Noname manuscript No.
(will be inserted by the editor)

Online and Offline Scheduling with Cache-Related Preemption Delays

Guillaume Phavorin · Pascal Richard · Joël Goossens · Claire Maiza · Laurent George · Thomas Chapeaux

Received: date / Accepted: date

Abstract In this paper, we consider the problem of scheduling hard real-time tasks subjected to preemption delays on a uniprocessor system. While most of the existing work focus on either reducing these additional delays or improving the system predictability by bounding them, we focus in this work on the problem of taking scheduling decisions while considering preemption delays. We first study the behavior of existing online scheduling policies such as RM and EDF when accounting for preemption delays. Then we prove that there exists no optimal online algorithm for the problem of scheduling sporadic tasks with preemption delays. Eventually, we propose an optimal offline solution to the problem of scheduling tasks subjected to preemption delays using mathematical programming.

Keywords Offline Scheduling · Online Scheduling · RM · EDF · Preemption Delays · CRPD · Sustainability · Mathematical Programming

Guillaume Phavorin
LIAS, Université de Poitiers, France
E-mail: guillaume.phavorin@univ-poitiers.fr

Pascal Richard
LIAS, Université de Poitiers, France
E-mail: pascal.richard@univ-poitiers.fr

Joël Goossens
PARTS, Université Libre de Bruxelles, Belgium
E-mail: joel.goossens@ulb.ac.be

Claire Maiza
Verimag, Université Grenoble-Alpes, France
E-mail: claire.maiza@imag.fr

Laurent George
LIGM, UPEM - ESIEE Paris, France
E-mail: laurent.george@univ-mlv.fr

Thomas Chapeaux
PARTS, Université Libre de Bruxelles, Belgium
E-mail: thomas.chapeaux@ulb.ac.be

1 Introduction

Commercial off-the-shelf components are now widespread in real-time embedded systems as they allow significant cost reduction. They include several micro-architectural features designed to improve the overall performance, such as pipelines and cache memories. A pipeline enables acceleration by overlapping the execution steps (fetch, decode, execution, memory access, register write back) of different instructions, as those steps require separate hardware circuits. Caches are fast memories located between the processor registers and the main memory. They aim at bridging the speed gap between the processor and the main memory by exploiting spatial (i.e. lines of contiguous blocks, for example instruction blocks) and temporal localities (e.g. loops where blocks are reused several times).

These technologies increase significantly the processor performances, and not considering them would lead to a waste of hardware resources. But on the other hand, they make the system behavior harder to predict. Because of the pipeline, the instruction execution behavior depends on the system execution history. As for the cache, the instruction execution time depends whether this instruction can be found in the cache (cache hit) or has to be loaded from the main memory (cache miss), which can be up to 10 times more costly as shown for example in Levinthal (2009). This problem is significant as predictability is a core matter for critical embedded systems. It becomes even worse when considering multiple tasks. Those tasks might preempt, i.e. interrupt for some amount of time, each other causing *preemption delays*. As stated in Buttazzo (2011), those delays are due to different sources. *Scheduling delays* result from the scheduler invocation and context switches. *Pipeline delays* correspond to the time needed to flush the pipeline after a task preemption, and the time to refill it when the task resumes its execution. *Bus-related delays* result from potential contention in the bus used to access the main memory. Finally, *cache-related preemption delays* (CRPDs) might occur as one task can overwrite some cache locations used by a preempted task. When resuming its execution, the preempted task may experience additional cache misses which would not have occurred if the task had not been preempted. This results in additional reloads from the main memory. Scheduling, pipeline and bus delays can be bounded by a constant as stated in Altmeyer et al (2012) and are often less penalizing than the CRPD. Actually, the CRPD can be as high as 44% of the task worst-case execution time (WCET) as shown in Pellizzoni and Caccamo (2007). So, these additional delays can be responsible for possible deadline misses and, as a consequence, cannot be neglected anymore when performing real-time scheduling analyses.

Different ways to deal with preemption delays have been proposed in the literature. Some works focus on reducing those delays as in Ding et al (2014). In other works, upper bounds are computed on preemption delays to take them into account in the schedulability analysis as an additional delay as in Altmeyer et al (2012). In both cases, no change occurs at the scheduling level. Popular algorithms such as Rate Monotonic (RM) or Earliest Deadline

First (EDF) are used. Some recent works introduce scheduling modifications (for example preemption points in Bertogna et al (2011)) but they still use common fixed-priority schedulers.

