Robust Schedulability Tests for Fixed Job Priorities: Addressing Context Switch Costs with Non-Resumable Delays

Joël Goossens, Damien Masson

Robust Schedulability Tests for Fixed Job Priorities: Addressing Context Switch Costs with Non-Resumable Delays

Joël Goossens, Université libre de Bruxelles (ULB), Brussels, Belgium,
joel.goossens@ulb.be

Damien Masson, Université Gustave Eiffel (UGE), CNRS, LIGM Paris, France,
damien.masson@esiee.fr

Abstract

In this research, we extend the traditional model of recurrent real-time tasks to incorporate the cost of context switches; we introduce the concept of *Non-Resumable delays* (i.e., loading phases are preemptive, but the processing time already expended in attempting to load a task is forfeited). We propose a model that addresses previous methodology flaws regarding the pessimism and scheduling anomalies, which could be implementable on certain Real-Time Operating System (RTOS) and models high-level abstraction of what is a context switch activity. We provide two exact schedulability tests for two popular scheduling families (Fixed Task Priority as Rate Monotonic and Fixed Job Priority as Earliest Deadline First). These tests are sustainable regarding the execution times and the preemption costs. They are based on the notion of simulation interval and properties of the schedule repetition. We conclude with a discussion of the advantages and drawbacks of our model and the future perspectives it opens up.

Contents

1 Introduction

In this paper, we tackle the challenge of modelling and incorporating the cost of context switches into the feasibility analysis of real-time systems. Our focus is on uniprocessor platforms where traditional periodic hard real-time tasks are executed (Liu and Layland implicit deadline tasks [24] or Leung et al. constrained deadline tasks [23]).

One of the numerous assumptions commonly found in the literature on real-time scheduling theory, even back to its roots in [24, 23], is that preemption costs can be integrated into the worst-case execution time (knows as the WCET) of tasks. The key justification for this assumption is that, if the preemption cost is a constant within the system or, at least, if an upper bound is known, then under common scheduling algorithms like FTP or Earliest Deadline First (EDF) a job (a task release) can only preempt another one *once*. Therefore, if the cost associated with handling the preemption is attributed to the *preempting* task, it is deemed "safe" to include it in the tasks' worst-case execution time. Note that this reasoning is not applicable to all schedulers. For example, this is not the case for Least Laxity First (LLF), where a job can be preempted multiple times by the same job.

Firstly, we observe that this assumption is *pessimistic*, as not every job release in a system corresponds to a preemption. One could argue that even in scenarios where the system is idle upon job release, a Context Switch (CS) remains necessary (at least, the system must prepare the execution context of the job). We will demonstrate our alignment with this perspective in our work, wherein we not only address preemption costs but, more broadly, incorporate *Loading Delays (LDs)*.

We aim to address a second drawback associated with this assumption, namely, preemption is not equivalent to the "regular" execution of a task as we pointed out in [19]. Depending on the considered system, it can be done as an internal routine of the operating system, or handled with hardware interrupts (e.g., on a bare metal embedded system). In the mentioned work, we considered the case where these routines or interrupt handlers are non-preemptive. We exhibited that taking these *non*-preemptive delays into account leads to scheduling anomalies[1] that invalidate classical schedulability analysis results. We tackled the problem by studying a safe bound on a simulation interval for any work-conserving scheduler, which was unfortunately pretty much intractable, and we observed that for the specific case where the resuming delays were equal to one, there was no more scheduling anomalies.

In this paper, we suggest relaxing the non-preemptive assumption slightly and investigate a model where the code for handling CSs is executed in a *Non-Resumable (NR) fashion*. This implies that preempting this code is allowed, but the processing time already expended in attempting to load a task is forfeited. We will see that this is not sufficient to get rid of scheduling anomalies. To do so, we also have to introduce in our model Starting Delays (SDs), i.e., we have to consider that the CS routines are called not only to resume a job, but also when it starts its execution. With such a model, having an *exact*

[1] A scheduling anomaly is a counter-intuitive phenomenon where a locally faster execution leads to an increase in the execution/response time of other activities, potentially resulting in a missed deadline.

schedulability test based on processing demand or on worst-case response times would require counting the preemptions. Instead, in this paper, we demonstrate that the simulation intervals established for FTP and EDF remain valid. Furthermore, we extend these results to encompass a broader class of schedulers.

Finally, we emphasize that the model we propose is well-suited for incorporating more general operations that a modern RTOS might perform. For example, as proposed in [6], security mechanisms can be implemented, such as clearing the caches during a CS (see Section 4 for details).

Paper organisation The organization of this paper is outlined as follows: in this section we introduced our work. Section 2 introduces our theoretical framework. Following that, in Section 3 we present the related results regarding simulation intervals for uniprocessor systems, and models that take into account CSs. Section 4 is devoted to the presentation of our task model. The main results are outlined in Section 5, i.e., the two *exact* schedulability tests and their sustainability. Finally, we conclude in Section 6.

2 Models and notations

We formalise here the task model and the scheduler class considered.

2.1 Task model

In this paper we consider the scheduling of a set τ of n asynchronous constrained deadline periodic tasks upon uniprocessor. More formally, a *periodic task* τ_i is characterized by the tuple $\langle O_i, C_i, D_i, T_i \rangle$. O_i *(the offset)* corresponds to the release time of the *first* job of the task. C_i corresponds to the Worst-Case Execution Time (WCET) of the task. D_i *(the* relative *deadline)* corresponds to the time-delay between a job release and its corresponding deadline i.e. a job released at time t must be completed before or at time $t + D_i$. T_i *(the period)* corresponds to the *exact* duration between two consecutive task releases. The deadlines are *constrained*: $D_i \leq T_i \, \forall i$. The hyper-period, denoted as H, is by definition the least common multiple of the periods: $\text{lcm}\{T_i \mid i = 1, \ldots, n\}$.

The model will be completed in Section 4.1, after our state-of-the-art review and before the presentation of an initial example.

2.2 Scheduler class

In this research, we consider a broad class of schedulers, the CFJP (see Definition 1) which notably includes Rate Monotonic (RM), Deadline Monotonic (DM), and EDF (at least with a CFJP tie-breaker when two jobs have the same deadline).

Let $J_{i,k}$ be the k^{th} job ($k = 1, 2, \ldots$) of task τ_i, which releases at time $O_i + (k-1)T_i$. We also introduce the notation: $J_{i,k} \succ J_{j,\ell}$ which means that the job $J_{i,k}$ has a higher priority than the job $J_{j,\ell}$. Of course, a higher priority will only be effective if the two jobs indeed compete, i.e., if there exists a time instant such that *both* jobs are active. The scheduler is work-conserving and preemptive. Priorities are *fixed* at job level but are also *cyclic* (see Definition 1 for a formal definition). Furthermore, we assume that job priorities are known at design time, derived from static task and job parameters. It is worth noting that, for periodic tasks, the job release time is *static* and known at design time.

Definition 1 (CFJP)**.** *A scheduling rule is said to be Cyclic Fixed Job Priority if* $\forall i, j, k, \ell$*:*

$$J_{i,k+h_i} \succ J_{j,\ell+h_j} \Leftrightarrow J_{i,k} \succ J_{j,\ell}$$

with $h_i = \frac{H}{T_i}$ *and* $h_j = \frac{H}{T_j}$.

For example, EDF belongs to CFJP if, in the event of multiple jobs having the same absolute deadline, the tie is broken by assigning a higher priority to the job with the smaller index. FTP algorithms (for example, RM) also belong to CFJP since different tasks cannot share the same priority. In the remainder of the paper, we assume CFJP schedulers. Also, by a slight abuse of language, jobs $J_{i,k}$ and $J_{i,k+h_i}$ will be referred to as *cousins*.

3 State of the art

In this section, we present the state of the art related to simulation intervals as well as models that take preemption costs into account.

3.1 Simulation intervals

We start with the notion of simulation interval, which served to construct schedulability test consisting on simulating the system. The interest of these intervals is also to provide a *representative range* in simulations, for example, for statistics or real-time scheduling analysis tool as Cheddar [15]. We can also mention the works [1, 2], which are inspired on the schedule repetition properties to provide an exact response-time bounds of periodic DAG.

Definition 2 (Simulation interval [18])**.** *A simulation interval is a finite interval of the form* $[0,X)$ *that includes the periodic part (the cycle) of the schedule.*

Let us note that we assume here that the task durations (WCET) are *constant*; we shall address this matter in the paper when we study sustainability (Section 5.4).

The paper [18] summarises the different simulation intervals of the literature depending on the platform (uni or multi-processor), the deadlines (constrained or arbitrary) and the scheduler type (EDF, RM, fixed priority, dynamic priority).

Table 1: Main results concerning simulation interval

Deadlines	Feature	Scheduler	Interval	Ref.
$D_i \leq T_i$	–	FJP	$[0, O^{\max} + 2H)$	[22]
arbitrary	–	FJP	$[0, O^{\max} + 2H)$	[17]
$D_i \leq T_i$	–	FTP	$[0, S_n + H)$	[16]
$D_i \leq T_i$	mutual exclusion,simple precedence	work-conserving	$[0, \theta_c + H)$	[13]
$D_i \leq T_i$	**Non-Preemptive**	work-conserving	Equation 1 (page 6)	[19]

Table 1[2] summarises the state of the art regarding *uni*processor systems. In the "Feature" column, "–" indicates that the authors consider the classical periodic task model. On line 3, note that S_n is given by the recursive equation $S_1 = O_1, S_i = \max(O_i, O_i + \lceil \frac{\max\{0,(S_{i-1}-O_i)\}}{T_i} \rceil T_i)$. Only the last line mentions a model (Non-Preemptive) that takes into account preemption costs.

