Real-Time Syst (2011) 47: 198–223
DOI 10.1007/s11241-010-9113-6

Feasibility analysis under fixed priority scheduling with limited preemptions

Gang Yao · Giorgio Buttazzo · Marko Bertogna

Published online: 14 January 2011

Abstract Preemptive scheduling often generates a significant runtime overhead that may increase task worst-case execution times up to 40%, with respect to a fully non-preemptive execution. In small embedded systems, such an extra cost results in longer and more variable response times that can significantly affect the overall energy consumption, as well as the system predictability. Limiting preemptions is often possible without jeopardizing schedulability. Although several authors addressed schedulability analysis under different forms of limited preemptive scheduling, current results exhibit two major deficiencies: (i) The maximum lengths of the non-preemptive regions for each task are still unknown under fixed priorities; (i) The exact response time analysis for tasks with fixed preemption points is too complex.

This paper presents the schedulability analysis of real-time tasks with non-preemptive regions, under fixed priority assignments. In particular, two different preemption models are considered: the floating and the fixed preemption point model. Under each model, the feasibility analysis is addressed by deriving simple and effective schedulability tests, as well as an algorithm for computing the maximum length of the non-preemptive regions for each task. Finally, simulation experiments are presented to compare the two models in terms of schedulability.

Keywords Limited preemption scheduling · Schedulability analysis · Fixed priority

This work has been partially supported by the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement No. 216008.

G. Yao · G. Buttazzo · M. Bertogna (✉)
Scuola Superiore Sant'Anna, Pisa, Italy
e-mail: m.bertogna@sssup.it

G. Yao
e-mail: g.yao@sssup.it

G. Buttazzo
e-mail: g.buttazzo@sssup.it

1 Introduction

Most of the schedulability tests available for periodic task sets 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, a fully preemptive scheduler is often more efficient in terms of achievable processor utilization. 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 cost need to be taken into account at each preemption (Gebhard and Altmeyer 2007):

- a *scheduling cost* σ, 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* π, 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* γ, 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 (Altmeyer and Gebhard 2008; Gebhard and Altmeyer 2007; Li et al. 2007).

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 (Sha et al. 1990), or Stack Resource Policy (Baker 1991), 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 while limiting the length of non-preemptive execution.

1) *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.
2) *Floating Non-Preemptive Regions (NPR)*. Another approach consists in considering for each task τ_i a number of NPRs, with a maximum length q_i^{max}, whose location is unknown. In this model, NPRs can be considered to be floating in the task code.
3) *Preemption Thresholds (PT)*. A different approach for limiting preemptions is based on the concept of preemption thresholds, proposed by Wang and Saksena (1999) 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. This work has been later improved by Regehr (2002).

It is worth noting that, in the FPP model, the length of the final non-preemptive chunk plays a crucial role for 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. In the floating NPR model, instead, the exact location of each non preemptive region is not known a priori, so that a task could be preempted even an arbitrarily small amount of time before the end of the execution, increasing the resulting response time.

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, therefore improving the estimation of preemption overheads and worst-case execution times (Gebhard and Altmeyer 2007).

In this paper, a simple and effective schedulability test is derived for both the FPP and the floating NPR models, and an efficient algorithm for computing the maximum length of the non-preemptive regions for each task is also illustrated.

a) Motivating example: Let us consider a task set consisting of three 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.

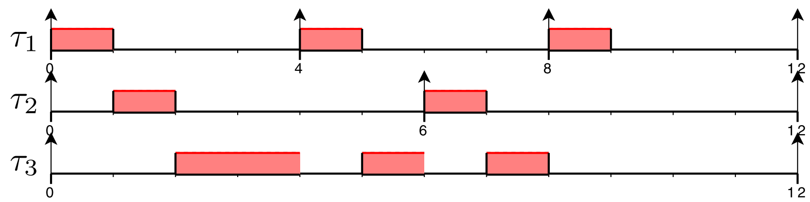
Assuming a synchronous activation of the task set, the schedule produced by Rate Monotonic in a fully preemptive mode (with zero preemption cost) is shown in Fig. 1(a). As clear from the figure, τ_3 is preempted twice and has a response time equal to 8 units of time. When the preemption cost is not negligible, the response time of τ_3 in fully preemptive mode is higher than 9, since the increased execution time causes τ_3 to experience an additional preemption from the third instance of τ_1, as shown in Fig. 1(b). 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, as illustrated in Fig. 1(c).

This simple example clearly shows that the last chunk 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.

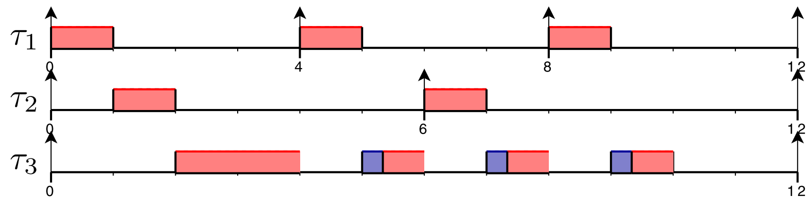
The same example also shows that, under fixed priority assignments, limiting preemptions may also improve schedulability, with respect to fully preemptive scheduling. In fact, if the relative deadline of task τ_3 is set to $D_3 = 7$, the fully preemptive schedules illustrated in Figs. 1(a) and 1(b) become infeasible, while the one shown in Fig. 1(c), generated by FPP, is still feasible (for any task phase).

1.1 Contributions of the paper

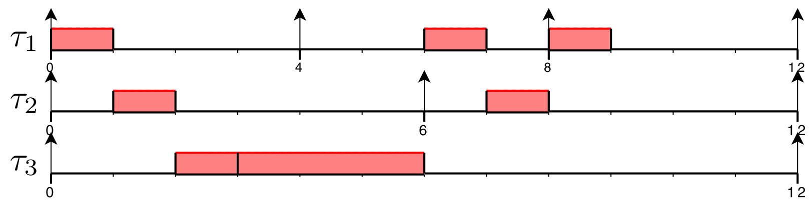
This work provides multiple contributions. First, a schedulability test is provided for fixed priority systems scheduled with limited preemptions, under the floating NPR



(a) Fully preemptive case with zero preemption cost.



(b) Fully preemptive case with preemption cost.



(c) Fixed preemption case with final subjob of τ_3 long 3.

Fig. 1 Fully preemptive vs. FPP scheduling

model. Using this result, an efficient algorithm is derived to compute, for each task, the maximum NPR length that allows all deadlines to be met. This information can be used with a proper scheduler to decrease the number of preemptions of each task.