For hard real-time systems, scheduling algorithms can be either offline or online. An offline scheduler computes its entire schedule before the system starts running and uses it at runtime. So, it needs a complete knowledge of all job parameters (releases, execution times, deadlines). On the contrary, an online scheduler makes its scheduling decisions at runtime. It only knows the system current state, i.e. the parameters of all released jobs. On the one hand, offline schedulers have often been used in real-time systems (Burns (1995), Bate (1998)), because they require less runtime overhead than online ones and do not suffer any scheduling anomaly (Xu and Parnas (1993)). But on the other hand, online algorithms are very popular in the real-time scheduling literature because they allow more flexibility, as they can deal with unpredicted events. Among online scheduling algorithms, EDF is widely studied, because it is optimal for scheduling hard real-time independent jobs, as long as no preemption delay is considered. This optimality result does not stand any more when preemption delays are considered. Actually, the problem of scheduling with preemption delays is NP-hard as proved in Phavorin et al (2015b) and neither fixed-task nor fixed-job priority-based scheduling algorithms can be optimal for this problem.

This Research. We study the problem of scheduling real-time tasks subjected to preemption delays. These delays are represented by timed penalties which are inputs for our problem. They are assumed to be computed beforehand by a timing analysis, as task WCETs are in common scheduling approaches (see for example Altmeyer et al (2012)). As we do not consider explicitly any cache analysis, the problem of scheduling with preemption delays is independent from any cache management strategy, such as partitioning. We consider hereafter both the online and the offline scheduling problems.

Paper Organization. The remainder of this paper is organized as follows: in Section 2, we present some works dealing with preemption delays. In Section 3, we define the problem of scheduling when considering preemption delays. In Section 4, we consider the online scheduling problem. In particular, we illustrate several scheduling anomalies that may occur at runtime if an online scheduling algorithm is used. We also show that no online algorithm can be optimal for the problem of scheduling independent sporadic tasks when considering preemption delays. In Section 5, we propose an optimal solution to this problem using offline scheduling. Then, in Section 6, we compare our solution with RM and EDF and quantify the loss of schedulability due to CRPDs. Finally, we conclude in Section 7 and present some future work.

2 Related work

For hard real-time systems, predictability has to be ensured, i.e. all deadlines will always be met at runtime. When preemption delays are considered, those additional delays have to be taken into account at some point either in the WCET or during the schedulability analysis or test. To achieve such a goal, the first step consists in bounding the preemption delay a task can experience when resuming its execution after a preemption. As it is difficult to ensure worst-cases through measurement, static analysis is mostly preferred to compute upper bounds on preemption delays. In particular, in the case of CRPDs, a cache analysis is proposed in Ferdinand and Wilhelm (1999) using abstract interpretation. Then, bounds on the CRPD can be computed using Evicting Cache Blocks, ECBs, i.e. blocks used by a preempting task that might override some cache locations used by the preempted task (Tomiyama and Dutt (2000)), or Useful Cache Blocks, UCBs, i.e. blocks used by a task which are reused latter on and will have to be reloaded if evicted from the cache because of a preemption (Lee et al (1998)). Those bounds are incorporated either in the WCET or in a schedulability analysis to ensure determinism. To add the CRPD to the task WCET (see for example Brandenburg (2011)), a safe estimation of the maximal number of preemptions the task may experience has to be computed as proposed for example in Yomsi and Sorel (2007). To incorporate the CRPD during the schedulability analysis, most authors focus on fixed-task priority scheduling and use the Response Time Analysis, considering either the preempting task (Busquets-Mataix et al (1996b)), the preempted one (Lee et al (1998)) or both of them (Altmeyer et al (2012)). An extension to EDF is proposed in Lunniss et al (2013).

