Feasibility Analysis under Fixed Priority Scheduling with Fixed Preemption Points[*]

Gang Yao, **Giorgio Buttazzo** and **Marko Bertogna**
Scuola Superiore Sant'Anna
Pisa, Italy
{g.yao, g.buttazzo, m.bertogna}@sssup.it

Abstract

Limited preemption models have been proposed as a viable alternative between the two extreme cases of fully preemptive and non-preemptive scheduling. In particular, allowing preemption to occur only at predefined preemption points reduces context switch costs, simplifies the access to shared resources, and allows more predictable estimations of worst-case execution times. Current results related to such a model, however, exhibit two major deficiencies: (i) The exact response time analysis is too complex; (ii) The maximum lengths of the non-preemptive regions for each task are still unknown.

In this paper, we address the problem of schedulability analysis of real-time tasks having fixed priorities and fixed preemption points. In particular, a novel task model is proposed to improve the feasibility analysis, and an efficient feasibility test is derived under specific but general assumptions. Finally, an algorithm for computing the maximum length of fixed non-preemptive regions for each task is described, and some simulation experiments are presented to validate the proposed approach.

1 Introduction

Since the pioneering work of Liu and Layland [21], a lot of research has been done in the area of real-time scheduling to analyze and predict the schedulability of a task set under different scheduling policies and task models. Most of the available results have been derived under a fully preemptive model, where every task can be suspended in any point and at any time, in favor of a task with higher priority. When context switch overhead is ignored in the analysis, as done in most scheduling papers, the fully preemptive model is more efficient in terms of processor utilization, and allows better schedulability results.

In practice, however, arbitrary preemptions can introduce a significant runtime overhead and may cause high fluctuations in task execution times, so degrading system predictability. In particular, three different types of costs need to be taken into account at each preemption [13]. A scheduling cost is due to the time taken by the scheduling algorithm to suspend the running task, insert it into the ready queue, switch the context, and dispatch the new incoming task. A Pipeline cost is due to the time taken to flush the processor pipeline when the task is interrupted and the time taken to refill the pipeline when the task is resumed. A cache-related cost is due to the time taken to reload the cache lines evicted by the preempting task. This time depends on the specific point in which preemption occurs and on the number of preemptions experienced by the task [20, 13, 1].

Moreover, to avoid unbounded priority inversion when accessing shared resources, preemptive scheduling requires the implementation of specific concurrency control protocols, such as Priority Inheritance, Priority Ceiling [23] or Stack Resource Policy [2], which introduce additional overhead and complexity, whereas non-preemptive scheduling automatically prevents unbounded priority inversion.

On the other hand, fully non-preemptive scheduling is too inflexible for certain applications and could introduce large blocking times that would prevent guaranteeing the schedulability of the task set.

To overcome such difficulties, different scheduling approaches have been proposed in the literature to avoid arbitrary preemptions and limit the length of non-preemptive execution.

- *Fixed preemption points (FPP)*. According to this model, each task is divided into a number of non-preemptive chunks (also called subjobs) by inserting predefined preemption points in the task code. If a higher priority task arrives between two preemption points of the running task, preemption is deferred until

the next preemption point.

- *Floating non-preemptive regions.* Another approach consists in defining for each task τ_i a maximum non-preemptive interval q_i and switching to non-preemptive mode whenever needed, for no more than q_i units of time. With respect to the case of fixed preemption points, in this model non-preemptive regions are not fixed, but can be considered to be floating in the task code.

- *Preemption Thresholds.* A different approach for limiting preemptions is based on the concept of preemption thresholds, proposed by Wang and Saksena [24] under fixed priority systems. This method allows a task to disable preemption up to a specified priority, which is called preemption threshold. Each task is assigned a regular priority and a preemption threshold, and the preemption is allowed to take place only when the priority of arriving task is higher than the threshold of the running task.

From a practical point of view, using fixed preemption points allows achieving higher predictability. In fact, by properly selecting the preemption points in the code, it is possible to reduce cache misses and context switch costs, so improving the estimation of preemption overhead and worst-case execution times [13].

For the reasons explained above, in this paper we consider a limited preemption model with fixed preemption points (FPP). In this model, the length of the final non-preemptive chunk plays a crucial role in reducing the task response time. In fact, all higher priority jobs arriving during the execution of the final chunk of the running task do not cause a preemption, and their execution is postponed at the end of the task.

Motivating example   Let us consider a task set consisting of 3 periodic tasks, with relative deadlines equal to periods. The task set is described as $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\} = \{(1,4),(1,6),(4,12)\}$, where the first number represents the task computation time and the second the period.

Assuming a synchronous activation of the task set, the schedule produced by Rate Monotonic in fully preemptive mode is shown in Figure 1(a). As clear from the figure, τ_3 is preempted twice and has a response time equal to 8 units of time. However, if the last 3 units of τ_3 are executed non preemptively, the two preemptions do not take place and the response time reduces to 6 units, as shown in Figure 1(b).

This simple example clearly shows that the last portion of a task, when executed in non-preemptive mode, can significantly reduce the interference from higher priority tasks, thus reducing the task response time. However, a long non-preemptive region can cause large blocking to higher priority tasks, possibly jeopardizing the system feasibility.
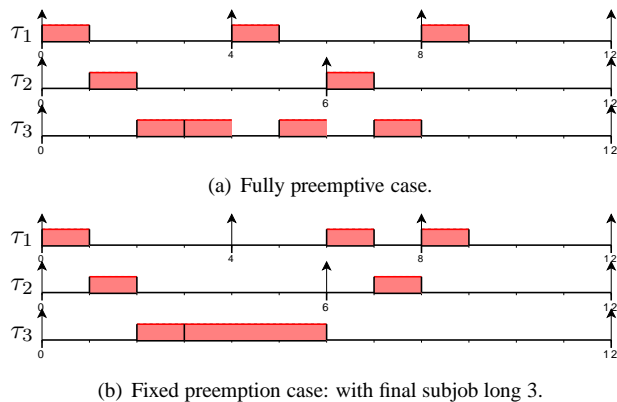

(a) Fully preemptive case.