Under the FPP model, a new task interface is proposed to capture the relevant timing parameters that affect schedulability. A simplified feasibility test is presented for task systems complying with this interface, by identifying the conditions under which the feasibility check of a fixed-priority task set can be limited only to the first instance of each task. This allows deriving a simpler and effective schedulability test, which does not require checking multiple task instances within a certain period, as done in the general case proposed by Bril et al. (2009). Based on this result, an algorithm for computing a bound on the NPR length for each task is presented, discussing how such a bound varies as a function of the length of the final NPR.

This work integrates two previous preliminary results obtained by the same authors for the floating NPR model (Yao et al. 2009) and for the FPP model (Yao et al. 2010). Moreover, it extends the schedulability analysis by introducing preemption costs. However, for the sake of clarity the analysis is first presented without overhead, and then extended by introducing preemption costs.

1.2 Paper organization

The rest of the paper is organized as follows. Section 2 presents some related work. Section 3 describes the task model and the methodology adopted in the paper. Section 4 presents the schedulability analysis for the floating NPR model. The FPP model is analyzed in Sect. 5, deriving a set of conditions under which the response time analysis of fixed priority tasks with given subjob division can be simplified. Section 6 illustrates, for both the floating and the fixed preemption point model, the algorithm for computing the maximum NPR length for each task without violating the system feasibility. Considerations regarding the differences between both preemptive models are presented in Sect. 7. Section 8 reports some simulation results. Finally, Sect. 9 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 (Frederickson 1983; Garey et al. 1981). Non-preemptive tasks were considered in the Spring Kernel (Stankovic and Ramamritham 1991), where a heuristic algorithm was used to find a feasible schedule or reduce the number of deadline misses. A more general characterization of periodic tasks has been considered in Lawler and Martel (1981), Leung and Whitehead (1982). In this model, tasks may have a deadline smaller than or equal to the next release time. For this more general model, Mok (1983) 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. (1991) showed that non-preemptive scheduling of concrete periodic tasks[1] is NP-hard in the strong sense. George et al. (1996) 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. (2007) 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.

Fixed priority scheduling with deferred preemptions, allowed only at some predefined points inside the task code, has been proposed and investigated by Burns (1994), who however did not address the problem of computing the maximum length of non-preemptive chunks. Bril et al. (2009) further improved the response time analysis under this model. The authors 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 one or 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.

Under the floating NPR model, Baruah (2005) computed the longest non-preemptive interval for each task that does not jeopardize the schedulability of the task set under EDF. Yao et al. (2009) addressed the same problem, but under fixed priorities. Later, Yao et al. (2010) extended the analysis under the FPP model and

[1] A concrete periodic task is a periodic task that comes with an assigned initial activation.

presented a comparative study to evaluate the impact on schedulability of different limited preemptive methods (Yao et al. 2010).

When taking preemption overhead into account, the schedulability analysis becomes more complex, because cache-related preemption delays (CRPDs) significantly increase worst-case execution times (Lee et al. 1998; Staschulat and Ernst 2004), which in turn affect the total number of preemptions (Ramaprasad and Mueller 2006), as clearly illustrated in Fig. 1(b). Under the FPP model, however, the negative influence of CRPDs can be alleviated by appropriately selecting the potential preemption points. In Bertogna et al. (2010), a method is proposed to select the preemption points, under the assumption of a fixed preemption cost at each preemption point.

3 Task model and methodology

In this section, we present the task model and the terminology used throughout the paper.

3.1 Task model

We consider a set $\mathcal{T} = \{\tau_1, \tau_2, \dots, \tau_n\}$ 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 (WCET) 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 $\tau_{i,k}$ $(k = 1, 2, \dots)$ 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:

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

Tasks can be preempted, but contain a set of non-preemptive regions (NPRs) where preemption is disabled and deferred until the end of the region. Two kinds of non-preemptive regions are considered:

- *Floating Non-Preemptive Regions*. With this model, the position of each NPR inside the task code is unknown. The only available information is the length q_i^{max} of the longest NPR inside each task. This model has been adopted for instance in Baruah (2005) for EDF scheduling.
- *Fixed Non-Preemptive regions*. With this model, the exact location of each NPR is known, so that the schedulability analysis can potentially take advantage of it, as done in Bril et al. (2007), Burns (1994).

It is worth noting that the first model is more constraining in terms of schedulability, meaning that a feasible task set with floating NPRs is also feasible when the NPRs are located in fixed positions.

The main objective of this work is to compute for each task the longest non-preemptive region that preserves the schedulability with respect to the fully preemptive case. The following notation is used throughout the paper:

$q_i^{\max}$ denotes the duration of the longest non-preemptive region of task τ_i.
Q_i denotes the maximum possible value of $q_i^{\max}$ that preserves the feasibility of the task set with respect to the fully preemptive case.
B_i denotes the blocking time of task τ_i due to the non-preemptive regions of lower priority tasks.
U_{tot} denotes the total utilization of the task set, that is, the sum of all tasks utilizations: $U_{tot} = \sum_{i=1}^{n} C_i/T_i$.

Tasks may access shared resources, provided that each critical section is confined within an NPR. Preemption cost is considered in the schedulability analysis by properly inflating task execution times. For the sake of clarity, however, the analysis is first presented without overhead and later extended by introducing preemption costs.

The analysis for the floating model is presented in Sect. 4, while the analysis for the FPP model is reported in Sect. 5. The rest of this section briefly recalls the main elements used to perform the analysis.

3.2 Request bound function

Schedulability analysis is performed using the *request bound function* $\mathrm{RBF}(\tau_i, t)$, defined as the maximum cumulative execution request that can be generated by jobs of τ_i within an interval of length t. In Lehoczky et al. (1989), it has been shown that

$$\mathrm{RBF}(\tau_i, t) = \left\lceil \frac{t}{T_i} \right\rceil C_i. \tag{1}$$

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

$$W_i(t) = C_i + \sum_{\tau_j \in hp(i)} \mathrm{RBF}(\tau_j, t). \tag{2}$$

A necessary and sufficient schedulability test for fixed priority preemptive tasks was derived by Lehoczky et al. (1989), by checking whether for every task τ_i there exists a value $t \le D_i$ such that $W_i(t) \le t$. This is stated in the following lemma (Lehoczky et al. 1989).

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

$$W_i(t) \le t \tag{3}$$

where $W_i(t)$ *is defined in* (2).