Bounding the preemption delays to take them into account either in the WCET or during the schedulability analysis allows us to overcome the predictability problem. But, these additional delays can also threaten the system schedulability because of the overall overhead which is incurred. So, several works focus on reducing preemption delays. CRPD is mostly studied as it accounts for the most part of these delays. CRPD reduction can be achieved at hardware level by designing new cache management policies as proposed for example in Reineke et al (2014). Another solution to reduce the overall CRPD is to prevent tasks from overwriting each other cache locations, using either partitioning (Altmeyer et al (2014)) or locking techniques (Ding et al (2014)) or even a combination of both of them (Vera et al (2003)). Other works, such as in Lunniss et al (2012), prefer to use task code placement in the main memory to reduce interference in the cache (because of the cache mapping). Memory management can also be modified to completely eliminate the CRPD as proposed in Whitham and Audsley (2012): the cost of saving and later restoring the cache content when a preemption occurs is then included in the WCET of the preempting task. At the scheduling level, works focus on reducing the number of preemptions but without necessarily considering preemption delays. Preemption number reduction can be achieved through the use of preemption thresholds (Keskin et al (2010); Wang et al (2015)) or deferred preemption

(Yao et al (2011)). Very few works have indeed focused on devising scheduling policies that *take their scheduling decisions using preemption delays.* In Bril et al (2014), preemption delays are used to compute thresholds whereas in Bertogna et al (2011); Peng et al (2014); Cavicchio et al (2015), preemption points are selected to minimize the overall preemption overhead. Some work also exists as far as multiprocessors are concerned. But most of them deal with partitioned multiprocessor soft real-time scheduling: cache-aware decisions influence only the taskset partitioning process in order to reduce conflicts between tasks as in Calandrino and Anderson (2008).

None of those prior works deal with the general problem of scheduling with preemption delays, i.e. find an optimal scheduling algorithm (if any). Considering the preemption delay as an additional timing parameter radically changes the scheduling problem.

3 Problem Statement

The classic Liu and Layland task model (Liu and Layland (1973)) has to be slightly modified to take preemption delays into account. An *additional parameter*, s_i, is used to represent an upper bound on the additional delay task τ_i has to pay each time it is preempted. The s_i parameter represents the extra-work added to the task processing requirement every time it is preempted. This parameter can model:

– the context switch delay as in Lee and Shin (2014): $s_i = \alpha$,
– the CRPD as in Altmeyer et al (2011); Lee et al (1998): $s_i = \text{BRT} \cdot |\text{UCB}_i|$,
– or a combination of both: $s_i = \alpha + \text{BRT} \cdot |\text{UCB}_i|$.

where $|\text{UCB}_i|$ represents the maximum number of Useful Cache Blocks for Task τ_i and BRT is the Block Reload Time, i.e. the time needed to load a memory block from the main memory into the cache.

We assume all s_i values to be computed beforehand as WCETs are in the scheduling theory.

A *periodic* (i.e. recurrent) *task* τ_i is represented by the tuple $\tau_i(C_i, T_i, D_i, s_i)$, standing respectively for the task WCET, its period, its relative deadline and finally an upper bound on its preemption delay. An additional parameter, o_i, may be added to represent the task first release date. In this paper, we consider, unless specified, synchronous tasks, i.e. $o_i = 0, \forall i$, with implicit deadlines, i.e. $D_i = T_i, \forall i$. A periodic task τ_i generates an infinite sequence of jobs noted: $J_{ij}(r_{ij}, p_{ij}, d_{ij}, s_{ij})$ (being respectively the job release date, its execution time, its absolute deadline and an upper bound on its preemption delay). If not stated otherwise, we consider that we have: $p_{ij} = C_i$. This means that all jobs of a same periodic task have the same worst-case execution time. They are also considered to have the same preemption delay bound $s_{ij} = s_i$. For synchronous periodic tasks, we consider the jobs generated over the hyperperiod, i.e. the least common multiple of the task periods. We recall that a processor utilization can be computed for each task as: $u_i = \frac{C_i}{T_i}$. The total processor utilization for a taskset is equal to: $U = \sum_i u_i$.

Sporadic tasks are a generalization of the periodic case. For a sporadic task τ_i, T_i represents the *minimum* inter-arrival time between two consecutive jobs of the task. A sporadic taskset τ can potentially generate an infinite number of distinct real-time *instances* over different executions of taskset τ. A given instance of τ that meets all deadlines is said to be a *feasible instance.* Informally, a sporadic taskset τ is feasible if, and only if, for every possible real-time instance of τ, there exists a schedule that meets all deadlines (Fisher et al (2010)).