(b) Fixed preemption case: with final subjob long 3.

Figure 1. Fully preemptive scheduling vs. fixed preemption points.

Contributions of the paper   This work provides four main contributions. First, we propose a novel task model that allows improving and simplifying the feasibility test of tasks with fixed preemption points. Second, we identify the conditions under which the feasibility check of a fixed-priority task set can be limited only to the first instance of each task (instead of checking multiple instances within a certain period, as proved by Bril et al. [8]). Third, based on this result, we present an efficient test to verify the schedulability of fixed priority tasks with fixed non-preemptive regions, and, finally, we present an algorithm for computing a bound on the length of non-preemptive chunks for each task, discussing how such a bound varies as a function of the length of the final subjob.

Paper Organization   The rest of the paper is organized as follows. Section 2 presents some related work. Section 3 introduces the new task model and the methodology used in the paper. Section 4 determines the conditions under which the response time analysis for the FPP model can be simplified. Section 5 presents the feasibility test for fixed priority tasks with given subjob division. Section 6 illustrates the algorithm for computing the maximum length of subjobs for each task without violating the system feasibility. Section 7 reports some simulation results. Finally, Section 8 states our conclusions and future work.

2   Related Work

Most work on non-preemptive scheduling has typically focused on single-job models, where tasks have precedence relations, are invoked only once, and must be completed before a deadline [12, 11]. A more general characterization of periodic tasks has been considered in [19, 16, 18]. In this model, tasks may have a deadline smaller than the next

release time. For this more general model, Mok [22] has shown that the problem of deciding schedulability of a set of periodic tasks with mutually exclusive sections of code is NP-hard.

Jeffay et al. [15] showed that non-preemptive scheduling of concrete periodic tasks is NP-Hard in the strong sense. George et al. [14] provided comprehensive feasibility analysis on non-preemptive scheduling, however, the authors assumed either a completely non-preemptive or a fully preemptive model. Davis et al. [10] considered typical applications of non-preemptive fixed priority scheduling on a CAN bus, and presented the analysis to bound worst-case response times of real-time messages.

Under the floating model, Baruah et al. [3] computed the longest non-preemptive interval for each task that does not jeopardize the schedulability of the task set under EDF, with respect to the fully preemptive case. Yao et al. [25] addressed the same problem, but under fixed priorities.

Fixed priority scheduling with deferred preemptions, which allows preemption only at some predefined points inside the task code, has been proposed by Burns [9], who applied the response time analysis to verify the schedulability of fixed priority tasks with arbitrary phasing and constrained deadlines, but did not address the problem of computing the maximum length of non-preemptive chunks. Bril et al. [8] identified a critical situation that may occur in the presence of non-preemptive regions, deriving the analysis to take such a phenomenon into account. In particular, in certain situations, the execution of the last non-preemptive chunk of a task τ_i can delay the execution of a some higher priority tasks, which can later interfere with the subsequent invocations of τ_i. Identifying such a situation, later referred to as *self-pushing* phenomenon, requires a more complex test, since the analysis cannot be limited to the first job of each task, but it must be performed on multiple task instances within a certain period.

The research presented in this paper is motivated by the need of limiting the number of preemptions to better estimate the preemption overhead, reduce the worst-case execution times, and improve schedulability. However, this work focuses on schedulability analysis and on the computation of the maximum length of non-preemptive regions, leaving the problem of estimating preemption overhead to a future research. Moreover, compared to similar previous results [3, 25], this work assumes fixed preemption points instead of arbitrary positions.

3 Task Model and Methodology

In this section, we present the task model and the terminology used throughout the paper. Then, we introduce some definitions that are used to derive the feasibility test.

3.1 Task model

We consider a set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n periodic or sporadic tasks that have to be executed on a uniprocessor under fixed priority scheduling. Each task τ_i is characterized by a worst-case execution time C_i, a relative deadline D_i, and a period (or minimum inter-arrival time) T_i between two consecutive releases. Each task consists of an infinite sequence of jobs $\tau_{i,k}$ ($k = 1, 2, \ldots$) with arrival time $r_{i,k}$ and absolute deadline $d_{i,k} = r_{i,k} + D_i$. Tasks can be scheduled by any fixed-priority assignment and are indexed by decreasing priority, meaning that τ_1 is the highest priority task. In particular, the following notation is used in the paper:

$$\begin{cases} hp(i) = \{\tau_j \mid j < i\} \\ hep(i) = \{\tau_j \mid j \leq i\} \\ lp(i) = \{\tau_j \mid j > i\} \end{cases}$$

We assume that every task τ_i consists of m_i non-preemptive chunks (subjobs), obtained by inserting $m_i - 1$ preemption points in the code. Thus, preemptions can only occur at the subjobs boundaries. The kth subjob has a worst-case execution time $q_{i,k}$, hence $C_i = \sum_{k=1}^{m_i} q_{i,k}$. In particular, the last subjob of job $\tau_{i,k}$ is denoted as $F_{i,k}$.

To improve the schedulability analysis, two additional parameters q_i^{max} and q_i^{last} are introduced in the task model:

$$\begin{cases} q_i^{max} = \max_{k=1}^{m_i}\{q_{i,k}\} \\ q_i^{last} = q_{i,m_i} \end{cases} \tag{1}$$

The reasons for choosing these two values can be summarized as follows:

1. Non-preemptive execution can possibly cause blocking to higher priority tasks and the feasibility of a task τ_k is affected by the size q_i^{max} of the longest subjob of each lower priority task $\tau_i \in lp(k)$.