In this research, we will demonstrate that two simulation intervals for two scheduling families remain valid in our model with preemption costs. We will see that proving these new properties is far from being incremental. Indeed, we will prove that the results of [22] (line 1 of Table 1), and [16] (line 3) still hold with our model.

[2]Where θ_c represents the last acyclic idle time (see [13] for details).

3.2 Taking preemptions into account

Numerous studies have explored methods for incorporating scheduling delays, as well as other overheads associated with the hardware platform or operating system, into scheduling analyses. Various strategies have been developed for managing this interference. Some involve constraining these delays to include them in schedulability assessments alongside the WCET or as blocking factors, assigning cost overheads to either preempted tasks, preempting tasks, or both. Other approaches suggest alternative task models accompanied by corresponding analyses. Lastly, certain works modify scheduler algorithms to account for the presence of such delays.

It's worth noting that some scheduling delays arise from factors external to the selected scheduling policy, such as overheads imposed by the hardware platform or the operating system. Despite their origin, these delays must be accounted for in schedulability analyses or, more broadly, in evaluating scheduler performance. As highlighted by [19], these delays affect the simulation interval and the periodicity of the schedule.

The initial inquiries into the effects of scheduler implementation and the kernel on scheduling were undertaken in [20]. This study outlines four distinct methodologies for implementing a scheduler. A detailed analysis of the inherent costs associated with each methodology is provided and the widely used schedulability condition for FTP (with constrained deadlines) is extended. Authors incorporate additional costs either in the task execution time term (C_i) or the blocking time term (B_i) of the schedulability condition.

The work in [11] highlights the pessimism surrounding the sufficient conditions established in previous works and provide a more detailed analysis of the overhead associated with a scheduler implemented based on the tick scheduling paradigm.

Other approaches (see, for instance, [5, 8, 10, 32]) consist in constraining the number of preemptions or in computing an upper limit on the potential number of preemptions. This latter approach allows for the incorporation of a maximum blocking time associated with system interference in the schedulability analysis.

Another aspect of disturbance associated with preemptions, as examined in the literature, pertains to the utilization of caches. While caches offer a reduction in response times, they also introduce a notable source of variability in WCETs due to the possibility of tasks encountering additional delays resulting from *cache misses*. The runtime overhead stemming from cache misses induced by premature preemptions is termed Cache-Related Preemption Delay (CRPD) [12, 21, 29, 3, 30, 25]. It has been demonstrated that optimal scheduling considering CRPD is an NP-hard problem [28]. More recently, a feasibility interval has been provided for this problem [31].

In [33, 27], an approach is proposed to integrate the expense of reloading a task after a preemption into the feasibility analysis of a system scheduled with FTP. In the considered model, the preemption duration is the *same* for all tasks; it is a system-level parameter. The preemption involves an *non-resumable* sequence of operations (called "atomic", i.e., which can be preempted, but then the entire sequence must be re-executed later). It is demonstrated that with this model, the critical instant cannot be characterized, the worst-case response time of a task may occur during the transient phase, and the compatibility conditions of the Optimal Priorities Assignment algorithm (OPA) are not met [4]. An optimal Fixed Job Priority (FJP) assignment algorithm is also proposed.

In [9] OSEK kernel overheads (utilizing FTP) are investigated, considering activation, termination, and preemption delays. Activation and termination entail additional durations added to the WCET of a task, while preemption delays are factored in at each activation of a higher priority task. A worst-case response time analysis is presented with a constraint on the number of preemptions, ensuring that the synchronous scenario remains the worst-case scenario albeit with some level of pessimism.

None of these works take into account the fact that executing RTOS routines to implement a context switch is not equivalent to normal execution.

We did so in [19], proposing a model that takes into account preemption and context switches costs, modelled as *non*-preemptive sections with a fixed duration which is a task parameter. That model is subject to scheduling anomalies, particularly exact schedulability tests with this model are *not C*-sustainable, i.e., even if the system is deemed schedulable considering the WCET, an early completion of a job can lead to miss a deadline (formal definition is given in Section 5.4). In addition to not being C-sustainable, another drawback of the test is its reliance on a gigantic simulation interval (see Equation 1 where $L_{\max}$ denotes the length of the largest non-preemptive section). Therefore, this method is not applicable, at least in the general case. It remains valid in particular scenarios, such as harmonic periods.

$$H \cdot (n+1) \cdot (L_{\max}+1) \prod_{i=1}^{n} (\max\{0, O_i + D_i - T_i\} + 1) \tag{1}$$

In this paper, to address these drawbacks, we propose an extension of this model where the delays are tackled as non-resumable sequence of operations, like proposed in [33]. This model is introduced and motivated in the next section.

4 Non-resumable loading and starting delays

The classic way to implement a preemptive scheduler in an operating system is to generate an Interrupt Request (IRQ) periodically or sporadically. The associated Interrupt Service Routine (ISR) will apply the scheduling policy to select the next task, and if it is different from the previous one, it will proceed with the CS: saving the context of the preempted task to load the context of the new task. Various other actions can also be performed here, as saving and restoring caches states or changing security mode for example.

Specific implementations in RTOS can, of course, be optimized to avoid some of the IRQs. It is worth noting that the ISR processing time depends on whether there is a preemption (task switch) or not.

Incorporating these ISR processing times into the tasks' WCETs is the approach of the classic state-of-the-art real-time scheduling analysis. It is a valid approach provided these times are negligible compared to the WCETs. This applies when considering CS in the raw sense, i.e., saving and loading new values for state word registers. However, if we relax this assumption, it presents two drawbacks.

The first is pessimism: the ISR processing time in the event of an actual preemption (with CS) must be added to the WCET of all tasks, even though not all job released will necessarily result in an actual preemption. We will illustrate this with an example in Section 4.2.

The second comes from the fact that ISR processing is generally done in non-preemptive mode. As analysed in our previous works [19], unless in the case where this processing cost is equal to 0 or 1 (time is discrete), this has several disastrous consequences: classical schedulability analysis, such as response time analysis, is not valid because scheduling anomalies are possible. We then proposed a bound on the necessary simulation interval to test the schedulability of the system (see Equation 1). However, this bound is terribly large, making its practical use almost impossible, except in very specific cases. Moreover, the bound is no longer valid if we consider systems where tasks can complete their executions before their WCET.

In this work, we propose modelling preemption processing not as non-preemptive code blocks, but as non-resumable code blocks: they can be interrupted, but must be entirely restarted later. Although we are unaware of any RTOS where this is implemented in this manner, it is possible to do so in practice. This requires a system with multiple levels of IRQ priorities (such as NucleusRTOS [14]). Assuming that CS processing is implemented in non-resumable ISRs, we then propose a model 1) for which we demonstrate reasonable and practically usable simulation intervals, and 2) for which these intervals remain valid even if tasks execute for less time than their WCET.

To demonstrate these properties, we need to consider a non-resumable code block during each CS, including when starting a new job: we distinguish the initial job loading time from the resume time. The

constraint on these times is that the initial loading time must at least be equal to the resume time, but it can be longer.

This model is noteworthy from a theoretical perspective as it offers a practical solution to the problem of incorporating preemption times into schedulability analysis, avoiding pessimism and scheduling anomalies. Furthermore, it adapts well to modelling higher-level preemption processing, for instance, if we wish to add security mechanisms, such as erasing a task's traces (in caches, for example), as proposed in [6]. In this work, the authors suggest inserting a non-preemptive code segment each time a *critical* task must yield the processor, to prevent an *attacking* task from retrieving information about the critical task by *clearing* the memory. An equivalent approach would be to systematically add the cleaning operations at the beginning or resumption of each *non-secure* (and thus potentially attacking) task. These code sections could then very logically be NR rather than Non-Preemptive (NP), and the system could thus be analysed with our methodology.

4.1 Task model with NRLD

We first introduce the term Non-Resumable (NR) operation (Definition 3), followed by the model.

Definition 3 (Non-Resumable)**.** *A Non-Resumable (NR) operation is a type of computing operation that* can be interrupted*. However, if interrupted, the system is required to restart the operation from its initial state.*

In the following, we distinguish between two kinds of Loading Delays (LDs): Starting Delays (SDs) and Resuming Delays (RDs).

Definition 4 (Starting Delay)**.** *The Starting Delay (SD) represents the required time to load a job for the* first *time when it becomes ready for execution.*

Definition 5 (Resuming Delay)**.** *The Resuming Delay (RD) represents the required time to reload a job after it has been preempted by another job.*

Definition 6 (Loading Delay)**.** *Loading Delay (LD) is a general term to designate either Starting Delay, Resuming Delay or both.*

We consider *non*-resumable starting and resuming delays upon uniprocessor. We complement here the model presented in Section 2 by the addition of two further parameters. Each task τ_i is characterised by the tuple $\langle O_i, C_i, D_i, T_i, \mathrm{SD}_i, \mathrm{RD}_i \rangle$, where $\mathrm{SD}_i \geq 0$ designates the *worst-case* duration of the SD of task τ_i, and where where $\mathrm{RD}_i \geq 0$ designates the *worst-case* duration of the Resuming Delay (RD) of task τ_i. We consider that the system time is *discrete*. Without loss of generality, the parameters of the tasks are therefore integers. In this work we assume that $\mathrm{SD}_i \geq \mathrm{RD}_i \ \forall i$.