The *offline scheduling* problem consists in computing a static schedule before runtime. It has a complete knowledge of the whole system life, i.e. all jobs that will be issued and all their parameters (Mok (1983)). The *online scheduling* problem consists in taking a scheduling decision at a given date knowing only the current system state, i.e. the parameters of all jobs which have been released at that date. Jobs are released over time. When a job is released, all its parameters are assumed to be known. EDF is an online scheduler as its scheduling decisions are only based on the absolute deadlines of the ready jobs. No knowledge of the future is needed.

In this work, as it is often the case in other papers (see for example Busquets-Mataix et al (1996a) and Lee et al (1998)), we assume the preemption delay bound s_i to be paid in a whole immediately when a task resumes its execution after a preemption. In practice, this delay is spread over the task execution. For example, when considering the CRPD, an additional cost is incurred every time a task references a memory block (either instructions or data) that is no longer in the cache because of the interference due to preempting tasks. Moreover, we consider that, if a task is preempted while paying its preemption delay s_i, then another full preemption delay s_i is also paid. Such an assumption is pessimistic but predictable. Consider again the CRPD and suppose that a task has a memory block that has been evicted from the cache by a preempting task. The task will need to reload this block sometimes after resuming its execution. If the task is preempted once again before the missing memory block is re-referenced, then the block is not in the cache as it has not yet been reloaded. So the Block Reload Time penalty will have to be paid only once. More precision may be gained by accounting for the exact preemption points which, in turn, means a better knowledge of accessed memory blocks (see for example Cavicchio et al (2015)). The simplifications used in this paper, although introducing pessimism, allow us to deal with a simpler preemption delay model due to the fact that it is independent of the preemption point (where the preemption occurs in the task code).

In the remainder of this paper, we show that unfortunately when considering preemption delays, no online scheduler is optimal and numerous scheduling anomalies can occur. So, we propose an optimal offline approach to schedule real-time tasks subjected to CRPDs and use it to evaluate the loss of schedulability of existing scheduling policies.

4 The *Online* Scheduling Problem

When dealing with systems subjected to hard real-time constraints, worst-case behaviors are considered to determine taskset schedulability. As shown in Baruah and Burns (2006), those cases are not necessarily obvious to determine. To study this matter, Burns and Baruah introduce the notion of *sustainability* (see Definition 1 hereafter).

In the remainder of this section, we show that RM, DM and EDF are not sustainable when preemption delays are considered. Next, we consider the more general issue of designing an optimal scheduling algorithm when preemption delays are considered. We prove that no online scheduling algorithm can be optimal as clairvoyance is needed with regard to release dates.

4.1 Non-sustainability of RM, DM and EDF accounting for preemption delays

We first recall the definition of sustainability given in Burns and Baruah (2008):

Definition 1 *A scheduling policy and/or 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 task(s) are changed in any, some, or all of the following ways:*

1. *decreased execution requirements,*
2. *larger periods, and*
3. *larger relative deadlines.*

Note that, actually, Burns and Baruah also consider the impact of the task jitter. In this paper, we do not deal with this parameter.

To be sustainable, a scheduling policy and/or a schedulability test must be sustainable with regard to all the parameters listed above. As stated in Burns and Baruah (2008), RM and DM are sustainable with regard to execution requirements and relative deadlines but not with regard to the period parameter. The EDF scheduling policy for periodic tasks is sustainable when no preemption delay is considered, as soon as some conditions are fulfilled:

– EDF is sustainable with regard to execution requirements and relative deadlines for any periodic taskset,
– EDF is sustainable with regard to the period parameter for zero-offset (synchronous) periodic tasksets.

We now consider the sustainability of RM, DM and EDF when preemption delays are considered. Under the preemption delay-aware model, we extend Definition 1 by introducing the notion of *sustainability with regard to preemption delays.*

Tasks $\tau_i(C_i,T_i,D_i,s_i)$	Generated jobs $J_{ij}(r_{ij},p_{ij},d_{ij},s_{ij})$
$\tau_1(1,4,4,s_1)$	$J_{11}(0,1,4,s_1)$ $J_{12}(4,1,8,s_1)$ $J_{13}(8,1,12,s_1)$
$\tau_2(3,12,12,s_2)$	$J_{21}(0,3,12,s_2)$
$\tau_3(3,12,12,s_3)$	$J_{31}(0,3,12,s_3)$
$\tau_4(2,12,12,s_4)$	$J_{41}(0,3,12,s_4)$