The smallest $t \in \mathbb{R}^+$ that satisfies (3) corresponds to the worst-case response time $WR_i(C_i)$ of τ_i. The inequality does not need to be evaluated at every $t \in (0, D_i]$, but only at those values of t at which RBF has a discontinuity, i.e., $\{t \in [C_i, D_i] \mid t = k \cdot T_j,\ k \in \mathbb{N} \text{ and } \forall T_j,\ j \leq i\}$. Moreover, Bini and Buttazzo further reduced the number of points to be checked to the following set (Bini and Buttazzo 2004):

$$\mathcal{TS}(\tau_i) \doteq \mathcal{P}_{i-1}(D_i) \tag{4}$$

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

$$\begin{cases} \mathcal{P}_0(t) = \{t\} \\ \mathcal{P}_i(t) = \mathcal{P}_{i-1}(\lfloor \frac{t}{T_i} \rfloor T_i) \cup \mathcal{P}_{i-1}(t). \end{cases} \tag{5}$$

The above set $\mathcal{TS}(\tau_i)$ is referred to as the **testing set** for task τ_i. The size of $\mathcal{TS}(\tau_i)$ is *pseudo-polynomial* in the parameters of the task set (Bini and Buttazzo 2004). In the remainder of this paper, $\mathcal{TS}(\tau_i)$ is used to compute the longest NP region for each task.

3.3 Worst-case occupied time

As shown by Bril et al. (2009), the worst-case response time $WR_i(C_i)$ of a task τ_i can be also computed by considering the *worst-case occupied time* $WO_i(C_i)$, which is the longest possible span of time from the release until the time at which the task starts or resumes its execution after the completion of C_i units of computation time. The following relation holds, by taking the limit from the left-hand side:

$$WR_i(C_i) = \lim_{x \uparrow C_i} WO_i(x) \tag{6}$$

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

$$t = x + \sum_{\tau_j \in hp(i)} \left(\left\lfloor \frac{t}{T_j} \right\rfloor + 1 \right) C_j. \tag{7}$$

Notice that, in (7), 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 Fig. 1, the worst-case response time of τ_3 is 8 in Fig. 1(a) and 6 in Fig. 1(c), whereas its worst-case occupied time is 9 in both cases.

3.4 Blocking factor

In the presence of non-preemptive regions, Lemma 1 has to be modified to take into account the additional blocking factor B_i that must be considered for each task τ_i. This blocking factor is equal to the longest NPR belonging to lower priority tasks.

Definition 1 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,

$$\alpha_i = \max_{\tau_k \in lp(i)} q_k^{\max} \tag{8}$$

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

Therefore, the maximum blocking time that τ_i may experience is:

$$B_i = \lim_{\epsilon \downarrow o} (\alpha_i - \epsilon)^+ \tag{9}$$

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.

4 Floating non-preemptive regions model

The schedulability analysis in the presence of blocking factors has been extended by Bini and Buttazzo (2004), where Theorem 4 can be restated as follows by considering floating NPRs:

Theorem 1 *A task set $\mathcal{T}$ with floating NPRs is schedulable with a fixed priority algorithm if and only if $\forall \tau_i \in \mathcal{T}$ there exists $t \in \mathcal{TS}(\tau_i)$ such that*

$$W_i(t) + B_i \le t. \tag{10}$$

Notice that condition (10) is necessary and sufficient for guaranteeing the schedulability when considering floating NPRs, whereas it becomes only sufficient when the regions are fixed. The result of Theorem 1 can be used to determine the maximum amount of blocking a task τ_i can tolerate without missing any of its deadlines. This amount will be called the *blocking tolerance* of task τ_i and will be denoted with β_i. Thus,

$$\beta_i = \max_{t \in \mathcal{TS}(\tau_i)} \{t - W_i(t)\}. \tag{11}$$

Computing β_i with (11) requires the evaluation of all points in the testing set $\mathcal{TS}(\tau_i)$, and has therefore pseudo-polynomial complexity.

Before showing how the blocking tolerance β_i can be used to compute the maximum allowed non-preemptive chunk length of each task, the following section introduces a simplified schedulability analysis for the FPP model. Then, the information on the length of the last NPR is used to derive a larger blocking tolerance than the one given by (11).

5 Fixed preemption points model

In the FPP model, each 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 k^{th} 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 $F_{i,k}$, and its length with $q_i^{last} = q_{i,m_i}$.

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 Fig. 1(c)).

In the schedulability analysis, there is no need to consider the length of all NPRs, but just q_i^{last} and $q_i^{\max}$. Therefore, each task is assumed to be characterized by the following 5-tuple:

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

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.

5.1 Critical instant

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 times of the tasks causing the worst-case response time of τ_i is defined as the critical instant for τ_i (Liu and Layland 1973).

When tasks have non-preemptive regions, Bril (2004) 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. (2009) 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.

5.2 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. Burns and Wellings also recognize their relevance in the analysis of non-preemptive tasks (Burns and Wellings 2009), although they are not formally used to derive the results.

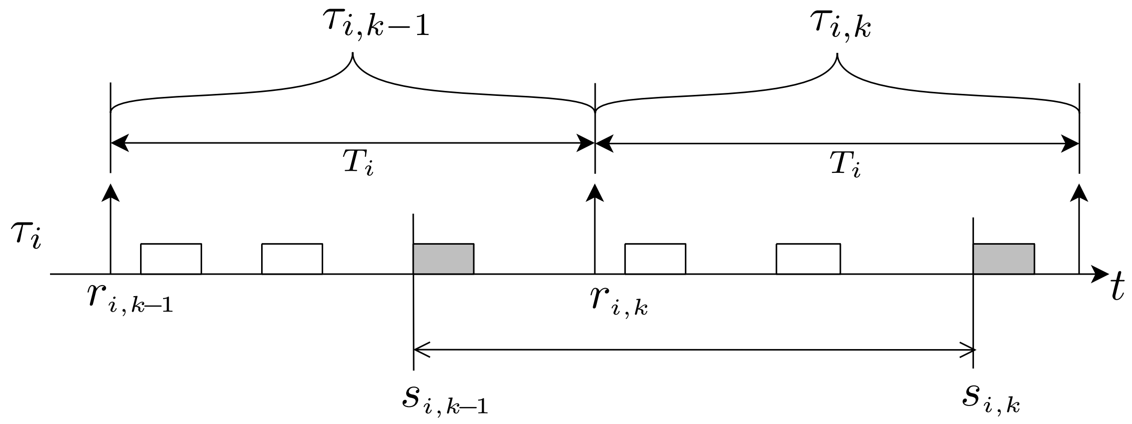


Fig. 2 The self-pushing phenomenon

In the following, we formally prove that conditions A1 and A2 allow simplifying the feasibility test by restricting the analysis to the first job of each task under the critical instant. 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.

5.3 Properties of the self-pushing scenario

Definition 2 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:

$$\exists k > 1, \quad R_{i,k}^{FPP} > R_{i,1}^{FPP}. \tag{12}$$

Notice that $R_{i,k}^{FPP}$ denotes the generic response time of one job while $R_{i,1}^{FPP}$ is the one under critical instance. 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 Fig. 2, where the final subjobs are depicted in gray. The following properties can be derived on time interval $[s_{i,k-1}, s_{i,k}]$.