2. For task τ_i, the length q_i^{last} of the final subjob directly affects its response time. In fact, all higher priority jobs arriving during the execution of τ_i's final subjob do not cause a preemption, since their execution is postponed at the end of τ_i (see the example in Figure 1).

Therefore, we consider each task to be characterized by the following 5-tuple:

$$\{C_i, D_i, T_i, q_i^{last}, q_i^{max}\}.$$

The advantage of such a model will be shown throughout the paper. In the following, the superscript P and FPP will be used to denote that a specific parameter or function refers to the preemptive and FPP model, respectively. The following subsections present some concepts that will be used in the analysis to derive the main results.

3.2 Critical instant

To simplify the analysis, the feasibility check to determine whether a given task τ_i is schedulable under a certain scheduling policy is done under the worst-case scenario that leads to the largest possible response time. The activation time of τ_i causing the worst-case scenario is defined as the critical instant [21].

When tasks have non-preemptive regions, Bril et al. [7] showed that the critical instant of τ_i occurs when it is released simultaneously with all higher priority tasks, and the longest non-preemptive subjob of lower priority tasks starts an infinitesimal time before the release of τ_i.

Bril et al. [8] also showed that, when tasks have non-preemptive regions at the end of their code, the worst-case response time may not occur in the first job. Hence, the feasibility of a task set cannot be checked by analyzing only the first job of each task, as done in fully preemptive systems, but it must be checked for multiple jobs within a certain time interval, which introduces significant computation complexity.

3.3 Request bound function

Schedulability analysis is performed using the *request bound function* RBF(τ_i, t), defined as the maximum cumulative execution request that can be generated by jobs of τ_i within an interval of length t from the critical instant. In [17], it has been shown that

$$\text{RBF}(\tau_i, t) = \left\lceil \frac{t}{T_i} \right\rceil C_i. \tag{2}$$

The cumulative execution request of a task τ_i and all higher priority tasks over an interval of length t is therefore bounded by:

$$W_i(t) = C_i + \sum_{\tau_j \in hp(i)} \text{RBF}(\tau_j, t). \tag{3}$$

A necessary and sufficient schedulability test for fixed priority preemptive tasks was derived by Lehoczky et al. [17], by checking whether for every task τ_i there exists a value $t \leq D_i$ such that $W_i(t) \leq t$. This is stated in the following lemma [17].

Lemma 1. *A fixed-priority task set is feasible under fully preemptive scheduling if and only if $\forall \tau_i \in \mathcal{T}, \exists t \leq D_i$, such that*

$$W_i(t) \leq t. \tag{4}$$

where $W_i(t)$ is defined in Equation (3).

If t^* is the smallest value that satisfies Equation (4), then $W_i(t^*) = t^*$ corresponds to the worst-case response time.

3.4 Worst-case occupied time

As shown by Bril [6], the response time of a job can be computed by considering the *occupied time* $WO_i(C)$, which is the interval of time between the job release and the time at which the job starts or resumes its execution after the completion of C units of computation time. Then, he showed that the worst-case response time WR_i of a task can be expressed in terms of worst-case occupied time WO_i by taking the following limit from the left-hand side:

$$WR_i(C) = \lim_{x \uparrow C} WO_i(x). \tag{5}$$

where $WO_i(x)$ is the smallest $t \in \mathbb{R}$ that satisfies

$$t = x + \sum_{\tau_j \in hp(i)} \left(\left\lfloor \frac{t}{T_j} \right\rfloor + 1 \right) C_j. \tag{6}$$

Notice that, in Equation (6), the only difference with respect to the worst-case response time is that the ceiling function is replaced by the floor plus one. This essential difference indicates that the response time is computed when the job finishes its execution, regardless of whether other higher priority tasks are released at the end, whereas the occupied time also accounts for the higher priority jobs arriving at the end of the current job's execution.

For example, in the schedule illustrated in Figure 1, the worst-case response time of τ_3 is 8 in Figure 1(a) and 6 in Figure 1(b), whereas its worst-case occupied time is 8 in both cases.

4 Simplifying conditions

In this section, we prove that, under the FPP model, the feasibility test can be restricted to the first job of each task, activated at its critical instant, if the following conditions hold:

A1. (Constrained deadlines) $D_i \leq T_i$.

A2. (Preemptive feasibility) The task set is feasible under a fully preemptive model.

Notice that these conditions are not restrictive and are verified for most real-time applications. To prove that under these assumptions the analysis can be simplified, we first introduce the concept of *Self-Pushing* phenomenon and derive a number of properties under such a condition, then we prove the main theorem.

4.1 Properties of the self-pushing scenario

Definition 1. *Under fixed-priority scheduling, a* self-pushing *phenomenon on a task τ_i is defined as the condition in which there exists a job $\tau_{i,k}$, with $k > 1$, such that*

its response time is larger than the first job under the critical instant, that is:

$$\exists k > 1, \quad R_{i,k}^{FPP} > R_{i,1}^{FPP}. \tag{7}$$

Now, assume that there exists a self-pushing phenomenon in task τ_i and let $\tau_{i,k}$, $k > 1$ be the first job such that $R_{i,k}^{FPP} > R_{i,1}^{FPP}$. Let $s_{i,k}$ and $s_{i,k-1}$ be the start times of final subjob $F_{i,k}$ and $F_{i,k-1}$, respectively. Such a scenario is illustrated in Figure 2, where the final subjobs are depicted in gray. The following properties can be derived on time interval $[s_{i,k-1}, s_{i,k}]$.
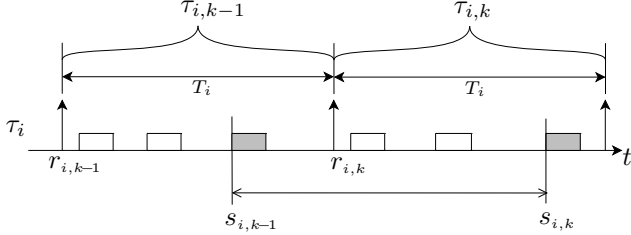

Figure 2. The self-pushing phenomenon.