Table 1 Task and job characteristics for the proofs of Theorems 1 ($s_1 = s_2 = s_3 = s_4 = 0.6$) and 4 ($s_1 = s_2 = s_4 = 1$ and $s_3 = 1$ then 0.6)

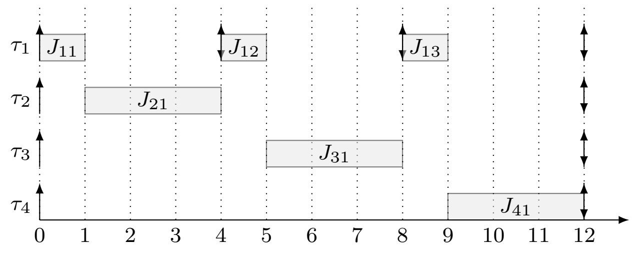


Fig. 1 Schedule with $s_{21} = s_{31} = 0.6$ and $C_2 = 3$ for RM, DM and EDF

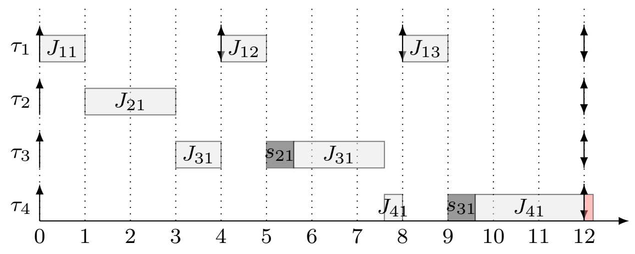


Fig. 2 Schedule with $s_{21} = s_{31} = 0.6$ and $C_2 = 2$ for RM, DM and EDF

Definition 2 *A scheduling policy and/or a schedulability test is* sustainable with regard to the preemption delay parameter *if any system deemed schedulable by the schedulability test remains schedulable when the value of the preemption delay parameter of one or more individual task(s) is decreased.*

The motivation for considering preemption delay for the sustainable analysis is similar to the one for WCET given in Burns and Baruah (2008): only upper bounds on preemptions delays are considered. Moreover, those bounds are computed independently of the real program point at which the preemption will occur. So for all realistic systems, variability in preemption delays is to be expected and, as a consequence, sustainability with regard to this parameter is required.

We now study the sustainability of RM, DM and EDF when preemption delays are considered.

Theorem 1 RM, DM *and* EDF *are not sustainable with regard to the execution requirement parameter when preemption delays are considered.*

Proof We consider the example of a system composed of four synchronous periodic tasks τ_1, τ_2, τ_3 and τ_4 whose characteristics are synthesized in Table 1. The preemption delay parameter is the same for the four tasks and is equal to 0.6. For RM, DM and EDF, we assume that task indexes are used to break the ties, which means that the priorities and the resulting schedules are the same for the three scheduling policies. Over the hyperperiod, τ_1 issues four jobs J_{11}, J_{12} and J_{13} whereas τ_2, τ_3 and τ_4 issue one job each, respectively J_{21}, J_{31}, J_{41}. All job characteristics can also be found in Table 1.

Figure 1 depicts the schedule constructed by either RM, DM or EDF over the hyperperiod. The system experiences no preemption delay and the taskset is schedulable. Note that, for graphical representation ease, we depict the delay s_{ij} incurred by a preemption immediately after the preempted job J_{ij} resumes its execution. Remember that, if s_{ij} models the CRPD, then it is actually spread over the job execution.

However, decreasing the execution time of J_{21} to 2 makes the taskset unschedulable. Indeed, as depicted in Figure 2, because of J_{21}'s lesser execution time, J_{31} and J_{41} can execute earlier. The system experiences two preemptions and J_{41} misses its deadline because of the overall preemption overhead.

So, RM, DM and EDF are no longer sustainable with regard to the execution requirement parameter when preemption delays are considered. □

Theorem 2 EDF *is not sustainable with regard to the deadline parameter when preemption delays are considered.*

Proof The system is composed of three synchronous periodic tasks τ_1, τ_2 and τ_3 whose characteristics are synthesized in Table 2. The CRPD parameter for each task is equal to 1. Over the hyperperiod, τ_1 issues three jobs J_{11}, J_{12} and J_{13}, τ_2 releases two, J_{21} and J_{22}, and τ_3 only one, J_{31}.

Figure 3 depicts the schedule constructed by EDF over the hyperperiod. All deadlines are met and the system is schedulable.