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

Proof Since $F_{i,k-1}$ cannot be preempted during its execution, let us consider the start time $s_{i,k-1}$ of $F_{i,k-1}$. 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

$$R_{i,k}^{FPP} = s_{i,k} + q_i^{last} - r_{i,k} > R_{i,1}^{FPP}. \tag{13}$$

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

$$R_{i,k-1}^{FPP} = s_{i,k-1} + q_i^{last} - r_{i,k-1} \leq R_{i,1}^{FPP}. \tag{14}$$

Combining (13) and (14), and noticing that $r_{i,k} \geq r_{i,k-1} + T_i$, we have

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

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 at or after time instant t', some tasks from $hp(i)$ and $lp(i)$ are activated or the processor becomes idle, 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, $R_{i,k}^{FPP} \leq R_{i,1}^{FPP}$, which contradicts the self-pushing assumption and proves the property. □

5.4 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 Fig. 3. We assume this job arrives at time $s_{i,k-1}$. Since at this time all tasks from $hp(i)$ are finished and subjob $F_{i,k-1}$ can start, the synthetic job will also start upon arrival.

From Property 2, the occupied time of this job, denoted as $O_i^{FPP}(C_i)$, can be expressed as

$$O_i^{FPP}(C_i) = s_{i,k} - s_{i,k-1} > T_i. \tag{15}$$

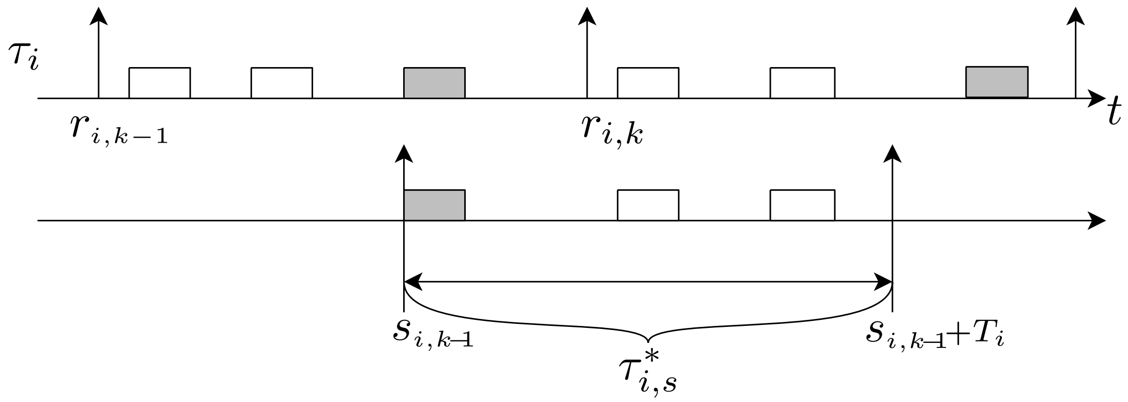


Fig. 3 Synthetic task instance $\tau^*_{i,s}$

Under the FPP model, high-priority tasks arriving during the execution of the final subjob are 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 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 $hep(i)$, job $\tau^*_{i,s}$ suffers no blocking from $lp(i)$. Therefore, the occupied time for this job under P and FPP model will be the same, that is:

$$O_i^P(C_i) = O_i^{FPP}(C_i). \tag{16}$$

Now, from Property 1, we know that $s_{i,k-1}$ cannot coincide with the arrival of tasks from $hp(i)$, hence, the worst-case for job $\tau^*_{i,s}$ is that all tasks from $hp(i)$ arrive at the same time $\epsilon(\epsilon \downarrow o)$ after $s_{i,k-1}$ and function $WO_i^P(x)$ is left-continuous at C_i. Using (6), we have:

$$WR_i^P(C_i) = WO_i^P(C_i) \geq O_i^P(C_i). \tag{17}$$

Now, combining (15), (16) and (17) together:

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

which means that a job with the same parameters as task τ_i will have response time larger than T_i. This contradicts the assumptions and proves the lemma. □

Using Lemma 2, we can prove the following theorem.

Theorem 2 *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 preemptively 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. □

It is worth pointing out that in the proof of Theorem 2 the value of q_i^{last} is never used, meaning that the theorem holds independently of the value q_i^{last}.

5.5 Sufficient schedulability test for the FPP model

In this section, the result stated in Theorem 2 is used to derive a test for checking the feasibility of a set of fixed priority tasks under the FPP model.

Since the final subjob of each task 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:

$$W_i^{FPP}(t) = (C_i - q_i^{last}) + \sum_{\tau_j \in hp(i)} \mathrm{RBF}(\tau_j, t). \tag{18}$$

Notice that the execution request of τ_i's final subjob (q_i^{last}) 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 3 *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*

$$W_i^{FPP}(t) + \alpha_i \leq t \tag{19}$$

where $W_i^{FPP}(t)$ and α_i are defined in (18) *and* (8), *respectively.*

Proof We first prove the theorem for tasks with $\alpha_i = 0$. If $\alpha_i = 0$, e.g., for the lowest priority task τ_n, the blocking time due to lower priority tasks is zero. Since the non-preemptive execution of subjobs will only possibly reduce the interference and the blocking time is always zero, hence the feasibility can be verified as in the fully preemptive case, which is feasible by assumption, independently of (19).

When $\alpha_i > 0$, let t^* be the earliest time that satisfies (19). Hence, there $\exists t^* \leq D_i - q_i^{last}$ and:

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

Using (1) and (18), 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:

$$WR_i^P(C_i + \alpha_i - q_i^{last}) = t^*. \tag{20}$$

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 (9). Hence, we have:

$$WO_i(C_i + B_i - q_i^{last}) = \lim_{\epsilon \downarrow 0} WO_i(C_i + \alpha_i - \epsilon - q_i^{last}) \tag{21}$$

According to (6), we have:

$$\lim_{\epsilon \downarrow 0} WO_i(C_i + \alpha_i - \epsilon - q_i^{last}) = WR_i(C_i + \alpha_i - q_i^{last}) \tag{22}$$

Combining (20), (21) and (22) together:

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

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 2, we conclude the entire task is feasible under FPP model. Hence the theorem follows. □

Condition (19) 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, as we did for Lemma 1, the number of points can be further reduced to the following set (Bini and Buttazzo 2004):

$$\mathcal{TS}'(\tau_i) \doteq \mathcal{P}_{i-1}(D_i - q_i^{last}). \tag{23}$$

where $\mathcal{P}_i(t)$ is defined in (5).