Property 1. The start time $s_{i,k-1}$ cannot coincide with the arrival time of tasks from $hp(i)$.

Proof. If a higher priority job arrives when the final subjob $F_{i,k-1}$ is about to start, then preemption will take place before the execution of $F_{i,k-1}$; that is, $F_{i,k-1}$ will start executing after that higher priority job. Hence, the property holds. □

Property 2. The interval $[s_{i,k-1}, s_{i,k}]$ is larger than T_i, that is

$$s_{i,k} - s_{i,k-1} > T_i.$$

Proof. According to the definition of self-pushing, we have

$$R_{i,k}^{FPP} = s_{i,k} + q_i^{last} - r_{i,k} > R_{i,1}^{FPP} \tag{8}$$

Since $\tau_{i,k}$ is the first job experiencing self-pushing, for $\tau_{i,k-1}$ we have

$$R_{i,k-1}^{FPP} = s_{i,k-1} + q_i^{last} - r_{i,k-1} \le R_{i,1}^{FPP}. \tag{9}$$

Combining Equations (8) and (9), and noticing that $r_{i,k} \ge r_{i,k-1} + T_i$, we have

$$s_{i,k} - s_{i,k-1} > r_{i,k} - r_{i,k-1} \ge T_i.$$

which proves the property. □

Property 3. The processor is always executing jobs from $hep(i)$ in $[s_{i,k-1}, s_{i,k}]$.

Proof. This can be proved by contradiction. Let $t' \in [s_{i,k-1}, s_{i,k}]$ be the first time instant in which the processor is not executing tasks from $hep(i)$. Clearly, t' cannot be in $[s_{i,k-1}, s_{i,k-1} + q_i^{last}]$, since $F_{i,k-1}$ starts executing non-preemptively at $s_{i,k-1}$. Also, since in $[r_{i,k}, s_{i,k}]$ $\tau_{i,k}$ has remaining execution to be completed, t' cannot be in $[r_{i,k}, s_{i,k}]$. Hence, t' must be within $(s_{i,k-1} + q_i^{last}, r_{i,k})$. All tasks from $hp(i)$ arriving before t' must get finished before that time, by definition of t'. If other tasks are activated at or after t', the overall interference (including blocking) will certainly be no greater than the total delay experienced by the first job (which is activated at the critical instant). Hence, $R_{i,k}^{FPP} \le R_{i,1}^{FPP}$, which contradicts the self-pushing assumption and proves the property. □

4.2 Simplified feasibility analysis

The following lemma uses the previous properties to show that no self-pushing can occur when conditions A1 and A2 are verified.

Lemma 2. *If the task set has constrained deadlines (A1) and is preemptively feasible (A2), then no self-pushing phenomenon can occur under the fixed-priority FPP model.*

Proof. By contradiction. Assume τ_i experiences a self-pushing and let $\tau_{i,k}$ ($k > 1$) be the first job with $R_{i,k}^{FPP} > R_{i,1}^{FPP}$. We show that this contradicts the preemptive feasibility or the constrained deadline assumption.

Consider a "synthetic" job $\tau_{i,s}^*$, consisting of the final subjob $F_{i,k-1}$ and job $\tau_{i,k}$ excluding its final subjob $F_{i,k}$, i.e., $\tau_{i,s}^* \doteq F_{i,k-1} \cup (\tau_{i,k} - F_{i,k})$. Obviously, $\tau_{i,s}^*$ has the same execution time C_i. Job $\tau_{i,s}^*$ is illustrated in Figure 3.
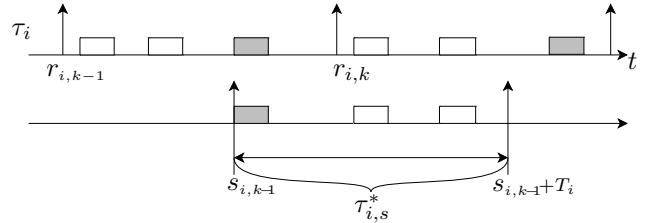

Figure 3. Synthetic task instance $\tau_{i,s}^*$.

From Property 2, we can write:

$$WO_i^{FPP}(C_i) > T_i. \tag{10}$$

Under the FPP model, high-priority tasks arriving during the execution of the final subjob will be deferred to the end of the running task. Since their start times are aligned with the finish time of the current task, the occupied time under the FPP model ($WO^{FPP}(C_i)$) takes such interferences into account. And since, from Property 3, in $[s_{i,k-1}, s_{i,k}]$ the processor is executing only tasks from $hep(i)$, we can write:

$$WO_i^P(C_i) = WO_i^{FPP}(C_i). \tag{11}$$

Now, from Property 1, we know that $s_{i,k-1}$ cannot coincide with the arrival of tasks from $hp(i)$, hence function $WO_i^P(x)$ is left-continuous for $x = C_i$. Thus, using Equation (5), we have:

$$WR_i^P(C_i) = WO_i^P(C_i). \qquad (12)$$

Now, combining Equations (10), (11) and (12) together we have

$$WR_i^P(C_i) > T_i.$$

which contradicts the assumptions and proves the lemma. $\qquad\square$

By using Lemma 2, we can now prove the following theorem.