The rationale for considering SD to be at least equal to RD is crucial, as this property is essential for exact schedulability tests to be *sustainable* with respect to execution times. The formal definition and proof of this property are provided in Section 5.4. Notably, a counterexample demonstrating the consequences of neglecting SD is presented in Figure 5 on page 15.

The introduction of Non-Resumable Loading Delays (NRLDs) resolves the priority inversion issue inherent in the model with Non-Preemptive Loading Delays (NPLDs).

We will also use the notation $e_{i,t}$ for the amount of time for which the last job of task τ_i released *strictly* before t *has executed* at time t. Note that the quantity $e_{i,t}$ does *not* take loading times into account, hence we have $0 \leq e_{i,t} \leq C_i$. We assume that $\forall i \ e_{i,t} = C_i$ for $t \leq O_i$.

In order to define tests that are both exact and robust (without scheduling anomalies), we will proceed in two steps. First, we will assume that the durations C_i, SD_i, and RD_i are *constants* (these are in fact bounds on the actual durations). Under this assumption, we will define exact tests based on the repetition of the schedule (simulation interval). Second, we will show that our analysis is robust with respect to

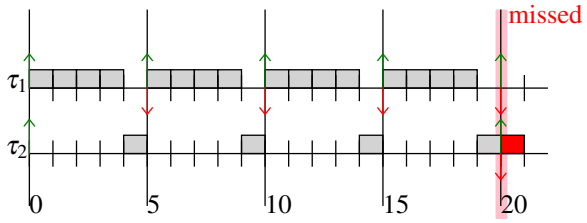


Figure 1: System modelled to account classical analysis theory is not schedulable (load greater than 1 and deadline missed at 20).

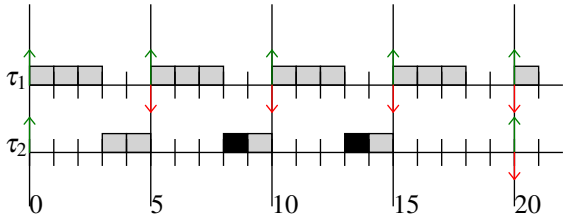


Figure 2: System is schedulable accounting only real preemption costs (cycle reached at time 20).

these durations. This means that if the system passes the test (is schedulable with the constants), it remains schedulable even if the actual durations are smaller than the bounds. In particular, during the system execution, an SD can be smaller or even zero compared to the corresponding RD. Thus, we assume until Section 5.4 that the durations C_i, SD_i, and RD_i are *constants*.

4.2 Motivating example

We consider a real-time system composed of two tasks, τ_1 and τ_2, scheduled with EDF. Task τ_1 has a period and a deadline of 5 time units and requires 2 execution time units. Task τ_2 has a period and a deadline of 20 time units and requires 3 execution time units. The CS cost when loading or resuming a preempted task on the platform is 1 time unit.

Classical methodology incorporates the cost of CSs into the WCETs of the tasks. We will have $C_1 = 4$, because τ_1 requires 2 time units to execute, 1 time unit to load, and 1 time unit to handle the preemption cost in the case where a job preempts another one (this cost is accounted to the preempting task rather than the preempted one to easily limit it to one per job). We will have $C_2 = 5$, because τ_2 requires 3 time units to execute, 1 time unit to load, and 1 time unit to handle a possible preemption cost.

As shown in Figure 1, the resulting system is *not* schedulable. This is due to an overestimation of the number of preemptions.

All the scheduling figures in this paper are generated by the open-source tool *Draw Schedule*, which can be downloaded or used online directly [26]. Upward arrows represent activations, downward arrows represent deadlines. Light squares indicate executions, and dark squares represent loading times.

In our previous work [19], we proposed modelling the system preemption costs as non-preemptive sections attributed to the preempted task, thereby modelling the resuming operation. Applied to our example, this gives us the values $C_1 = 3$ (1 for loading, 2 for executing) and $C_2 = 4$, with an additional non-preemptive execution block of 1 time unit before each resuming. As shown in Figure 2, this approach is less pessimistic and the resulting system is now schedulable. However, the analysis associated to this model does not scale, and the feasibility relies on an online respect of all system parameters: the model is not sustainable regarding WCETs and LDs.

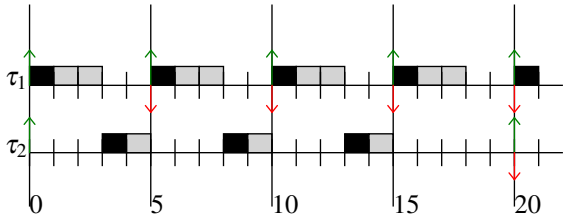


Figure 3: System modelled with SDs and RDs is schedulable (cycle reached at time 20).

In this paper, we propose to consider these additional execution blocks as NR (see Definition 3) and to go further by also modelling SD (see Definition 4) as additional execution blocks separated from the regular execution accounted in the WCET. Applied to our example, this gives us the values $C_1 = 2$ and $C_2 = 3$ (execution time only), with a loading time and a resuming time of 1 for all tasks. Figure 3 illustrates the resulting schedule. The feasibility property is now sustainable regarding WCETs and LDs, i.e. if these values are lesser than considered during the schedulability analysis process when the system is executed, the system is guaranteed to remain feasible. However, the model relies on the capability for the RTOS to implement the CSs in a NR fashion.

5 Properties

In this section, we present the important properties of our model, focusing in particular on the two schedulability tests. We begin by demonstrating that, concerning response times, our model is strictly superior to the one in [19]. We then study the general CFJP class, which includes EDF as a popular representative. Finally, we examine the specific case of the FTP class, which includes RM. Afterward, we investigate the sustainability of the tests concerning execution and loading delays.

5.1 Response times

Considering NRLD model, we prove that response times are always lower or equal than those with the same scheduler considering NPLD model.

Theorem 1 (Response time)**.** *Let $J = \{J_1, J_2, J_3, \ldots\}$ be a set of jobs, let S_1 and S_2 be two schedules of J (for the* same *work-conserving FJP assignment) for NPLD and NRLD, respectively. Let $e^S_{i,t}$ be the cumulative amount of time for which job J_i has executed (loading time excluded) at time t in the schedule*[3] *S. Then for each job J_i and for each instant t (not smaller than the release time of J_i):*

$$e^{S_1}_{i,t} \leq e^{S_2}_{i,t} \ .$$

Proof. By contradiction. Let $t_1 \geq 0$ be the *first* time instant such that:

$$\exists\, i \text{ s.t. } e^{S_1}_{i,t_1+1} > e^{S_2}_{i,t_1+1} \ . \tag{2}$$

Note that trivially, we have $e^{S_1}_{i,0} = e^{S_2}_{i,0} = 0$. Consequently, we know that $e^{S_1}_{i,t_1} = e^{S_2}_{i,t_1}$ and $\forall j \neq i, t \leq t_1$ $e^{S_1}_{j,t} \leq e^{S_2}_{j,t}$. The situation means that at time instant t_1 the job J_i is executed in S_1 but not in S_2.

Let's first observe that:

1. i is *unique* since we consider uniprocessor.

[3] $e^S_{i,t}$, which pertains to a set of jobs, should not be confused with $e_{i,t}$, which is defined for a set of periodic tasks.

2. The situation cannot be associated with the *release* of a higher priority job in S_2 at time t_1 since this would also be the case in S_1 at time t_1 (S_1 and S_2 schedule the same set of jobs with the same work-conserving FJP scheduler, the only difference being the Non-Resumable/Non-Preemptive behaviour during load sections).

Also note that if the job J_i is completed at time t_1 in S_2 we have $e_{i,t_1}^{S_2} = e_{i,t_1+1}^{S_2} = C_i$ (the duration of the job J_i) consequently $e_{i,t_1+1}^{S_1} > C_i$ which is impossible. Consequently, in the following we will assume that job J_i is *not* completed at time t_1 in S_2. We will distinguish two cases: at time $t_1 - 1$ the job J_i is executed in S_1 or not.

Case 1. If at time $t_1 - 1$ job J_i is executed in S_1 then job J_i is executed in *both* schedules at time $t_1 - 1$ (this is a consequence of t_1 being *first* time satisfying Equation 2). Consequently, in both schedules J_i is executed at time t_1 since in both schedules we have the same active job with the highest priority (J_i) and the same remaining execution time at time t_1. In this situation we have to schedule J_i at time t_1 in S_2 as well, which leads to a contradiction.

Case 2. Job J_i is executed at time t_1, but not at time $t_1 - 1$ in S_1. We will show that this situation also leads to a contradiction. First, we consider the situation where J_i is released at time t_1. In this scenario, we know that $\mathrm{SD}_i = 0$, and J_i must also be executed at time t_1 in S_2, which leads to a contradiction. Now, let's assume that J_i is released before t_1. If we consider the schedule S_1, as job J_i runs for the first time at time t_1 or was preempted in the past, the execution is preceded by a block of loading slot(s) of duration λ_i (corresponding to its job/task parameter, here we know that $\lambda_i > 0$ — i.e., $\lambda_i = \mathrm{SD}_i > 0$ if J_i runs for the first time, $\lambda_i = \mathrm{RD}_i > 0$ if J_i was preempted before t_1 — otherwise J_i must be executed at time $t_1 - 1$)[4]. Moreover, we know that at *both* $t_1 - \lambda_i$ and t_1, job J_i has the highest priority. Consequently, there are no new releases of higher priority jobs in S_1 in the interval $[t_1 - \lambda_i, t_1]$.