Theorem 3 allows finding the maximum length that subjobs of tasks in $lp(i)$ can have without jeopardizing the feasibility of τ_i. Thus, from (19), the blocking tolerance β_i for each task τ_i results

$$\beta_i = \max_{t \in \mathcal{TS}'(\tau_i)} \{t - W_i^{FPP}(t)\}. \tag{24}$$

As we will prove in Sect. 7, the value of β_i given by (24) is greater than or equal to the one given by (11). This means that the blocking tolerances in the floating case cannot be larger than in the FPP case. As expected, the FPP model can take advantage of the smaller response time allowed by the deterministic location of the last NPR, reducing the interference from higher priority tasks.

Similarly to the floating case, the computation of β_i requires the evaluation of a pseudo-polynomial number of points in the testing set.

6 Longest non-preemptive regions

Starting with a fully preemptive task set $\mathcal{T}$, which is schedulable with a fixed priority algorithm, this section shows how to determine, for each task τ_i, the largest possible NPR preserving system schedulability, both for the floating and the FPP model.

Let Q_i be the maximum possible length that any NPR belonging to τ_i can have, without jeopardizing the system feasibility under limited preemption scheduling with fixed priority. Note that Q_i is derived without considering the limitation of the worst-case execution time, hence it can be $Q_i > C_i$.

Since τ_1 is the highest priority task, its longest NPR can be arbitrarily large without making the system unschedulable. Therefore,

$$Q_1 = \infty.$$

The next theorem shows how to derive Q_i for the other tasks.

Theorem 4 *Given a preemptively feasible task set with constrained deadlines, the task set is feasible under fixed priority with a limited preemption scheduler if* $\forall \tau_i$, $i > 1$

$$q_i^{\max} \le Q_i \doteq \min_{\tau_k \in hp(i)} \{\beta_k\}, \tag{25}$$

where β_k *is given by* (11) *in the floating NPR case, and by* (24) *in the FPP case.*

Proof The blocking time experienced by a task τ_k is bounded by

$$B_k = \max_{\tau_i \in lp(k)} \{q_i^{\max}\}.$$

Since β_k is the blocking tolerance of τ_k, the schedulability of the task set can be expressed as follows:

$$\forall k \mid 1 \le k < n: \quad B_k \le \beta_k,$$

that is

$$\forall k \mid 1 \le k < n: \quad \max_{\tau_i \in lp(k)} \{q_i^{\max}\} \le \beta_k.$$

Note that β_n is not considered, because the lowest priority task can never be blocked, so that if it was schedulable with a fully preemptive scheduler, it is still schedulable with limited preemptions. The schedulability condition for the remaining tasks can be expressed in the following form

$$\bigwedge_{1 \le k < n} \left(\max_{\tau_i \in lp(k)} \{q_i^{\max}\} \le \beta_k \right).$$

The inner inequality can be written as a system of inequalities, as follows:

$$\bigwedge_{1 \le k < n} \left(\bigwedge_{k < i \le n} \left(q_i^{\max} \le \beta_k\right) \right)$$

which can be rewritten as

$$\bigwedge_{1<i\le n}\left(\bigwedge_{1\le k<i}\left(q_i^{\max}\le\beta_k\right)\right)$$

which is equivalent to

$$\forall i \mid 1<i\le n\colon\quad q_i^{\max}\le\min_{1\le k<i}\{\beta_k\},$$

or

$$\forall i>1\colon\quad q_i^{\max}\le\min_{\tau_k\in hp(i)}\{\beta_k\}=Q_i,$$

proving the theorem. □

Another way to derive Q_i is given by the following corollary.

Corollary 1 *Given a preemptively feasible task set with constrained deadlines, the maximum NPR length of each task $\tau_i, 2\le i\le n$, that guarantees feasibility under fixed priority with floating or fixed preemption points is given by*

$$Q_i=\min\{\beta_{i-1},Q_{i-1}\},\tag{26}$$

where $Q_1=\infty$, and β_{i-1} can be computed by (11) *in the floating NPR case, and by* (24) *in the FPP case.*

Proof From Theorem 4, the upper bound on the maximum NPR length of τ_i is given by

$$Q_i=\min_{\tau_k\in hp(i)}\{\beta_k\}.\tag{27}$$

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

$$Q_{i-1}=\min_{\tau_k\in hp(i-1)}\{\beta_k\},$$

(27) can be rewritten as

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

which proves the corollary. □

Hence, the maximum NPR length Q_i allowed for each task can be easily derived from the β_i values. The pseudo-code of the algorithm that computes Q_i for each task and checks the task set schedulability is presented in Algorithm 1. The procedure works for both the floating and the FPP model, computing β_i using (11) or (24), respectively. Note that the additional parameter q_i^{last} (the length of the last NPR) must by provided as an input in the FPP case. Lines 2 and 3 set the initial values for τ_1. The *for-loop* in Line 4 checks the task feasibility one by one, in decreasing priority order, using the condition in Corollary 1. If the algorithm reaches Line 9, then all the tasks are feasible and the algorithm returns *true*, providing as output the maximum allowed NPR length Q_i for each task τ_i. Otherwise, if there is a task with q_i^{max} exceeding the maximum possible value (Line 6), the procedure returns *false*, meaning that the task set is not schedulable.

Algorithm 1: Limited_Preemption_Test

Input: $\{D_i, C_i, T_i, q_i^{max}, q_i^{last}\}, \forall \tau_i \in \mathcal{T}$ preemptively feasible and $D_i \leq T_i$
Output: $Q_i, \forall \tau_i \in \mathcal{T}$ and feasibility of the task set under floating or fixed limited preemption scheduling

```
1  begin
2      β_1 = D_1 − C_1;
3      Q_1 = ∞;
4      for i ← 2 to n do
5          Q_i = min{Q_{i−1}, β_{i−1}};
6          if q_i^max > Q_i then
7              return "false"
8          Calculate β_i using (11) or (24);
9      return "true";
10 end
```

7 Considerations

This section presents some considerations on the different preemption models analyzed in this paper.

7.1 Maximum allowed NPR lengths

As mentioned in the previous sections, the maximum allowed non-preemptive chunk length in the FPP case is larger than in the floating NPR case. It is worth pointing out that the value of Q_i for task τ_i only depends on $\beta_j(\tau_j \in hp(i))$, as expressed in (25). In the FPP case, according to (18) and (24), the blocking tolerance β_i is a function of q_i^{last}. However, q_i^{last} does not *directly* affect Q_i, but only the value of β_i, which is used to compute $Q_j(\tau_j \in lp(i))$. Depending on the knowledge we have on the length of the last subjob, three cases can be distinguished:

1) *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}.

2) *The value of* q_i^{last} *is given*. In this case, the upper bound, denoted as Q_i^g, is computed as described in Sect. 6.
3) *The value of* q_i^{last} *is the maximum possible one*. This is the best case, and the upper bound, denoted as Q_i^*, results to be the highest.