Theorem 1. *Given a preemptively feasible task set with constrained deadlines, the task set is feasible under fixed priority scheduling with FPP, if the first job of each task is feasible under the critical instant.*

Proof. From Lemma 2, we know that there is no self-pushing phenomenon when tasks are preemptive feasible and have constrained deadlines. Hence, for each task τ_i, the response time of any job $\tau_{i,k}$ will be no greater than the one of the first job at the critical instant. That is, $R_{i,k}^{FPP} \leq R_{i,1}^{FPP}$. Hence, if the first job of each task under the critical instant is feasible, then all the forthcoming jobs will also be feasible. The theorem follows. $\qquad\square$

It is worth pointing out that in the proof of Theorem 1 the value of q_i^{last} is never used, meaning that the theorem holds independently of the value q_i^{last}.

5 Feasibility analysis for the FPP model

In this section, the result stated in Theorem 1 is used to derive a test for checking the feasibility of a set of fixed priority tasks under the FPP model.

Definition 2. *For each task τ_i, the* subjob allowance α_i *is the length of the longest subjob belonging to lower priority tasks in $lp(i)$. That is,*

$$\alpha_i = \max_{\tau_k \in lp(i)} q_k^{max}. \qquad (13)$$

where $q_{n+1}^{max} = 0$ for completeness.

Under fixed priority scheduling with FPP, the presence of non-preemptive subjobs causes the following effects:

On one hand, the non-preemptive execution of any subjob may cause a blocking time to higher priority tasks, however, no job will be blocked after it has started and any job can be blocked for at most once by subjobs belonging to

lower priority tasks. Therefore, the maximum blocking time that τ_i may experience is:

$$B_i = (\lim_{\epsilon \downarrow o} \alpha_i - \epsilon)^+ \qquad (14)$$

where ϵ is an arbitrary small number to guarantee that subjob from $lp(i)$ actually starts before τ_i. The downarrow in the equation denotes the right-hand limit and the notation x^+ stands for $\max\{x, 0\}$, indicating that the blocking time cannot be negative.

On the other hand, since the final subjob cannot be preempted by any other tasks, it will continue to completion once started. Hence, checking the feasibility of a job is equivalent to checking whether the final subjob can start at least q_i^{last} units of time before the deadline.

Taking into account these two effects, the cumulative execution request under the FPP model, denoted as $W_i^{FPP}(t)$, can be represented as:

$$W_i^{FPP}(t) = (C_i - q_i^{last}) + \sum_{\tau_j \in hp(i)} \text{RBF}(\tau_j, t). \qquad (15)$$

Notice that the execution request of the final subjob is excluded in $W_i^{FPP}(t)$. The feasibility condition for the task set using $W_i^{FPP}(t)$ and α_i is stated in the next theorem.

Theorem 2. *A preemptively feasible task set with constrained deadlines and given subjob division is schedulable under fixed priority with FPP, if for each task τ_i there exists $t \in (0, D_i - q_i^{last}]$ such that*

$$W_i^{FPP}(t) + \alpha_i \leq t. \qquad (16)$$

where $W_i^{FPP}(t)$ and α_i are defined in Equation (15) and (13), respectively.

Proof. We first prove the theorem for tasks with $\alpha_i = 0$. If $\alpha_i = 0$, e.g., the lowest priority task τ_n, the blocking time due to lower priority tasks is zero and the feasibility can be verified as in the fully preemptive case.

When $\alpha_i > 0$, let t^* be the earliest time that satisfies Equation (16). Hence, we know there $\exists t^* \leq D_i - q_i^{last}$ and:

$$W_i^{FPP}(t^*) + \alpha_i = t^*.$$

Using Equation (2) and (15), this can be written as:

$$(C_i + \alpha_i - q_i^{last}) + \sum_{\tau_j \in hp(i)} \left\lceil \frac{t^*}{T_j} \right\rceil C_j = t^*.$$

which is equivalent to:

$$WR_i^P(C_i + \alpha_i - q_i^{last}) = t^*. \qquad (17)$$

Since in this proof all WR and WO functions refer to the preemptive model, we omit the P superscript to simplify the notation. The start time of the final subjob of τ_i is given by

$WO_i(C_i + B_i - q_i^{last})$, where B_i is the actual blocking time given by Equation (14). Hence, we have:

$$WO_i(C_i + B_i - q_i^{last}) = \lim_{\epsilon \downarrow 0} WO_i(C_i + \alpha_i - \epsilon - q_i^{last}) \quad (18)$$

According to Equation (5), we have:

$$\lim_{\epsilon \downarrow 0} WO_i(C_i + \alpha_i - \epsilon - q_i^{last}) = WR_i(C_i + \alpha_i - q_i^{last}) \quad (19)$$

Combining Equations (17), (18) and (19) together, we have:

$$WO_i(C_i + B_i - q_i^{last}) = t^*.$$

Therefore, the final subjob will start at t^* and finish at $t^* + q_i^{last}$. Since $t^* \leq D_i - q_i^{last}$, the first job of τ_i meets its deadline and, from Theorem 1, we conclude the entire task is feasible under the FPP model. Hence the theorem follows. $\square$

Condition (16) does not need to be evaluated at every $t \in (0, D_i - q_i^{last}]$, but only at those values of t at which RBF has a discontinuity, i.e. $\{t \in [0, D_i - q_i^{last}] \mid t = k \cdot T_j, \ k \in \mathbb{N} \text{ and } \forall T_j, \ \tau_j \in hp(i)\}$. Moreover, similarly to the methods presented in [4] by Bini and Buttazzo, the number of points can be further reduced to the following set:

$$\mathcal{TS}(\tau_i) \doteq \mathcal{P}_{i-1}(D_i - q_i^{last}). \quad (20)$$

where $\mathcal{P}_i(t)$ is defined by the following recurrent expression:

$$\begin{cases} \mathcal{P}_0(t) = \{t\} \\ \mathcal{P}_i(t) = \mathcal{P}_{i-1}\left(\left\lfloor \frac{t}{T_i} \right\rfloor T_i\right) \cup \mathcal{P}_{i-1}(t) \end{cases} \quad (21)$$

Theorem 2 allows finding the maximum length that subjobs of tasks in $lp(i)$ can have without jeopardizing the feasibility of τ_i. Thus, from Equation (16), the maximum possible value α_i for task τ_i, denoted as *blocking tolerance β_i*, results:

$$\beta_i = \max_{t \in \mathcal{TS}(\tau_i)} \{t - W_i^{FPP}(t)\}. \quad (22)$$

Corollary 1. *Given a preemptively feasible task set with constrained deadlines and a specific subjob division, the task set is feasible under fixed priority if $\forall \tau_i, i > 1$*

$$q_i^{max} \leq \min_{\tau_j \in hp(i)} \{\beta_j\}. \quad (23)$$

where β_j is given by Equation (22).