Note that *if* in S_2 the job J_i has the highest priority at time $t_1 - \lambda_i$, there are also no new releases of jobs with higher priority within the interval $[t_1 - \lambda_i, t_1]$. Consequently, J_i *must* also be executed at time t_1, which leads to a contradiction. Thus, the only case in which Equation 2 can be verified is the case where, in S_2 at time $t_1 - \lambda_i$, the job J_i is *not* the one with the highest priority ready for execution. Consequently, at time $t_1 - \lambda_i$ there is another job with higher priority, say J_k, which is completed in S_1 but active (not completed) in S_2, which means that:

$$\exists t_0 < t_1 - \lambda_i : e_{k,t_0+1}^{S_1} > e_{k,t_0+1}^{S_2}, \tag{3}$$

which contradicts the fact that t_1 is the *first* such instant (Equation 2) and proves the property. □

5.2 Simulation interval for CFJP

We will generalise the properties of Leung & Merrill [22] and show that $[0, O^{\max} + 2 \times H)$ is a simulation interval for CFJP (which includes EDF) in our framework. The first property to generalise concerns the progress of cousin jobs. In particular, the oldest cousin makes at least as much progress than its cousin one hyper-period later:

Lemma 1. *Let S be the schedule of an asynchronous periodic task system $\tau_1, \ldots, \tau_n$ constructed by a Cyclic Fixed Job Priority scheduler considering* NRLDs*. Then for each task τ_i and for each time instant $t \geq O_i$, we have $e_{i,t} \geq e_{i,t+H}$.*

Proof. We will show the property by contradiction. Let $t \geq O_i$ be the *first* time instant such that:

$$\exists\, i \text{ s.t. } e_{i,t} < e_{i,t+H}\ . \tag{4}$$

[4]It is at this point that we use the assumption that there are starting delays, since otherwise we can easily identify a counterexample of the property we are proving.

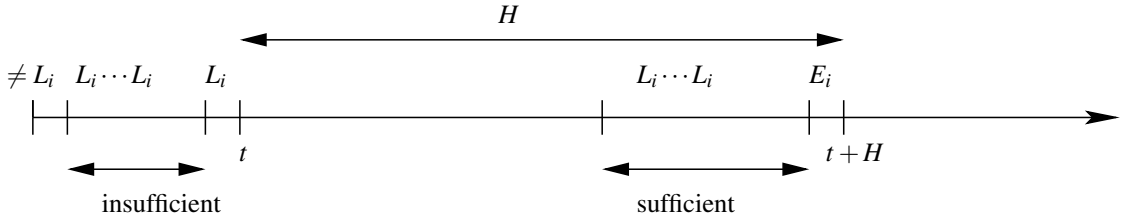


Figure 4: Case d.5, proof of Lemma 1

Let's first observe that i is *unique* since we consider uniprocessor scheduling. Equation 4 implies that the task τ_i is active at both instants $t-1$ and $t+H-1$, and τ_i is executing at time instant $t+H-1$ but not at time instant $t-1$. We will consider all cases where this can happen with NRLDs. For each case (*a–d.5*) we will show that the situation is not possible and leads to a contradiction. But first we introduce an additional notations: $S(t)$ denotes the schedule decision at time t, $S(t)=E_i$ meaning the execution of τ_i, $S(t)=L_i$ meaning the loading of τ_i (starting or resuming), and $S(t)=I$ meaning that the processor is idle. We extend the notation for the time interval $[t_a,t_b]$: $S(t_a,t_b)=[S(t_a),S(t_a+1),\ldots,S(t_b-1),S(t_b)]$.

Case a. $S(t-1)=E_j$, $j\neq i$ (and $S(t+H-1)=E_i$), in this case task τ_j, has an higher priority job, active at time instant $t-1$ but not at time instant $t+H-1$. I.e., one hyper-period later, the cousin job of τ_j is not active. Keep in mind that we are considering a CFJP schedulers, and therefore, the relative priorities of jobs remain the same one hyper-period later. The situation means that $e_{j,t-1}<e_{j,t+H-1}=C_j$, which contradicts the fact that t is the *first* instant which satisfies Equation 4.

Case b. $S(t-1)=I$ (and $S(t+H-1)=E_i$), this is impossible since τ_i is active (not completed) at time instant $t-1$ and the scheduler is work-conserving.

Case c. $S(t-1)=L_j, j\neq i$ (and $S(t+H-1)=E_i$), task τ_j has a higher priority job, active at time instant $t-1$ but not at time instant $t+H-1$. This scenario again contradicts the fact that t is the *first* instant which satisfies Equation 4.

Case d. $S(t-1)=L_i$ (and $S(t+H-1)=E_i$). This case will require a bit more attention, especially *Case d.5*. First notice that since $S(t-1)=L_i$ we know that $\mathrm{SD}_i>0$. Also note that $t-1$ cannot correspond to the release of τ_i, otherwise it should also be released at $t+H-1$ which is not possible since we have $S(t+H-1)=E_i$. In other words, τ_i is released strictly before $t-1$, meaning $t-2\geq 0$. We have five sub-cases to investigate:

Case d.1. $S(t-2)=E_i$. The sequence $S(t-2,t-1)=[E_i,L_i]$ is impossible since it is impossible at time $t-1$ (and $t+H-1$) for a new release of τ_i to occur since we have $S(t+H-1)=E_i$ and a starting delay is required necessarily in our model (since $\mathrm{SD}_i>0$).

Case d.2. $S(t-2)=E_j, j\neq i$. Task τ_j has a higher priority job active at time $t-2$ but completed at time $t+H-2$. This again contradicts the fact that t is the *first* instant which satisfies Equation 4.

Case d.3. $S(t-2)=I$ this impossible for a work-conserving scheduler since at time $t-2$ there is at least one active task (τ_i). Let's recall that it is impossible for a new release of τ_i to occur at $t-1$ since we have $S(t+H-1)=E_i$ and a loading delay is required.

Case d.4. $S(t-2)=L_j$, $j\neq i$, the sequence $S(t-2,t-1)=[L_j,L_i]$ is impossible, since the sequence means that a new release of τ_i (a higher priority job) occurs at time instant $t-1$ and consequently also one hyper-period later at time instant $t+H-1$ but we have $S(t+H-1)=E_i$ which is not allowed since at time instant $t+H-1$ a starting delay is required (since $\mathrm{SD}_i>0$ and $\mathrm{SD}_j>0$).

Case d.5. $S(t-2)=L_i$. Let's begin by noting that $e_{i,t-1}=e_{i,t-1+H}$ because t is the *first* time instant which satisfies Equation 4. Also note that for the same reason, it is not possible to have $S(t+H-2)=E_i$. So case *Case d.5.* corresponds to Figure 4. Firstly, if $e_{i,t-1+H}=0$ (and thus $e_{i,t-1}=0$), it is the *first* execution of a job of τ_i. That is, we have a *starting* block (of size $\mathrm{SD}_i>0$) *immediately* followed by the execution of τ_i, the schedule decision $S(t-1+H)=E_i$. Since $S(t-1)=L_i$, we must have p s.t.

$1 \leq p < \text{SD}_i$ and $S(t-1-p, t-2) = [L_i, \ldots, L_i]$ and $S(t-2-p) \neq L_i$, otherwise at time instant $t-1$ task τ_i *must* be executed (since the starting block is completed). The only possibility is to have either $S(t-2-p) = E_j$ or $S(t-2-p) = L_j$ with $i \neq j$ (the cpu cannot be idle). In both cases the situation means that a higher priority job is active (and not completed) at time $t-2-p$ while one hyper-period later, at time $t+H-2-p$, the cousin job is completed, this contradicts again that t is the first time instant which satisfies Equation 4. Secondly, if $e_{i,t-1+H} > 0$ (and thus $e_{i,t-1} > 0$), for both the cousin jobs of τ_i at time $t-1$ (and $t-1+H$) we need a block of size $\text{RD}_i > 1$ before executing the job at time $t-1$ (and $t-1+H$) but such a block is incomplete right before $t-1$. This is similar to the first case ($e_{i,t-1+H} = 0$): we encounter the same contradiction once again.

We have thoroughly covered all the cases, first for the slot $t-1$, with the possibilities being E_i, E_j, I, L_j, L_i. All situations are contradictory: E_i (impossible by assumption), E_j (case a., contradiction of Eq. 4), I (case b., contradiction of work-conserving), L_j (case c., contradiction of Eq. 4). This is independent of what happens in $t-2$. The last case L_i must be broken down into 5 sub-cases, based on the possibilities in slot $t-2$: E_i, E_j, I, L_j, L_i. Again, all 5 cases are contradictory: E_i (case d.1 requires a release of τ_i at $t-1$), E_j (case d.2, contradiction of Eq. 4), I (case d.3, contradiction of work-conserving), L_j (case d.4, requires a release of τ_i at $t-1$), and finally L_i (case d.5, as illustrated by Figure 4).

□

The second property concerns the periodicity of CFJP schedulable systems. We begin by formally defining the notion of *idle point*.