In practice, q_i^{last} might be limited to a certain value, because preemption points cannot be placed at arbitrary positions for a number of reasons, depending on the task structure, the presence of critical sections, or the presence of non-preemptive kernel primitives. As a consequence, Q_i^* represents a best-case upper bound for $q_i^{\max}$, computed to have a reference value in the evaluation. In particular, Q_i^* is computed using Algorithm 2.

Algorithm 2: Q_i^* compute

```
Input: {D_i, C_i, T_i}, ∀τ_i ∈ T preemptively feasible and D_i ≤ T_i
Output: Q_i^*, ∀τ_i ∈ T
begin
    β_1 = D_1 − C_1;
    Q_1^* = ∞;
    for i ← 2 to n do
        Q_i^* = min{Q_{i−1}^*, β_{i−1}};
        q_i^last = min{Q_i^*, C_i};
        Calculate β_i using (24);
end
```

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

$$Q_i^* \geq Q_i^g \geq Q_i^{float} \geq 0.$$

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; for Q_i^{float}, q_i^{last} is an arbitrary small number; whereas for Q_i^g, q_i^{last} has an intermediate value between the two extreme 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. □

7.2 Preemption overhead

Another difference between the floating and the fixed NPR model is in the impact they have on overall WCET of each task. The FPP model allows better timing predictability, because the number of points at which a task can be preempted is fixed and known at compile time. The preemption cost can then be upper bounded using suitable timing analysis tools that take into account cache-related preemption delays. In the floating NPR model, instead, a preemption can take place anywhere in the task code, so the timing analysis needs to consider the preemption cost at each single instruction, similarly to what is done for a fully preemptive scheduler. However, since a task can only be preempted at a limited subset of points, a complex analysis is needed to find the worst-case combination of preemption points that leads to the largest preemption overhead. To simplify the analysis, a pessimistic upper bound is typically computed taking the largest preemption cost ξ_i among all the instructions, and multiplying it by the maximum number ν_i of preemptions a task τ_i can experience. As mentioned in Sect. 1, ξ_i must take into account the cost related to the scheduler (σ), to the pipeline flushing (π) and to the cache (γ_i), hence $\xi_i = \sigma + \pi + \gamma_i$.

Without further information, the maximum number of preemptions that τ_i can experience under the floating NPR model can be as pessimistic as the one derived in the fully preemptive case. Remember that for the floating model $q_i^{\max}$ is just an *upper bound* on the maximum NPR length, so that a task could even execute in a fully preemptive way. However, if the scheduler can be modified to disable preemptions at the occurrence of a preemption request, the following variation of the floating NPR model can be defined to bound the number of preemptions independently of the other tasks.

Definition 3 Preemption-triggered NPR model Each task can run in two modes: *Normal* and *Non-preemptive*. When a task τ_i starts executing, it runs in Normal mode. As soon as a higher priority job arrives, the running task switches to Non-preemptive mode for at most Q_i time-units from the preemption request. If after this amount of time τ_i has not finished its execution, a preemption takes place, and the highest priority task is scheduled for execution.

When the preemption cost is not considered, it is easy to prove that the number of preemptions under the preemption-triggered NPR model is upper bounded by $\lceil C_i/Q_i \rceil - 1$. Note that this estimation is independent of the number of tasks in the system, which might be rather large in some practical applications. As showed in Yao et al. (2009), under fixed priority scheduling, the average value of Q_i/C_i is usually greater than 0.5 for a ten-tasks system, even under high system utilizations (90%).[2] Thus, the number of preemptions can be significantly decreased with this method.

When considering preemption costs, the number of preemptions ν_i experienced by a task τ_i is more difficult to compute, because the task execution time depends on ν_i. If C_i^{np} denotes the WCET of the task when executed non preemptively, then the WCET in the presence of preemption is:

$$C_i = C_i^{np} + \nu_i \times \xi_i.$$

[2] In Yao et al. (2009), all results are derived ignoring preemption cost.

But, since ν_i also depends on C_i, there is a circular dependency between both parameters. Such a dependency can be treated using the following recurrent relation:

$$\begin{cases} \nu_i^0 = \lceil \frac{C_i^{np}}{Q_i} \rceil - 1 \\ \\ \nu_i^s = \lceil \frac{C_i^{np} + \xi_i \nu_i^{s-1}}{Q_i} \rceil - 1 \end{cases}$$

where ξ_i is the maximum preemption overhead related to task τ_i. When the iteration process converges, $\nu_i^s = \nu_i^{s-1}$ is the worst-case estimation. Notice the value Q_i is calculated from $hp(i)$, hence, it is available if the computation is performed in decreasing priority order.

Under the FPP model, the maximum number of preemptions is much easier to compute and is equal to the number of preemption points $(m_i - 1)$ in the code. A more difficult related problem is instead selecting the fixed preemption points when they are not given a priori. The selection can be made at design time based on the information on the preemption overhead given by timing analysis tools. Suppose the preemption cost at each program point is known. The designer would like to select the preemption points in an optimal way, so that the chances of finding a feasible solution are maximized. One preliminary result is presented in Bertogna et al. (2010) to select the least number of preemption points, under the assumption of a fixed preemption cost at each point.

7.3 Increasing schedulability

This paper considered task sets that are preemptively schedulable with fixed priorities, and analyzed the feasibility under limited preemptive scheduling with given NPRs. An interesting question is whether there is a dominance of the limited preemption model with respect to fully preemptive and fully non-preemptive scheduling, when NPRs can be freely assigned.

It is easy to see that a preemptively feasible task set is also feasible with a limited preemption scheduler, using $q_i^{max} = 0, \forall i$. Similarly, a task set that is feasible under fully non-preemptive scheduling is also feasible with a limited preemption scheduler, using $q_i^{max} = C_i, \forall i$. Moreover, there exist fixed priority task sets that are unfeasible under both fully preemptive and fully non-preemptive scheduling, but can be successfully scheduled under a limited preemption model. For instance, consider the following example.

Example 1 A task system is composed of two sporadic tasks $\{\tau_1 = \{2, 4\}, \tau_2 = \{3, 6\}\}$. It can be easily verified that this task set is unfeasible under fixed priority fully preemptive and non-preemptive scheduling. However, the task set is feasible under the preemption-triggered NPR model, with $Q_2 = 2 - \epsilon$, and, it is also feasible under the FPP model, by setting $q_2^{last} = q_2^{max} = 2$.

Hence, the limited preemption scheduling dominates both fully preemptive and non-preemptive scheduling.