Proof. The corollary can simply be proved through Theorem 2 and the definition of subjob allowance. Note that q_1^{max} is not used in the test since τ_1 does not cause blocking to any other task. For $i > 1$, if q_i^{max} satisfies Equation (23), then from the definition of subjob allowance we know that $\alpha_j(\tau_j \in hp(i))$ will not exceed β_j, hence the schedulability is guaranteed by Theorem 2. $\square$

Using the value of β_i, we can derive the feasibility condition for each task. The pseudocode for the feasibility check is presented in Algorithm 1. Line 2 sets the initial value for τ_1. The *for-loop* in Line 3 checks the task feasibility one by one, in decreasing priority order, using the condition in Corollary 1. If the algorithm reaches Line 7, then all the tasks will be feasible and the algorithm returns *true*, otherwise, if there is a task with q_i^{max} exceeding the maximum possible value (Line 4), it returns *false*, meaning that the task set cannot be guaranteed.

Input: $\{D_i, C_i, T_i, q_i^{max}, q_i^{last}\}$ for $\forall \tau_i \in \mathcal{T}$, preemptively feasible and $D_i \leq T_i$.
Output: Feasibility of the task set under FPP
1 **begin**
2 $\quad$ $\beta_1 = D_1 - C_1$
3 $\quad$ **for** $i \leftarrow 2$ **to** n **do**
4 $\quad\quad$ **if** $q_i^{max} > \min_{\tau_j \in hp(i)} \{\beta_j\}$ **then**
5 $\quad\quad\quad$ **return** "*false*"
6 $\quad\quad$ Calculate β_i using q_i^{last} by Equation (22)
7 $\quad$ **return** "*true*"
8 **end**

Algorithm 1: Feasibility test for a given task set under fixed priority with FPP.

---

6 Subjob bounds

In this section, we illustrate a method for computing the maximum subjob length for each task under different circumstances and discuss how the feasibility of the task set can be determined as a function of such a bound.

Let Q_i be the maximum possible length that any subjob belonging to τ_i can have, without jeopardizing the system feasibility under FPP. Notice that q_i^{max} and q_i^{last} represent actual lengths in the task code for a given subjob division, whereas Q_i is the upper bound for such lengths. Moreover, Q_i is derived without considering the limitation of the worst-case execution time, hence it can be $Q_i > C_i$.

Corollary 1 already provides a bound for the subjob length of τ_i. However, we now derive an efficient way to compute Q_i recursively.

Since task τ_1 does not cause any blocking to other tasks and it does not experience any interference, we set:

$$\begin{cases} Q_1 = \infty \\ \beta_1 = D_1 - C_1. \end{cases} \quad (24)$$

The next lemma shows how to derive Q_i for the remaining tasks in the system.

Lemma 3. *Given a preemptively feasible task set with constrained deadlines, the maximum length of subjob from task*

$\tau_i, 2 \le i \le n$ *that guarantees feasibility under FPP is given by*

$$Q_i = \min\{\beta_{i-1}, Q_{i-1}\} \qquad (25)$$

where β_{i-1} can be computed by Equation (22) and the initial value for τ_1 is given in Equation (24).

Proof. From Corollary 1, the subjobs length of τ_i must satisfy

$$q_i^{max} \le \min_{\tau_k \in hp(i)} \{\beta_k\}.$$

So the upper bound of the subjob length of τ_i is given by

$$Q_i = \min_{\tau_k \in hp(i)} \{\beta_k\}. \qquad (26)$$

Noting that

$$\min_{\tau_k \in hp(i)} \{\beta_k\} = \min\left\{\beta_{i-1}, \min_{\tau_k \in hp(i-1)} \{\beta_k\}\right\}$$

and that $Q_{i-1} = \min_{\tau_k \in hp(i-1)}\{\beta_k\}$, Equation (26) can be rewritten as

$$Q_i = \min\{\beta_{i-1}, Q_{i-1}\}.$$

which proves the lemma. $\square$

Notice that the bound Q_i for the longest subjob of task τ_i is a function of q_i^{last}. Depending on the knowledge we have on the length of the last subjob, we can distinguish three cases:

- *The value of q_i^{last} is not available.* In this case, the guarantee has to be performed in the worst-case scenario in which τ_i can be preempted arbitrarily near the end of its code. This is equivalent of considering $q_i^{last} = \lim_{\epsilon \downarrow 0} \epsilon$, as done in the floating non-preemptive model. In this case, the upper bound on the subjob length will be denoted as Q_i^{float}.

- *The value of q_i^{last} is given.* In this case, the upper bound Q_i is performed as described above.

- *The value of q_i^{last} is equal to q_i^{max}.* In this case, the upper bound on the subjob length will be the highest and will be denoted as Q_i^*.

The computation of Q_i^* is done in a similar way as presented in Lemma 3, one task at a time in decreasing priority order. The crucial factor now is the value of q_i^{last}, which is set to the maximum possible value, equal to $\min\{C_i, Q_i^*\}$ to compute Q_{i+1}^*.