Definition 7 (Idle point)**.** *We say that $x \in \mathbb{N}$ is an* idle point *if all the jobs released strictly before x have completed their execution before or at time x.*

Unlike the concept of an idle *slot*, where the CPU is idle for a unit of time, the idle *point* means that the CPU has completed the pending work.

Lemma 2. *Let S be the schedule of a task system R constructed by a Cyclic Fixed Job Priority scheduler with Non-Resumable Loading Delay. If R is CFJP-schedulable, then $\forall i$ and $\forall t_1 \geq O^{\max} + H$ we have $e_{i,t_1} = e_{i,t_2}$, where $t_2 = t_1 + H$, and $O^{\max} = \max_{i=1}^{n} O_i$.*

Proof. We will show the property by contradiction. We assume, for the purpose of the contradiction, that $\exists\, \ell$ and $t_1 \geq O^{\max} + H$ s.t. $e_{\ell,t_1} \neq e_{\ell,t_2}$. By Lemma 1, we must have

$$e_{\ell,t_1} > e_{\ell,t_2} \tag{5}$$

We will first show that during the time interval $[t_1, t_2)$ there is *no* idle slot (Subproof A below). Then we will prove that the number of loading slots in the interval $[t_2, t_3)$ (one hyper-period later, where $t_3 = t_2 + H$) is greater than or equal to the number of loading slots in the interval $[t_1, t_2)$ (see Subproof B below). Finally, (after the two sub-proofs) we combine Eq. 5 and the two sub-properties to obtain a contradiction.

Subproof A. We will prove that the interval $[t_1, t_2)$ does not contain any idle slot. We will prove this by contradiction. For the purpose of the contradiction we will assume an idle slot during the time-slot $[t_1 + \Delta, t_1 + \Delta + 1)$ (with $\Delta < H$). This implies that all task computations requested prior to $t_1 + \Delta$ have finished their execution. Consequently, $t_1 + \Delta$ is an idle point and $\forall i, e_{i,t_1+\Delta} = C_i$. By definition, we know that for any t, $e_{i,t} \leq C_i$; and thus, in particular, for $t = t_1 - H + \Delta$, i.e., $\forall i, e_{i,t_1-H+\Delta} \leq C_i$, by Lemma 1 we have $\forall i, e_{i,t_1-H+\Delta} \geq e_{i,t_1+\Delta}$. We just proved that $\forall i, e_{i,t_1+\Delta} = C_i$. Consequently, $\forall i, e_{i,t_1-H+\Delta} = C_i$. In other words, all task computations requested prior to $t_1 - H + \Delta$ have finished their execution, as well. I.e., $t_1 - H + \Delta$ is an idle point as well. Since the task requests in the interval $[t_1 - H + \Delta, t_1 + \Delta)$ and $[t_1 + \Delta, t_2 + \Delta)$ are the same, since $\forall i, e_{i,t_1-H+\Delta} = e_{i,t_1+\Delta} = C_i$, and since we consider CFJP schedulers, the schedule in the interval $[t_1 - H + \Delta, t_1 + \Delta)$ and $[t_1 + \Delta, t_2 + \Delta)$ must be *identical*.

Since $t_1 \in [t_1 - H + \Delta, t_1 + \Delta)$, $t_2 \in [t_1 + \Delta, t_2 + \Delta)$, and $t_2 - t_1 = H$; we conclude that $\forall i, e_{i,t_1} = e_{i,t_2}$ contradicting our initial assumption (Eq. 5). We have just proved that if we assume that $\exists\, \ell$ s.t. $e_{\ell,t_1} > e_{\ell,t_2}$, there is no idle slot in the time interval $[t_1, t_2)$. △

Subproof B. We will prove that the amount of loading slots in the interval $[t_2, t_3)$ is greater or equal to the one in the interval $[t_1, t_2)$. The only factor that changes the amount of loading necessary to serve a job J_i is the number of times it is preempted: each preemption adds a loading time (RD_i). We will prove the property by contradiction. For the purpose of the contradiction we will assume that the amount of loading slots in the interval $[t_1, t_2)$ is *strictly* greater than the one in the interval $[t_2, t_3)$, it implies that there exists at least one job J_r of task τ_r which is preempted by a job with a greater priority released at time $t_p < t_2$ and a cousin job J'_r of the same task released on hyper-period later which is *not* preempted at time $t_p + H$, which is also the release of a job with a greater priority. There are two cases:

1. J'_r is completed at time $t_p + H$, i.e., $e_{r,t_p+H} = C_r$ but this is in contradiction with Lemma 1 since $e_{r,t_p} < C_r$ (J_r is preempted, not completed).
2. J'_r is not executing (or reloading/starting) at time $t_p + H - 1$. It implies that it exists a job J'_k with a higher priority than J'_r executing (or reloading/starting), since J'_r is pending and not executing. It follows that there must exist a cousin job of τ_k, the job J_k of priority higher than J_r completed at time $t_p - 1$ (since J_r is executing or reloading at time $t_p - 1$). To explain that J'_k is not completed at time $t_p + H - 1$ but J_k is at time $t_p - 1$, there must exists $\delta > 1$ such that a job with a higher priority is executing at time $t_p + H - \delta$ but a cousin job released one hyper-period before completed at time $t_p - \delta$. Repeating the argument, it exists another $\delta_2 > \delta$ with the same property, again and again, until $\delta_h > t_p$, which is impossible since no job can be active before time 0.

△

Now, we combine Eq. 5 and the two sub-properties to show that the system is overloaded and consequently not schedulable which is a contradiction and prove Lemma 2. Indeed, if there is no idle slot in $[t_1, t_2)$ and if $e_{\ell,t_1} > e_{\ell,t_2}$ we can make three observations: *(i)* for all jobs released in $[t_1, t_2)$, there is a cousin job released one hyper-period later in $[t_2, t_3)$; *(ii)* the amount of work to do in the interval $[t_1, t_2)$, is composed of the backlog at t_1 plus the necessary work to serve among the jobs released in the interval; that amount of work (including the necessary reloading slots) is at least equal to H (since $[t_1, t_2)$ does not contain any idle slot see subproof A); *(iii)* the amount of work to execute in $[t_2, t_3)$ is composed of the backlog at t_2, which is greater than the one at t_1 (Eq. 5) plus the necessary work to serve among the jobs released in the interval. This is composed of the executing part, which is the same as in $[t_1, t_2)$ plus the reloading slots (which is at least as large as that of the interval $[t_1, t_2)$, see subproof B). This implies that the amount of work to serve in $[t_2, t_3)$ is *strictly* greater than the one in $[t_1, t_2)$. Applying the same arguments for the subsequent intervals of length H, we conclude that the amount of work to serve is *strictly* increasing and so that the system cannot be schedulable, contradicting our hypothesis and proves Lemma 2. □

We now have the material to prove that $[0, O^{\max} + 2 \times H)$ is a simulation interval in our framework.

Theorem 2 (Simulation interval for CFJP)**.** *Any Cyclic Fixed Job Priority-schedulable asynchronous constrained-deadline periodic tasks with Non-Resumable Loading Delays upon uniprocessor platform reaches a cycle at or prior to:* $O^{\max} + 2H$.

Proof. This is a direct consequence of Lemma 2: if no deadline are missed in $[0, O^{\max} + 2H)$, the system is in the same state at time $O^{\max} + H$ and $O^{\max} + 2H$ and the schedule repeats. □

5.3 Simulation interval for FTP schedulers

In this section we consider a particular case of CFJP schedulers, the class FTP, where the priority are fixed at *task* level as RM. We will generalise the property of Goossens et al. [16] in our framework. Let's consider the values S_n defined as follows:

Definition 8 (S_n [16]). $S_1 = O_1$, $S_i = O_i + \left\lceil \frac{\max\{0,(S_{i-1}-O_i)\}}{T_i} \right\rceil T_i$ $(i > 1)$.

Informally speaking, S_i corresponds to the release time of the *first* job of τ_i at or after time S_{i-1}.

For FTP schedulers $[0, S_n + H)$ is a simulation interval with NRLD delays with and without SD. Indeed, since there is no priority inversion (see Definition 9) in our model the inductive proof can be extended easily in our framework.

Definition 9 (Priority inversion). *There is a* priority inversion *whenever a* lower-*priority task/job is executing (or loading) while a* higher-*priority task/job is blocked and unable to proceed until the lower-priority one completes.*

Theorem 3 (Simulation interval for FTP). *Any FTP-schedulable ($\tau_1 \succ \tau_2 \succ \cdots \succ \tau_n$) asynchronous constrained deadline periodic tasks with Non-Resumable Loading Delay delays with* and *without Starting Delay upon uniprocessor platform reaches a cycle at or prior to: $S_n + H_n$ where $H_n = \text{lcm}\{T_j \mid j = 1, \ldots, n\}$.*

Proof. Since the proof is practically identical to that published in [16], we present here a condensed version and invite the reader to consult [16] for details.

By induction.

Base case: When $n = 1$, if feasible the schedule for τ_1 reaches a cycle at $T_1 = H_1$ from the first release of τ_1 ($S_1 = O_1$).

Inductive hypothesis: Assume that the theorem holds for some arbitrary value $n = k$.