However, if we increase D_3 to 11, then J_{31} experiences two preemptions and a deadline is missed, as depicted in Figure 4. As a consequence, the system is not schedulable anymore. Indeed, as EDF uses absolute deadlines to compute job priorities, J_{31} has now a lower priority than J_{12} (resp. J_{22}), at time 4 (resp. 6) resulting in two preemptions. Note that, at time 8, EDF has to break the ties to compute priorities as both J_{31} and J_{13} have the same absolute deadline. We assume that task indexes are used as tie breaker which means that J_{13} is given a higher priority than J_{31} as depicted in Figure 4. Note that considering the reverse (i.e. J_{31} with a higher priority than J_{13}) would also result in the system not being schedulable any more as, this time, J_{13} would miss its deadline.

So, EDF is no longer sustainable with regard to the deadline parameter when preemption delays are considered. □

Theorem 3 EDF *is not sustainable with regard to the period parameter when preemption delays are considered.*

Tasks $\tau_i(C_i, T_i, D_i, s_i)$	Generated jobs $J_{ij}(r_{ij}, p_{ij}, d_{ij}, s_{ij})$
$\tau_1(1, 4, 3, 1)$	$J_{11}(0, 1, 3, 1)$ $J_{12}(4, 1, 7, 1)$ $J_{13}(8, 1, 11, 1)$
$\tau_2(2, 6, 4, 1)$	$J_{21}(0, 2, 4, 1)$ $J_{22}(6, 2, 10, 1)$
$\tau_3(3, 12, 6, 1)$	$J_{31}(0, 3, 6, 1)$

Table 2 Task and job characteristics for the proof of Theorem 2

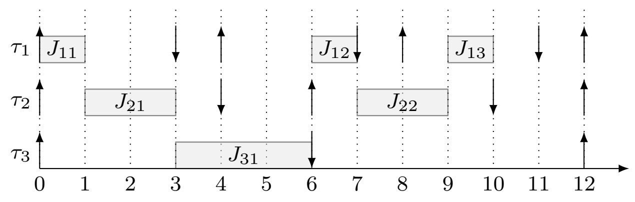


Fig. 3 EDF schedule with $D_3 = 6$

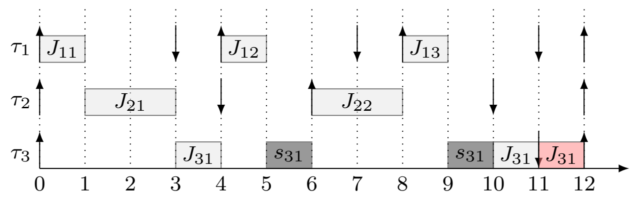


Fig. 4 EDF schedule with $D_3 = 11$

Proof The system is composed of three synchronous periodic tasks τ_1, τ_2 and τ_3 whose characteristics are synthesized in Table 3. The CRPD parameter for each task is equal to 1. Over the hyperperiod, τ_1 issues three jobs J_{11}, J_{12} and J_{13}, τ_2 releases two, J_{21} and J_{22}, and τ_3 only one, J_{31}.

Figure 5 depicts the schedule constructed by EDF over the hyperperiod. All deadlines are met and the system is schedulable.

However, if we increase T_2 by one unit of time, then J_{22} experiences one preemption and misses its deadline, as depicted in Figure 6. As a consequence, the system is not schedulable anymore. Indeed, as τ_2'period is increased, J_{22}'s release date is postponed. So J_{22} has not finished its execution when J_{13} is released. As $d_{13} < d_{22}$, EDF chooses to preempt J_{22} to execute J_{13}. The incurred CRPD causes J_{22} to miss its deadline.

So, EDF is no longer sustainable with regard to the period parameter when preemption delays are considered. □

Theorem 4 RM, DM *and* EDF *are not sustainable with regard to the preemption delay parameter.*

Tasks $\tau_i(C_i, T_i, D_i, s_i)$	Generated jobs $J_{ij}(r_{ij}, p_{ij}, d_{ij}, s_{ij})$
$\tau_1(2,4,2,1)$	$J_{11}(0,2,2,1)$ $J_{12}(4,2,6,1)$ $J_{13}(8,2,10,1)$
$\tau_2(2,6,4,1)$	$J_{21}(0,2,4,1)$ $J_{22}(6,2,10,1)$
$\tau_3(1,12,12,1)$	$J_{31}(0,1,12,1)$