Observation 1. *Given a preemptively feasible task set with constrained deadlines, in the FPP model we have that*

$$Q_i^* \ge Q_i \ge Q_i^{float} \ge 0.$$

Proof. This can be proved by considering the length of the final subjob. For the case of Q_i^*, q_i^{last} has the largest possible value. On the contrary, for Q_i^{float}, q_i^{last} is an arbitrary small number, while for Q_i, q_i^{last} has an intermediate value between the two cases.

Now, a larger final subjob reduces the interference from higher priority tasks, allowing a larger blocking time from lower priority tasks. Since the maximum subjob length is equal to the minimum blocking tolerance from $hp(i)$, the observation follows. $\square$

Figure 4 qualitatively illustrates the subjob upper bounds for each task under different assumptions on q_i^{last}.
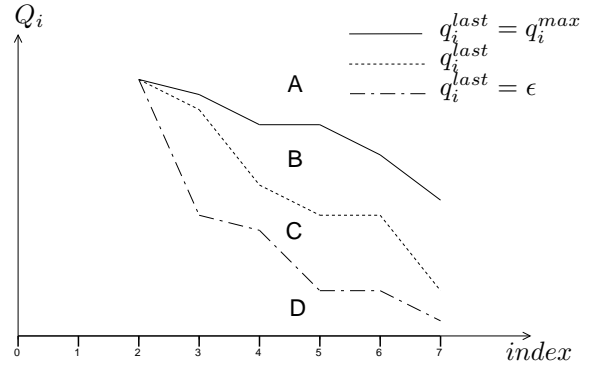

Figure 4. Feasibility regions for a task set.

Notice that all the curves are monotonically non-increasing with the task index. Since for τ_1 all three values are assumed to be infinite, they are not plotted in the figure. For τ_2, all three parameters have the same value $(D_1 - C_1)$, hence, there is only one point for τ_2. For a specific task set, with given subjob division, the sequence of $\{q_i^{max}\}$ values will be denoted as the *task curve*. Plotting the task curve on the graph presented above, we can immediately verify the feasibility of the task set (under assumptions A1 and A2) as follows:

1. There $\exists i,\ q_i^{max} > Q_i^*$. If a part of the task curve exceeds the solid line and falls in Zone A, the task set is not FPP-feasible under fixed priorities.

2. $\forall i,\ q_i^{max} \le Q_i^{float}$. If the task curve falls entirely below the dash-dotted line (Zone D), the corresponding task set is feasible, as proved in [25].

3. $\forall i,\ Q_i^{float} < q_i^{max} \le Q_i^*$. If the task curve falls inside the intermediate area (Zone B and C), the feasibility can be checked by the test presented in Section 5, as a function of q_i^{last}. In particular, the feasibility is guaranteed if the task curve falls totally inside zone C, whereas it is not if a part of the curve falls in zone B. In this case, in fact, there are subjobs exceeding the maximum allowed length.

7 Simulation Results

This section presents some experimental results performed on synthetic task sets to compare the maximum subjob length and the average number of preemptions under different situations.

The task set parameters used in the simulations were randomly generated as follows: The UUniFast algorithm [5] was used to generate a set of n tasks with total utilization equal to U_{tot}. Each computation time C_i was generated as a random integer uniformly distributed in a given interval $[5, 50]$, and then T_i was computed as $T_i = C_i/U_i$. The relative deadline D_i was generated as a random integer in $[C_i + 0.5 \cdot (T_i - C_i), \ T_i]$ and the unfeasible task sets under fully preemptive mode were discarded. In all the graphs, each plotted point represents the average value over 1000 randomly generated task sets.
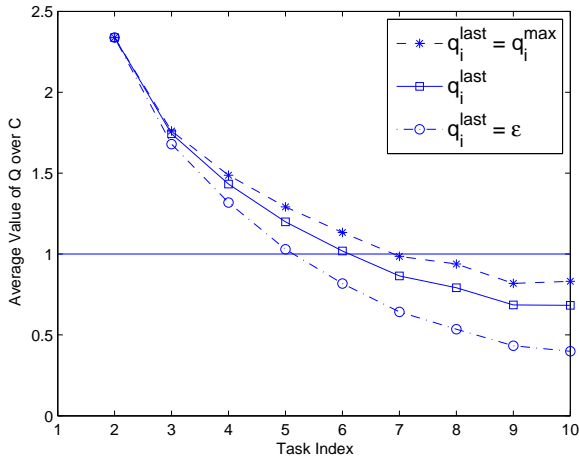

Figure 5. Average value of Q over C when $U_{tot} = 0.9$.

7.1 Experiment 1

In a first experiment, we considered a set of 10 tasks, monitoring the maximum subjob length for each task under different circumstances. Figure 5 plots the average ratio Q_i/C_i for each task when U_{tot} is equal to 0.9. Simulations were performed under different workloads, however, all the three values resulted to be very similar for low utilizations. Since all three values for τ_1 were set to infinity, the curves start from $i=2$. The value of Q_i was computed by Lemma 3 setting q_i^{last} equal to $\min\{C_i/2, \min_{j<i}\{\beta_j\}\}$.

This result shows that the subjob bound is affected by the length of the final subjob. As expected, Q_i^* is the maximum of all these three values and Q_i^{float} is the smallest. Note that the difference becomes larger for tasks with lower priorities.

7.2 Experiment 2

In a second experiment, we monitored the average number of preemptions produced in a run (lasting 1 million units of time) as a function of U_{tot}, under different scenarios. Here U_{tot} was varied from 0.5 to 0.95 with step 0.05 and $n = 15$.