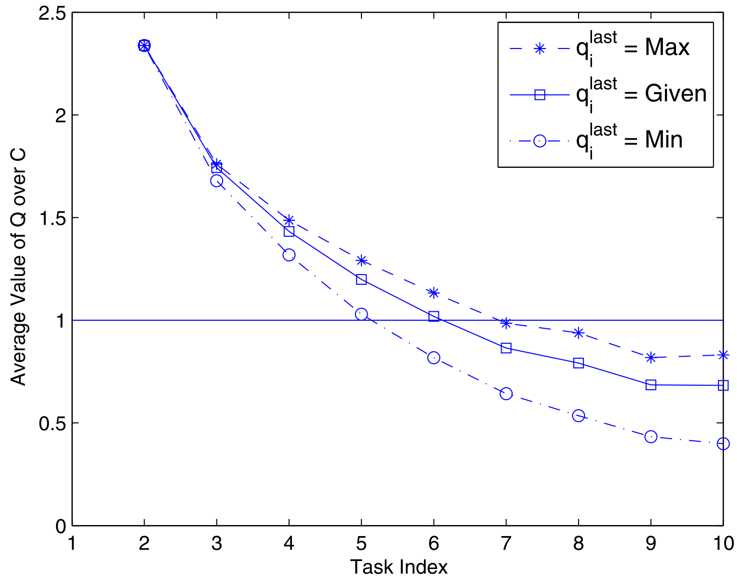


Fig. 4 Average value of Q_i/C_i for each task and for different length of the final NPR

8 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 (Bini and Buttazzo 2005) 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.

8.1 Experiment 1: maximum NPR length

A first experiment was carried out to evaluate how the length of the final NPR affects the maximum subjob length for each task. The test was performed on a set of 10 tasks with total utilization $U_{tot} = 0.9$. Figure 4 plots the average ratio Q_i/C_i achieved for each task under the three conditions on the last NPR explained at the beginning of Sect. 7.1.

Simulations were performed under different workloads, however, the values obtained under the three conditions resulted to be very similar for low utilizations. Note that, since Q_1 was set to infinity in all the three cases, the curves start from $i = 2$. The value of Q_i^g (intermediate curve) was computed with Algorithm 1, by 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^* (upper curve) is the maximum of the three values and Q_i^{float} (lower curve) is the smallest. Note that the difference becomes larger for tasks with lower priorities (i.e., higher index). This can be explained because the lower priority tasks

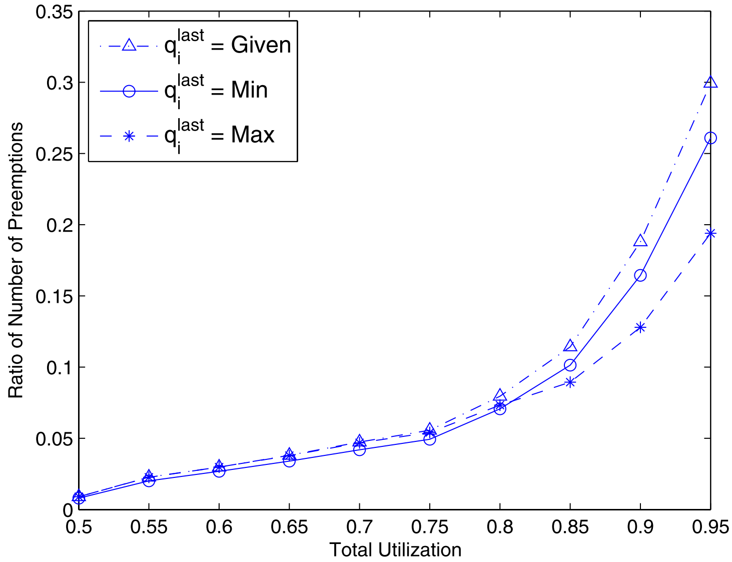


Fig. 5 Ratio of number of preemptions with respect to the fully preemptive case

have a higher chance to be preempted by high priority tasks, therefore, the length of the final subjob becomes more crucial: a higher value of q_i^{last} leads to a larger blocking tolerance and consequently a larger Q.

8.2 Experiment 2: average number of preemptions

A second experiment was carried out to monitor the average number of preemptions produced in a run (lasting 1 million units of time) as a function of U_{tot}, under the three different scenarios for q_i^{last}. Here, the test was performed on a set of 15 tasks whose total utilization U_{tot} was varied from 0.5 to 0.95 with step 0.05.

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, as in the preemption-triggered NPR model. 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 intervals long Q_i^* (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^g, preemption points are inserted in the same way as in the case of Q_i^*, but with interval length equal to Q_i^{float} ($Q_i^g = Q_i^{float}$). Figure 5 reports the average number of preemptions with respect to the fully preemptive case, for the three different scenarios, as a function of the system utilization U_{tot}.

As clearly showed in 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.

It is interesting to point out the subtle differences between Q_i^g and Q_i^{float}. Under the floating model, each preemption is deferred by Q_i^{float} units of time, unless the remaining execution time of the running task is less than Q_i^{float}. In the other case, instead, preemption points are inserted at fixed intervals of Q_i^g, hence, each preemption is deferred to the next point and the average deferred time is only around $Q_i^g/2$.

Since task computation time is fixed and $Q_i^g = Q_i^{float}$, the Q_i^g case generates more preemptions than the Q_i^{float} case, which is validated by the simulation results.

9 Conclusions

In this paper, we considered the problem of analyzing the feasibility of a task set with limited preemptions under fixed priority scheduling. Two different preemption models have been considered in detail: (i) the floating NPR model, in which no information is available on the location of the preemption points, and (ii) the FPP model, in which the location of each preemption point is specified a priori. The feasibility analysis under FPP has been simplified with respect to the existing literature, proving that, under given conditions (i.e, preemptive feasibility and constrained deadlines) guaranteeing the first job of each task is sufficient for the entire task set. Based on this result, an efficient feasibility test was presented. Another contribution of this work was the development of an algorithm for computing the maximum subjob length for each task, under both the floating and the FPP preemption model. Specific analysis has been carried out to investigate how such a value changes as a function of the final subjob length. Experimental simulations on randomly generated task sets validated the proposed approach and provided more quantitative results.