Table 3 Task and job characteristics for the proof of Theorem 3

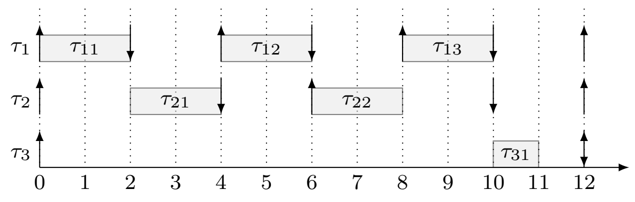


Fig. 5 EDF schedule with $T_2 = 6$

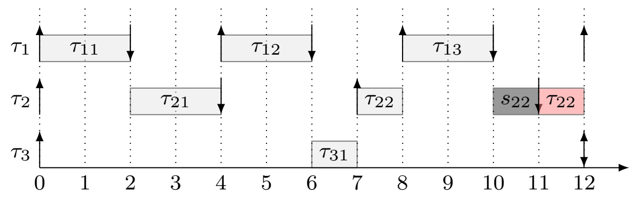


Fig. 6 EDF schedule with $T_2 = 7$

Proof The system is composed of the four synchronous periodic tasks used for the proof of Theorem 1 (see Table 1) but, now, with a preemption delay parameter which is equal to 1 for the four tasks.

Figure 7 depicts the schedule constructed by either RM, DM or EDF over the hyperperiod. All deadlines are met and the system is schedulable.

However, when J_{31}'s preemption delay is reduced to 0.6, the taskset becomes unschedulable. Indeed, as depicted in Figure 8, because of J_{31}'s lesser preemption delay overhead, J_{41} can begin its execution at time 7.6 and, as a result, is preempted by J_{13} at instant 8. As a consequence, J_{41} experiences a total preemption overhead of 1.6 and misses its deadline.

So, RM, DM and EDF are not sustainable with regard to the preemption delay parameter. □

Corollary 1 RM*,* DM *and* EDF *are not sustainable when preemption delays are considered.*

Proof To prove that a scheduling policy is not sustainable it is sufficient to show that it is not sustainable with regard to one of the criteria listed in Definition 1. As RM, DM and EDF are not sustainable with regard to either

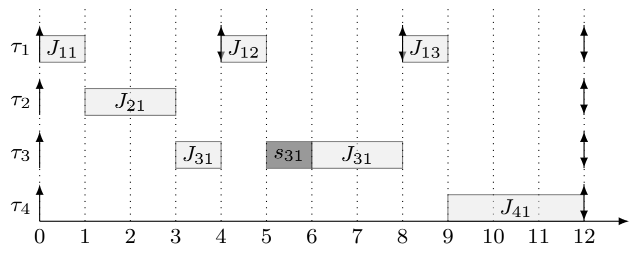


Fig. 7 Schedule with $s_{31} = 1$ for RM, DM and EDF

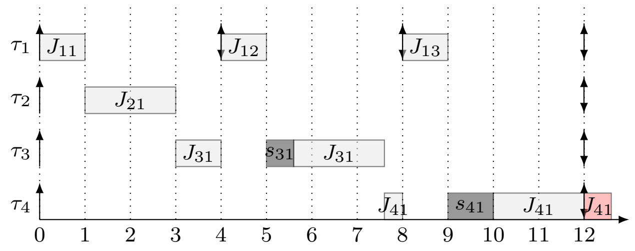


Fig. 8 Schedule with $s_{31} = 0.6$ for RM, DM and EDF

execution times (Theorem 1), deadlines (Theorem 2) or preemption delays (Theorem 4), then they are not sustainable. □

4.2 Inexistence of an optimal online scheduling algorithm

In this section, we focus on the problem of finding an *optimal online* algorithm to schedule tasks subjected to preemption delays. In Phavorin et al (2015b), we show that, when preemption delays are considered, neither fixed-task schedulers such as RM or DM nor fixed-job priority ones such as EDF can be optimal. In Phavorin et al (2015a), the non-optimal property is proved for the case of independent non-recurring jobs. We prove next that this non-optimality result actually applies to the more general case of sporadic tasks.