Inductive step: We will show that the theorem also holds for $n = k + 1$. We know that if the task set $\{\tau_1, \tau_2, \ldots, \tau_k\}$ is FTP-schedulable then the schedule of $\{\tau_1, \ldots, \tau_k\}$ reaches a cycle is at S_k with a period of H_k. As there is no priority inversion caused by task τ_{k+1}, the schedule and periodicity of tasks $\tau_1, \ldots, \tau_k$ remain unaffected by the releases made by task τ_{k+1}. Let S_{k+1} denote the first release of task τ_{k+1} after (or at) S_k (i.e., $S_{k+1} \geq S_k$). Combining the periodicity of the schedule of $\tau_1, \ldots, \tau_k$ and the periodicity of τ_{k+1} (from S_{k+1} with a period of T_{k+1}), we can conclude that the theorem is satisfied for $n = k + 1$. That is, if $\tau_1, \ldots, \tau_{k+1}$ is schedulable, then its FTP schedule reaches a cycle at S_{k+1} with a period of $\text{lcm}\{H_k, T_{k+1}\} = \text{lcm}\{T_j \mid j = 1, \ldots, k+1\} = H_{k+1}$. □

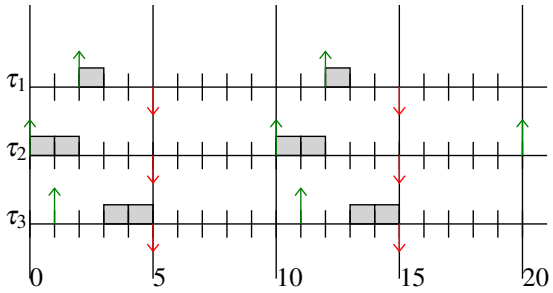
5.4 Sustainability

In this section, we will use the definition of *sustainability* as introduced in [7].

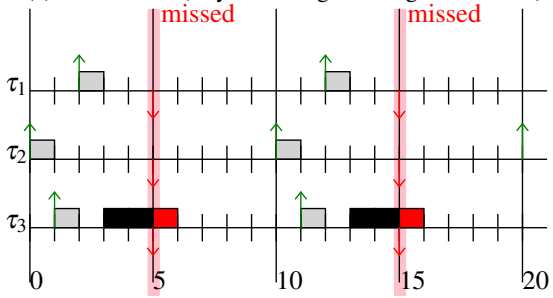
Definition 10 (Sustainable [7]). *A schedulability test for a scheduling policy is sustainable if any system deemed schedulable by the schedulability test remains schedulable when the parameters of one or more individual job[s] are changed in any, some, or all of the following ways: (i) decreased execution requirements; (ii) later arrival times; (iii) smaller jitter; and (iv) larger relative deadlines.*

Considering the NRLD model, we study here the sustainability of the test consisting of simulating system behaviour over the simulation intervals proven to be correct in Sections 5.2 and 5.3, for CFJP and in the special case of FTP, respectively.

More specifically, we are interested in the sustainability of this test regarding the WCET (C_i), the duration of LD (SD_i and RD_i) and the job arrival times (by abuse of language, T-sustainable in the following). We will refer to these properties with the terms C-sustainability, LD-sustainability, and T-sustainability, respectively.

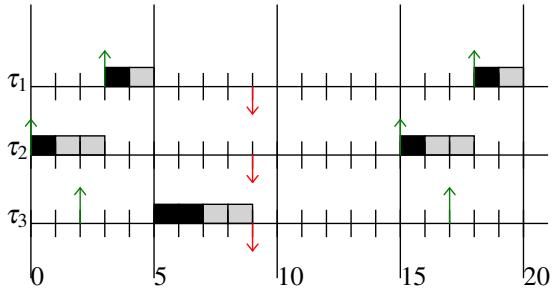


(a) Schedulable (a cycle of length 10 begins at time 2)

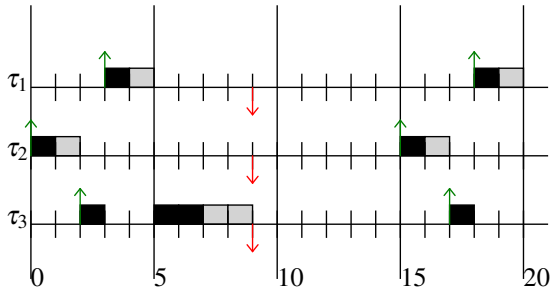


(b) Not schedulable if τ_2 terminates earlier (deadline miss at time 5)

Figure 5: **– No *C*-sustainability without SDs.** Sub-figure 5a shows the schedule obtained with EDF of a system composed of task $\tau_1 = \langle O_1 = 2, C_1 = 1, T_1 = 10, D_1 = 3, \mathrm{RD}_1 = 2\rangle$, $\tau_2 = \langle 0, 2, 10, 5, 2\rangle$ and $\tau_3 = \langle 1, 2, 10, 4, 2\rangle$. Sub-figure 5b shows the schedule of the same system when jobs of τ_2 terminates after having executing only for 1 time unit, causing the following jobs of τ_3 to miss their deadline. The reason for this is that the time slots allocated to τ_3 at time 3 (and 13) are normal executions in the first case and RDs in the latter one.



(a) Schedulable (a cycle of length 15 begins at time 3)



(b) Remains schedulable if τ_2 terminates earlier

Figure 6: **– *C*-sustainability with SDs.** This figure depicts a situation similar to the one in Figure 5, but when SDs are considered. Sub-figure 6a presents the schedule of the system composed by $\tau_1 = \langle O_1 = 3, C_1 = 1, T_1 = 15, D_1 = 6, \mathrm{SD}_1 = 1, \mathrm{RD}_1 = 1 \rangle$, $\tau_2 = \langle 0, 2, 15, 9, 1, 1 \rangle$ and $\tau_3 = \langle 2, 2, 15, 7, 2, 2 \rangle$. Sub-figure 6b shows the schedule of the same system when jobs of τ_2 terminates after having executing only for 1 time unit. This does not have any effect on τ_3's jobs response times. The reason for this is that time slots allocated to load τ_3 at time 5 are necessary, as an SD in the first case, as an RD in the latter one. Furthermore, the additional LD at time 3 cannot exceed the time gained by the early completion of τ_2.

First, one can note that without considering SDs, exact tests are *not* *C*-sustainable as shown by counterexample depicted by Figure 5. We can easily construct a similar example where a task terminates earlier because it was resumed *sooner* than expected, so exact tests when considering only RDs are not RD-sustainable neither. We will see that exact tests when considering *both* type of delays are however *C*-sustainable (Section 5.4.1), SD-sustainable, and RD-sustainable (Section 5.4.2). Finally, exact tests are not *T*-sustainable with any of the considered models (Section 5.4.3).

5.4.1 Sustainability regarding Worst-Case Execution Time

We prove in this section the *C*-sustainability property by proving that the completion times of a collection of jobs can only decrease when some jobs complete having executed for less than their WCET. This is formalised by Theorem 4. Figure 6 presents an example extended from the counterexample when SDs are not considered (Figure 5) to provide an intuitive understanding of what occurs.

The *C*-sustainability is a particularly important property since our main results in this work consist on bounding the simulation interval. If simulating the system over the interval was not a *C*-sustainable test, considering WCET would not be a conservative approach (the worst-case scenario would not be covered).

Theorem 4. *Considering NRLD model and CFJP scheduler, if any job terminates being executed for less than its WCET, it cannot results in any increase of any job response time.*

Proof. Consider the schedule S where all jobs have a worst-case duration (C_i for each job of τ_i). Consider S' as the schedule for the same set of tasks where all jobs, except for $J_{i,k}$, have a worst-case duration, while the actual execution time of $J_{i,k}$ is $C_i - w$ ($w > 0$). Until the completion time of $J_{i,k}$ in S', S and S' are identical. After the completion of $J_{i,k}$ in S', S consists of time slots allocated to other jobs, slots allocated to load $J_{i,k}$ (whether successfully or not), and slots allocated to execute $J_{i,k}$. All these time slots previously allocated to $J_{i,k}$ in S are available for other jobs in S'. The time slots previously allocated to loading/executing other jobs in S remain available for the same jobs in S' until they are completed. This is due to the use of a fixed job priorities (CFJP) scheduler and the absence of non-preemptive operations in our model, ensuring no priority inversions occur. As a result, these jobs can complete earlier or at the same time in S' as in S.

We must prove that allocating more time slots for executing or loading a job does not increase its response time. We start by proving that allocating one additional time slot does not increase the response time. The generalisation is straightforward, as the argument can be applied repeatedly. Let us consider a given job $J_{\ell,m}$ and compare its response time in S and in S', noting that in S' there is an additional slot for it.

Let's start by noting that an execution slot of $J_{\ell,m}$ in S cannot become a loading slot in S'.

Subproof. Suppose, for contradiction, that there exists a slot used to execute $J_{\ell,m}$ in S that is used to load it in S'. This would mean either that a contiguous sequence of slots used for loading (a loading block in the following) preceding the concerned execution block (a contiguous sequence of slots used for executing) in S is interrupted in S', which is not possible because the same slots are allocated to $J_{\ell,m}$ in S and S', or that the loading sequence is of the same length in both schedules but is sufficient in S but not in S' to lead to an execution. The only possibility is then that the sequence corresponds in S to an RD and in S' to an SD. For example, this occurs when the added slot is just before the execution of a higher priority job while $J_{\ell,m}$ has not yet started its execution. Even then, this assumes that $SD_\ell < RD_\ell$, which is impossible under the task model assumptions ($\forall i\ SD_i \geq RD_i$). △

Now let us note that the additional time slot in S' is either isolated (preceded and succeeded by a slot allocated to another job) or connected to slots already allocated to $J_{\ell,m}$. We therefore have a contiguous sequence of slots (a block in the following) allocated to $J_{i,k}$.