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

References

Altmeyer S, Gebhard G (2008) Wcet analysis for preemptive scheduling. In: 8th int. workshop on worst-case execution time analysis, Prague, Czech, pp 105–112

Baker TP (1991) Stack-based scheduling of real-time processes. Real-Time Syst 3(1):67–100

Baruah S (2005) The limited-preemption uniprocessor scheduling of sporadic systems. In: ECRTS '05: proc. of Euromicro conf. on real-time systems, pp 137–144

Bertogna M, Buttazzo G, Marinoni M, Yao G, Esposito F, Caccamo M (2010) Preemption points placement for sporadic task sets. In: Proceedings of 22nd Euromicro conference on real-time systems (ECRTS'10), Bruxelles, Belgium

Bini E, Buttazzo GC (2004) Schedulability analysis of periodic fixed priority systems. IEEE Trans Comput 53(11):1462–1473

Bini E, Buttazzo GC (2005) Measuring the performance of schedulability tests. Real-Time Syst 30(1–2):129–154

Bril R (2004) Specification and compositional verification of real-time systems. PhD thesis, Technische Universiteit Eindhoven (TU/e)

Bril R, Lukkien J, Verhaegh W (2009) Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. Real-Time Syst 42(1–3):63–119

Bril RJ, Lukkien JJ, Verhaegh WFJ (2007) Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred revisited. In: ECRTS '07: proc. of Euromicro conf. on real-time systems, pp 269–279

Burns A (1994) Preemptive priority based scheduling: an appropriate engineering approach. In: Son S (ed) Advances in real-time systems, pp 225–248

Burns A, Wellings A (2009) In: Real-time systems and programming languages: ADA, real-time Java and C/real-time POSIX (44th edn.). Addison-Wesley, Reading

Davis RI, Burns A, Bril RJ, Lukkien JJ (2007) Controller area network (CAN) schedulability analysis: refuted, revisited and revised. Real-Time Syst 35(3):239–272

Frederickson G (1983) Scheduling unit-time tasks with integer release times and deadlines. Inf Process Lett 16(4):171–173

Garey M, Johnson D, Simons B, Tarjan R (1981) Scheduling unit-time tasks with arbitrary release times and deadlines. SIAM J Comput 10(2):256–269

Gebhard G, Altmeyer S (2007) Optimal task placement to improve cache performance. In: Proc. of the ACM-IEEE int. conf. on embedded software, Salzburg, Austria, pp 259–268

George L, Rivierre N, Spuri M (1996) Preemptive and non-preemptive real-time uniprocessor scheduling. Research report RR-2966, INRIA, France

Jeffay K, Stanat D, Martel C (1991) On non-preemptive scheduling of period and sporadic tasks. In: Proc. of real-time systems symposium, pp 129–139

Lawler E, Martel C (1981) Scheduling periodically occurring tasks on multiple processors. Inf Process Lett 12(1):9–12

Lee C-G, Hahn J, Seo Y-M, Min SL, Ha R, Hong S, Park CY, Lee M, Kim CS (1998) Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. IEEE Trans Comput 47(6):700–713

Lehoczky J, Sha L, Ding Y (1989) The rate monotonic scheduling algorithm: exact characterization and average case behavior. In: Proc. of the real-time systems symposium, CA, USA, pp 166–171

Leung J, Whitehead J (1982) On the complexity of fixed-priority scheduling of periodic real-time tasks. Perform Eval 2(4):237–250

Li C, Ding C, Shen K (2007) Quantifying the cost of context switch. In: Proc. of workshop on experimental computer science, San Diego, California

Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J ACM 20(1):46–61

Mok A-L (1983) Fundamental design problems of distributed systems for the hard real-time environment. PhD thesis, MIT, USA

Ramaprasad H, Mueller F (2006) Tightening the bounds on feasible preemption points. In: RTSS '06. proc. of 27th real-time systems symposium, pp 212–222

Regehr J (2002) Scheduling tasks with mixed preemption relations for robustness to timing faults. In: Proc. of the 23rd IEEE real-time systems symposium, pp 315–326

Sha L, Rajkumar R, Lehoczky JP (1990) Priority inheritance protocols: an approach to real-time synchronization. IEEE Trans Comput 39(9):1175–1185

Stankovic JA, Ramamritham K (1991) The spring kernel: a new paradigm for real-time systems. IEEE Softw 8(3):62–72

Staschulat J, Ernst R (2004) Multiple process execution in cache related preemption delay analysis. In: Proc. of ACM int. conf. on embedded software, Pisa, Italy, pp 278–286

Wang Y, Saksena M (1999) Scheduling fixed-priority tasks with preemption threshold. In: Proc. of conf. on embedded and real-time computing systems and applications, pp 328–335

Yao G, Buttazzo G, Bertogna M (2009) Bounding the maximum length of non-preemptive regions under fixed priority scheduling. In: Proceedings of the 16th IEEE international conference on embedded and real-time computing systems and applications (RTCSA 2009), Beijing, China, August 24–26, 2009, pp 351–360

Yao G, Buttazzo G, Bertogna M (2010) Comparative evaluation of limited preemptive methods. In: Proceedings of the 15th IEEE international conference on emerging technology and factory automation (ETFA10), Bilbao, Spain, September 13–16, 2010.

Yao G, Buttazzo G, Bertogna M (2010) Feasibility analysis under fixed priority scheduling with fixed preemption points. In: Proceedings of the 16th IEEE international conference on embedded and real-time computing systems and applications (RTCSA 2010), Macao, China, August 23–25, 2010, pp 71–80

Gang Yao is a Postdoctoral Research Collaborator at the University of Illinois at Urbana Champagne. He received a Ph.D. in Computer Engineering from the Scuola Superiore Sant'Anna of Pisa, Italy, in 2010. His main interests include limited-preemption scheduling algorithms, fixed-priority systems and shared resource protocols.

Giorgio Buttazzo is Full Professor of Computer Engineering at the Scuola Superiore Sant'Anna of Pisa. He graduated in Electronic Engineering at the University of Pisa in 1985, received a Master in Computer Science at the University of Pennsylvania in 1987, and a Ph.D. in Computer Engineering at the Scuola Superiore Sant'Anna of Pisa in 1991. From 1987 to 1988, he worked on active perception and real-time control at the G.R.A.S.P. Laboratory of the University of Pennsylvania, Philadelphia. Prof. Buttazzo has been Program Chair and General Chair of the major international conferences on real-time systems. He is Chair of the IEEE Technical Committee on Real-Time Systems and member of the Euromicro Executive Board on Real-Time Systems. He has authored 6 books on real-time systems and over 200 papers in the field of real-time systems, robotics, and neural networks. He is Senior Member of IEEE.

Marko Bertogna is Assistant Professor at the Scuola Superiore Sant'Anna of Pisa, Italy. He graduated (summa cum laude) in Telecommunications Engineering at the University of Bologna in 2002, and received a Ph.D. in Computer Engineering from Scuola Superiore Sant'Anna of Pisa in 2008. In 2006 he visited the University of North Carolina at Chapel Hill, working with prof. Sanjoy Baruah on scheduling algorithms for single and multicore real-time systems. His research interests include scheduling and schedulability analysis of real-time multiprocessor systems, protocols for the exclusive access to shared resources, resource reservation algorithms and reconfigurable devices. He has authored over 30 papers in international conferences and journals in the field of real-time systems, receiving four Best Paper Awards. He is Member of IEEE.