Under the floating condition task τ_i switches to non-preemptive mode for Q_i^{float} units of time when a higher priority task arrives [25]. Under the Q_i^* condition, task τ_i executes non-preemptively if $C_i \leq Q_i^*$, otherwise, preemption points are inserted from the end of task code to the beginning, with Q_i^* length interval, i.e., all the subjobs, except the first one, have length equal to Q_i^*. For the sake of comparison, in the case of Q_i, we assume preemption points are inserted in the same way as in the case of Q_i^*, but with interval equal to Q_i^{float}. Figure 6 reports the ratios of the average number of preemptions under the FPP model with respect to the fully preemptive model, as a function of the system utilization U_{tot}.
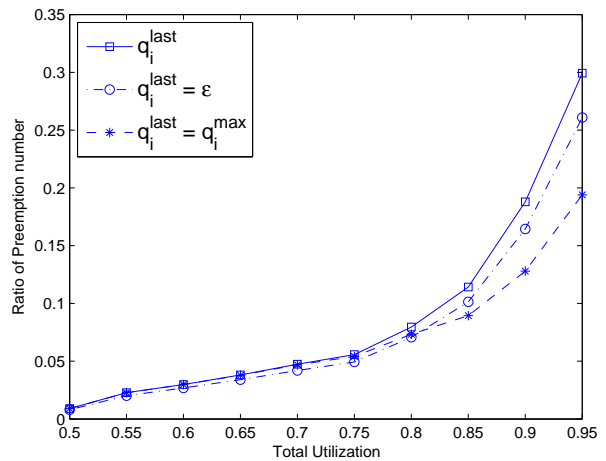

Figure 6. Ratio of number of preemptions with respect to the fully preemptive case, when $n = 15$.

As clear from the figure, the size of the last subjob is not a crucial parameter for reducing the number of preemptions when the task set utilization is low, whereas its influence becomes more relevant for higher workloads. In this condition, setting q_i^{last} to the maximum value achieves the least number of preemptions.

8 Conclusions

In this paper, we considered the problem of analyzing the feasibility of a task set with fixed preemption points un-

der fixed priority scheduling. The feasibility analysis under limited preemptions has been simplified with respect to the existing literature, proving that, under given conditions, guaranteeing the the first job of each task is sufficient for guaranteeing the feasibility of the entire task set. We also presented an algorithm for computing the maximum subjob length for each task, and discussed how such a value changes as a function of the last subjob length. Finally, we provided a global view on all the results under different final subjob length and demonstrated how the task set feasibility can be determined accordingly.

As a future work, we plan to exploit the exact preemption position to better estimate the cost of each preemption and worst-case execution time, thus making the system design more predictable.

References

[1] S. Altmeyer and G. Gebhard. Wcet analysis for preemptive scheduling. In *Proc. of the 8th Intl. Workshop on Worst-Case Execution Time Analysis*, Dagstuhl, Germany, 2008.

[2] T. P. Baker. Stack-based scheduling of real-time processes. *Real-Time Systems*, 3(1):67–100, March 1991.

[3] S. Baruah. The limited-preemption uniprocessor scheduling of sporadic systems. *In Proc. 17th ECRTS*, pages 137–144, July 2005.

[4] E. Bini and G. C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Trans. Comput.*, 53(11):1462–1473, 2004.

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

[6] R. Bril. *Specification and Compositional Verification of Real-Time Systems*. PhD thesis, Technische Universiteit-Eindhoven (TU/e), July 2004.

[7] R. Bril, W. Verhaegh, and J. Lukkien. Exact worst-case response times of real-time tasks under fixed-priority scheduling with deferred preemption. In *Work in Progress (WiP) session of ECRTS'04*, pages 57–60, June 2004.

[8] 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 revisited. In *Proc. of the 19th Euromicro Conference on Real-Time Systems*, pages 269–279, 2007.

[9] A. Burns. Preemptive priority based scheduling: An appropriate engineering approach. *In S. Son, editor, Advances in Real-Time Systems*, pages 225–248, 1994.

[10] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[11] G. Frederickson. Scheduling unit-time tasks with integer release times and deadlines. *Information Processing Letters*, 16:171–173, May 1983.

[12] M. Garey, D. Johnson, B. Simons, and R. Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. volume 10, pages 256–269, May 1981.

[13] G. Gebhard and S. Altmeyer. Optimal task placement to improve cache performance. In *Proceedings of the 7th ACM-IEEE International Conference on Embedded Software (EM-SOFT 07)*, pages 166–171, Salzburg, Austria, 2007.

[14] L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive real-time uniprocessor scheduling. Research Report RR-2966, INRIA, 1996.

[15] K. Jeffay, D. Stanat, and C. Martel. On non-preemptive scheduling of period and sporadic tasks. *Proc. of the 12th Real-Time Systems Symposium*, pages 129–139, Dec 1991.

[16] E. Lawler and C. Martel. Scheduling periodically occurring tasks on multiple processors. *Information Processing Letters*, 12(1):9–12, 1981.

[17] J. 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*, pages 166–171, Santa Monica, California, USA, Dec. 1989.

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

[19] J.-T. Leung and M. Merrill. A note on preemptive scheduling of periodic real-time tasks. *Information Processing Letters*, 11:115–118, 1980.

[20] C. Li, C. Ding, and K. Shen. Quantifying the cost of context switch. In *Proceedings of the 2007 Workshop on Experimental Computer Science*, San Diego, California, 2007.

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

[22] A.-L. Mok. *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*. PhD thesis, MIT, May 1983.

[23] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[24] Y. Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In *Proc. of the 6th Int. Conf. on Real-Time Computing Systems and Applications*, pages 328–335, 1999.

[25] G. Yao, G. Buttazzo, and M. Bertogna. Bounding the maximum length of non-preemptive regions under fixed priority scheduling. In *Proc. of the Conf. on Embedded and Real-Time Computing Systems and Applications*, Beijing, China, August 2009.