There are two cases, depending on the length of this block: *(i)* it can be less than or equal to λ_ℓ or *(ii)* greater than λ_ℓ where λ_ℓ is either equal to SD_ℓ when we consider a starting block or equal to RD_ℓ when we consider a resuming block.

Case (i): The block is composed exclusively of loading slots, it does not aid the job in progressing in S'. In S, the block was even shorter and the job was not progressing either. The remainder of the schedule remains unchanged.

Case (ii): the new block in S' is of size $s > \lambda_\ell$, which means that the first λ_ℓ slots are used in S' to load the job, while the $s - \lambda_\ell$ last ones will be available to execute it. In S the block size was at most $s - 1$, so the total number of potential execution slots in S' is necessarily increased by at least one compared to S. Indeed, all slots used to execute $J_{\ell,m}$ in S are also available to execute it in S' and as previously proved, no execution slot can become a loading one.

Since the total number of execution slots is always equal to C_ℓ that implies that the last slot affected in S for the execution of the job is no longer necessary to complete it, and will so be made available to other jobs in S'. So $J_{\ell,m}$ terminates earlier in S' than in S (not necessarily one time earlier, because some preceding loading slots can also become irrelevant, e.g. if the last slots allocated in S is λ_ℓ loadings and one unique execution). Note that the consequences of this early termination on other jobs to consider are exactly the same as for $J_{i,k}$ early termination.

We so prove that considering an early termination for a job can result in an early termination of either zero, one or several other jobs. □

Note that it is needed to consider Non-Resumable Starting Delays (NRSDs) with $\forall i\ SD_i \geq RD_i$ to have (and prove) this property. Without SDs (or with SD lesser than RD), starting a job earlier due to early termination of an higher priority job can result in adding a preemption later, and so an associated loading time (see Figure 5). Moreover with Non-Preemptive Resuming Delay (NPRD), starting a job earlier can result in starting a non-preemptive block of operations that will lead to completely change the schedule due to priority inversions.

Theorem 5 (C-sustainability)**.** *The test consisting in scheduling the system for its simulation interval is sustainable regarding the WCET when NRLD preemption model is considered.*

Proof. This is a direct consequence of Theorem 4: if all deadlines are met when a collection of jobs is scheduled according to their WCET for a given interval of time, if some of them terminate earlier (are executed for less than their WCET or are started earlier), then the completion times being lesser than or equal to the ones in the initial scenario, the deadlines are met. □

5.4.2 Sustainability regarding Loading Delays

We will establish the LD-sustainability, directly stemming from the C-sustainability.

Theorem 6 (LD-sustainability)**.** *The test consisting in scheduling the system for its simulation interval is sustainable regarding the* SD *and* RD *delays when NRLD preemption model is considered.*

Proof. This is a direct consequence of the C-sustainability. Indeed, when a job is ready for execution earlier than expected, it will result in an early completion of that same job, since all the allocated slots until its completion will still be allocated to the same jobs and that the total number of execution cannot increase (as for the C-sustainability, it is because SD are considered). The consequences for the remaining of the schedule will be exactly the same as if the job did execute for less than its WCET. Since the test is C-sustainable, it will also be LD-sustainable. □

5.4.3 Sustainability regarding job arrival times

Unfortunately, the test consisting in simulating the system until the end of the simulation interval is not T-sustainable with the considered framework. That is, if a system passes the test, it may no longer pass the test if a job is released *later* than initially (as depicted by counterexample in Figure 7). This counterexample also highlight the fact that the synchronous scenario is not the worst-case scenario. This justifies the interest in studying *a*synchronous systems for this model in this work.

5.5 Robust schedulability tests

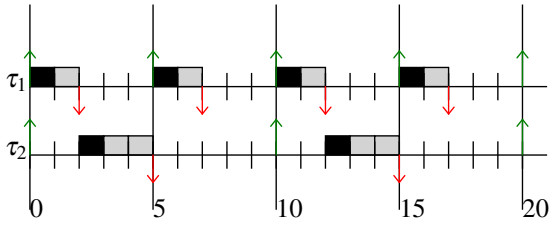
We now have all the elements needed to establish two exact and robust schedulability tests.

Test 1 (EDF-like schedulers)**.** *Let A be an CFJP-scheduler, then any asynchronous constrained deadline periodic tasks with NRLD upon uniprocessor platform is A-schedulable* iff (i) *we do not miss any deadline within the interval* $[0, O^{\max} + 2H)$*, and* (ii) $\forall i\ e_{i,O^{\max}+H} = e_{i,O^{\max}+2H}$.

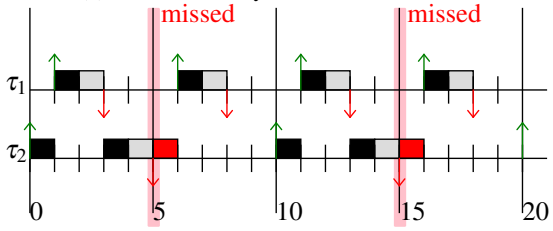
Proof. This is a direct consequence of Theorem 2. □

Test 2 (FTP schedulers)**.** *Let A an FTP-priority assignment* ($\tau_1 \succ \tau_2 \succ \cdots \succ \tau_n$)*, then any asynchronous constrained deadline periodic tasks with NRLD upon uniprocessor platform (with* and *without Starting Delay) is A-schedulable* iff (i) *we do not miss any deadline within the interval* $[0, S_n + H)$*, and* (ii) $\forall i\ e_{i,S_n} = e_{i,S_n+H}$*. (Where* S_n *is defined inductively in Definition 8.)*

Proof. This is a direct consequence of Theorem 3. □



(a) Schedulable: cycle of size 10 starts at 0



(b) A deadline is missed in each cycle

Figure 7: **– T-sustainability counterexample.** We consider the schedule produced by EDF of a system composed of task $\tau_1 = \langle O_1 = 0, C_1 = 1, T_1 = 5, D_1 = 2, SD_1 = 1, RD_1 = 1 \rangle$ and task $\tau_2 = \langle 0, 2, 10, 5, 1, 1 \rangle$. Sub-figure 7a shows the schedule until time 20, which is sufficient to observe that the system is schedulable since time 10 is equivalent to time 0. However if the first job of τ_1 arrives later, say at time 1 (and the third one at time 11) it leads to the activation scenario depicted in Sub-figure 7b, which exhibits that in that case the first job of τ_2 misses its deadline (red square at times 5 and 15).

Moreover, those tests are *robust*. This is a consequence of the sustainability properties (theorems 4, 5, and 6).

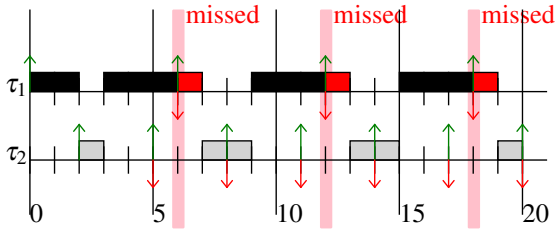
6 Conclusion

As observed in Section 4, taking into account the time required for context switches as part of the WCETs introduces pessimism. Moreover, especially if this time is long compared to the WCETs (for example, due to additional security operations or other high-level RTOS services), the classical theoretical results from the literature are no longer applicable.

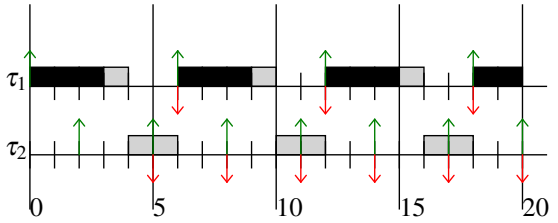
This is due to the fact that CS operations are implemented as NP code sections within most RTOS and therefore do not behave like normal executions. Considering this in the task model, as we have done in [19], poses major issues for system analysis: intractable simulation intervals, scheduling anomalies, and lack of sustainability properties.

The NR model proposed in this paper addresses these issues by introducing a more refined representation. Unfortunately, the classical theoretical results from the literature are no longer applicable to this model. Nevertheless, as discussed in Section 5.2 and Section 5.3, it is feasible to limit the simulation intervals for FTP and EDF with reasonable bounds.

From a theoretical perspective, the model appears promising, particularly owing to its absence of scheduling anomalies and the sustainability properties of the schedulability tests associated with a simulation over a finite interval, as demonstrated in Section 5.4. Reducing these bounds or exploring alternative methods to determine schedulability are short-term open challenges in this research. This could be investigated by seeking a sufficient condition or calculating bounds on response times (and certainly on the number of preemptions). Another question raised by this model would be to propose an effective



(a) Not EDF-schedulable (deadline miss at 6)



(b) Schedulable with $\tau_1 \succ \tau_2$ (a 6 length cycle starts at time 0)

Figure 8: **– EDF Non Optimality.** Sub-figure 8a shows the schedule obtained with EDF of a system composed of task $\tau_1 = \langle O_1 = 0, C_1 = 1, T_1 = 6, D_1 = 6, SD_1 = 3, RD_1 = 3 \rangle$ and task $\tau_2 = \langle 2, 1, 3, 3, 0, 0 \rangle$. System is not EDF-schedulable. Sub-figure 8b shows the same system successfully scheduled under FTP with $\tau_1 \succ \tau_2$.

scheduling algorithm. As illustrated in Figure 8, EDF is no longer optimal when considering this model with periodic tasks.

