Burns. Preemptive priority-based scheduling: an appropriate engineering approach. In Sang H. Son, editor, *Advances in real-time systems*, pages 225–248. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[9] M. Joseph and P. K. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, October 1986.

[10] U. Keskin, R.J. Bril, and J.J. Lukkien. Exact response-time analysis for fixed-priority preemption-threshold scheduling. In *Work-in-Progress Session (WiP) of the 15th International Conference on Emerging Technologies and Factory Automation (ETFA)*, Bilbao, Spain, September 2010.

[11] J. P. Lehoczky. Fixed priority scheduling of periodic tasks with arbitrary deadlines. In *IEEE Real-Time Systems Symposium (RTSS'90)*, Orlando, Florida, 1990.

[12] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the Real-Time Systems Symposium (RTSS'89)*, Santa Monica, California, USA, December 1989.

[13] J. Y.-T Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

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

[15] J. Regehr. Scheduling tasks with mixed preemption relations for robustness to timing faults. In *Proceedings of the IEEE International Real-Time Systems Symposium (RTSS'02)*, Cancun (Mexico), December 2002.

[16] M. Saksena and Y. Wang. Scalable real-time system design using preemption thresholds. In *Proceedings of the IEEE Real-Time Systems Symposium*, Los Alamitos, CA, November 2000. IEEE Computer Society.

[17] Y. Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In *Proceedings of the International Conference on Real-time Computing Systems and Applications*. IEEE Computer Society, 1999.

[18] G. Yao, G. Buttazzo, and M. Bertogna. Bounding the maximum length of non-preemptive regions under fixed priority scheduling. In *RTCSA*, Beijing, China, May–June 2009.

[19] G. Yao, G. Buttazzo, and M. Bertogna. Feasibility analysis under fixed priority scheduling with fixed preemption points. In *International Workshop on Real-Time Computing Systems and Applications (RTCSA)*, Macau, China, August 2010.