The model appears entirely realistic from a practical standpoint as long as an interrupt handler featuring multiple priority levels is available. Future works will need to demonstrate this feasibility by implementing this model in an RTOS.

It also appears that this model, defining very generally what constitutes LDs, efficiently models other mechanisms that one might want to include in a modern real-time system. For example, security mechanisms as described in [6] where it is considered that for each preemption, a security handler clears the cache of critical tasks to guard against the risk of a task attempting to read the data left in the cache.

The main drawback of the approach proposed in this paper for testing schedulability is that it is not applicable to a sporadic task model.

In conclusion, we thus proposed a realistic task model that considers the costs of preemption (and loading) that is not subject to scheduling anomalies. This model decreases the pessimism of classical approaches and permits a correct analysis in the case where LDs are not negligible parts of WCETs. For this model and classical schedulers (FTP, EDF), we provided a reasonable simulation interval. This provides *exact* schedulability tests, sustainable with respect to WCETs and LDs, taking preemptions into account.

Glossary

Acronyms

CFJP Cyclic Fixed Job Priority. 1, 3, 4, 9, 11, 13–16, 18

CRPD Cache-Related Preemption Delay. 5

CS Context Switch. 2, 3, 6–9, 18

DM Deadline Monotonic. 3

EDF Earliest Deadline First. 2–4, 8, 10, 11, 18, 19, 24, 26, 27

FJP Fixed Job Priority. 1, 5, 6, 10

FTP Fixed Task Priority. 1–6, 10, 14, 15, 18, 19, 27

IRQ Interrupt Request. 6, 7

ISR Interrupt Service Routine. 6, 7

LD Loading Delay. 1, 2, 7–9, 15, 17, 19, 25

LLF Least Laxity First. 2

NP Non-Preemptive. 5, 7, 18

NPLD Non-Preemptive Loading Delay. 8, 10

NPRD Non-Preemptive Resuming Delay. 17

NR Non-Resumable. 1–3, 7, 9, 18

NRLD Non-Resumable Loading Delay. 1, 7, 8, 10, 11, 14–18

NRSD Non-Resumable Starting Delay. 17

RD Resuming Delay. 7–9, 16, 17, 24, 25

RM Rate Monotonic. 3, 4, 10, 14

RTOS Real-Time Operating System. 1, 3, 6, 9, 18, 19

SD Starting Delay. 3, 7–9, 15–17, 24, 25

WCET Worst-Case Execution Time. 2–6, 8, 9, 15–19

References

References

[1] AHMED, S., AND ANDERSON, J. H. Tight Tardiness Bounds for Pseudo-Harmonic Tasks Under Global-EDF-Like Schedulers. In *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)* (Dagstuhl, Germany, 2021), B. B. Brandenburg, Ed., vol. 196 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 11:1–11:24.

[2] AHMED, S., AND ANDERSON, J. H. Exact response-time bounds of periodic dag tasks under server-based global scheduling. In *Real-Time Systems Symposium (RTSS)* (2022), IEEE, pp. 447–459.

[3] ALTMEYER, S., DAVIS, R. I., AND MAIZA, C. Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. *Real-Time Systems 48*, 5 (2012), 499–526.

[4] AUDSLEY, N. C. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Tech. Rep. YCS-164, Department of Computer Science, University of York, 1991.

[5] BARUAH, S. The limited-preemption uniprocessor scheduling of sporadic task systems. In *17th Euromicro Conference on Real-Time Systems (ECRTS'05)* (2005), pp. 137–144.

[6] BARUAH, S. Security-cognizant real-time scheduling. In *2022 IEEE 25th International Symposium On Real-Time Distributed Computing (ISORC)* (2022), pp. 1–9.

[7] BARUAH, S., AND BURNS, A. Sustainable scheduling analysis. In *27th IEEE International Real-Time Systems Symposium (RTSS'06)* (2006), pp. 159–168.

[8] BERTOGNA, M., AND BARUAH, S. Limited preemption edf scheduling of sporadic task systems. *IEEE Transactions on Industrial Informatics 6*, 4 (2010), 579–591.

[9] BIMBARD, F. *Dimensionnement temporel de systèmes embarqués : application à OSEK*. Phd thesis, CNAM, Paris, France, 2007.

[10] BURNS, A. Preemptive priority based scheduling: An appropriate engineering approach. In *Principles of Real-Time Systems* (1994), Prentice Hall, pp. 225–248.

[11] BURNS, A., TINDELL, K., AND WELLINGS, A. Effective analysis for engineering real-time fixed priority schedulers. *IEEE Transactions on Software Engineering 21*, 5 (1995), 475–480.

[12] BUSQUETS-MATAIX, J., SERRANO, J., ORS, R., GIL, P., AND WELLINGS, A. Adding instruction cache effect to schedulability analysis of preemptive real-time systems. In *Real-Time Technology and Applications* (1996), pp. 204–212.

[13] CHOQUET-GENIET, A., AND GROLLEAU, E. Minimal schedulability interval for real time systems of periodic tasks with offsets. *Theoretical of Computer Sciences 310* (2004), 117–134.

[14] COLIN, W. *Embedded RTOS Design: Insights and Implementation*. Elsevier Ltd, 2021.

[15] DRIDI, M., SINGHOFF, F., RUBINI, S., AND DIGUET, J.-P. ECTM: a network-on-chip communication model to combine task and message schedulability analysis. *Journal of Systems Architecture 114* (2021), 101931.

[16] GOOSSENS, J., AND DEVILLERS, R. The non-optimality of the monotonic priority assignments for hard real-time offset free systems. *Real-Time Syst. 13*, 2 (sep 1997), 107–126.

[17] GOOSSENS, J., AND DEVILLERS, R. Feasibility intervals for the deadline driven scheduler with arbitrary deadlines. In *The 6th IEEE International Conference on Real-time Computing Systems and Applications* (1999), pp. 54–61.

[18] GOOSSENS, J., GROLLEAU, E., AND CUCU-GROSJEAN, L. Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms. *Real-time systems 52*, 6 (2016), 808–832.

[19] GOOSSENS, J., AND MASSON, D. Simulation intervals for uniprocessor real-time schedulers with preemption delay. In *The 30th International Conference on Real-Time Networks and Systems (RTNS) 2022* (Paris, France, June 2022), ACM, pp. 36–45.

[20] KATCHER, D. I., ARAKAWA, H., AND STROSNIDER, J. K. Engineering and analysis of fixed priority schedulers. *IEEE transactions on Software Engineering 19*, 9 (1993), 920–934.

[21] LEE, C.-G., HAHN, H., SEO, Y.-M., MIN, S. L., HA, R., HONG, S., PARK, C. Y., LEE, M., AND KIM, C. S. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. *IEEE Transactions on Computers 47*, 6 (1998), 700–713.

[22] LEUNG, J. Y.-T., AND MERRILL, M. A note on preemptive scheduling of periodic, real-time tasks. *Information processing letters 11*, 3 (1980), 115–118.

[23] LEUNG, J. Y.-T., AND WHITEHEAD, J. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation 2*, 4 (1982), 237–250.

[24] LIU, C. L., AND LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM 20*, 1 (jan 1973), 46–61.

[25] LUNNISS, W., ALTMEYER, S., MAIZA, C., AND DAVIS, R. I. Integrating cache related pre-emption delay analysis into edf scheduling. In *19th Real-Time and Embedded Technology and Applications Symposium (RTAS)* (2013), IEEE, pp. 75–84.

[26] MASSON, D. Draw schedule, 2024. https://igm.univ-mlv.fr/~masson/draw-schedule.

[27] MEUMEU YOMSI, P. *Prise en compte du coût exact de la préemption dans l'ordonnancement temps réel monoprocesseur avec contraintes multiples*. Phd thesis, Université Paris Sud – Orsay, Paris, France, 2009.

[28] PHAVORIN, G., RICHARD, P., GOOSSENS, J., CHAPEAUX, T., AND MAIZA, C. Scheduling with preemption delays: anomalies and issues. In *The 23rd International Conference on Real Time and Networks Systems* (2015), pp. 109–118.

[29] STASCHULAT, J., AND ERNST, R. Scalable precision cache analysis for preemptive scheduling. In *ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems* (2005), pp. 157–165.

[30] TOMIYAMA, H., AND DUTT, N. Program path analysis to bound cache-related preemption delay in preemptive real-time systems. In *Proceedings of the Eighth International Workshop on Hardware/Software Codesign. CODES 2000 (IEEE Cat. No.00TH8518)* (2000), pp. 67–71.

[31] TRAN, H. N., RUBINI, S., BOUKHOBZA, J., AND SINGHOFF, F. Feasibility interval and sustainable scheduling simulation with CRPD on uniprocessor platform. *Journal of Systems Architecture 115* (2021), 102007.

[32] WANG, Y., AND SAKSENA, M. Scheduling fixed-priority tasks with preemption threshold. In *Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No.PR00306)* (1999), pp. 328–335.

[33] YOMSI, P. M., AND SOREL, Y. Extending rate monotonic analysis with exact cost of preemptions for hard real-time systems. In *Euromicro Conference on Real-Time Systems* (2007), IEEE, pp. 280–290.