Hereafter, we show that no optimal scheduler exists for *online* scheduling of sporadic tasks subjected to preemption-delays. Indeed, we prove that an optimal algorithm for scheduling a set of sporadic tasks when preemption delays are considered needs to be clairvoyant with regard to release dates. We recall that clairvoyance means that the scheduler has a complete knowledge of the whole system life (all jobs that will be issued and all their parameters). In order to show that online scheduling of sporadic tasks with preemption delays is impossible, we need to define a sporadic taskset τ and our obligation proof is twofold as explained in Fisher et al (2010):

1. an instance of τ cannot be scheduled by any online scheduler whereas a clairvoyant optimal algorithm can define a feasible schedule,
2. all possible instances of the taskset τ are feasible.

τ_i	C_i	T_i	D_i	s_i
τ_1	5	T	13	2
τ_2	4	T	6	1
τ_3	1	T	3	1

Table 4 Sporadic taskset τ used for the proof of Theorem 5

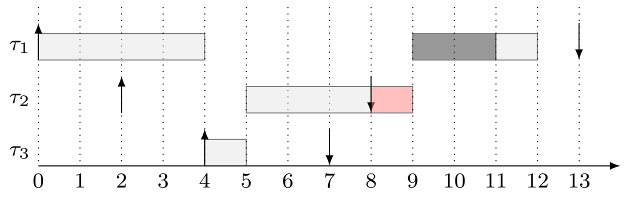


Fig. 9 Schedule constructed by the online algorithm for Theorem 5 - Case 1

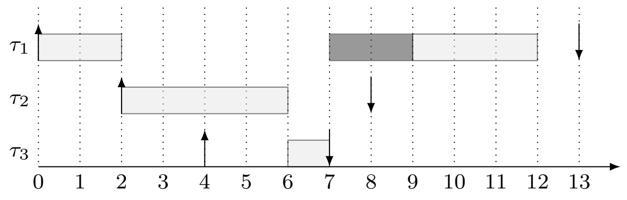


Fig. 10 Feasible schedule for Theorem 5 - Case 1

In Phavorin et al (2015a), we exhibited a counter-example using three jobs $J_1(0,5,12,1)$, $J_2(4,5,6,1)$ and $J_3(r_3,1,1,1)$. The release date of J_3 was set by an adversary according to the decision of the online scheduling algorithm at time 4 (i.e. J_2's release). But this counter-example cannot be used to prove that optimal online scheduling of sporadic tasks accounting for preemption delays is impossible. Suppose that J_1, J_2 and J_3 are each the first job of a sporadic task. To comply with our obligation proof, we have to show that all instances of the sporadic taskset are feasible. Consider the instance for which J_3 is released at time 6. In that case, J_2 is necessarily preempted once and misses its deadline. So, no valid schedule can be constructed. As a result, this sporadic taskset is not feasible. Therefore, the proof presented in Phavorin et al (2015a) for independent jobs cannot be directly extended to the sporadic task case.

For the proof in this paper, we will consider the sporadic taskset presented in Table 4. The three tasks have the same period T which can be set as large as possible (for example $T = \infty$). Thus, we can limit our examination to the first job generated by each task. To prove that no online scheduler can be optimal for the preemption delay-aware scheduling problem, we first have to show that an instance of the sporadic taskset cannot be scheduled by any online algorithm whereas a feasible schedule can be defined by a clairvoyant algorithm (Theorem 5). In a second time, we have to prove that the sporadic tasks is actually feasible, i.e. a feasible schedule can be constructed for all possible instances of the taskset (Lemma 1 and Theorem 6).

Theorem 5 *Optimal online scheduling of sporadic tasks accounting for preemption delays is impossible.*

Proof We show that there is an instance of Taskset τ, defined in Table 4, that cannot be scheduled by any online algorithm whereas a feasible schedule can be defined by a clairvoyant algorithm. The proof is based on a competitive analysis using a clairvoyant adversary. At some instant t, the online algorithm has to take a scheduling decision, i.e. to choose one job to be executed instead of another one. Depending on this choice, the adversary modifies the release date of a future job, such that the online algorithm is not able to schedule the whole system without missing a deadline. On the other hand, the adversary can find a feasible schedule by taking at t another decision than the one of the online algorithm. Then, we consider the case where the online algorithm takes the opposite decision at t, and the adversary proposes another release date to make the system not schedulable once again with respect to the online algorithm.

We consider the instance of τ (i.e. the taskset given in Table 4) defined by the following task offsets: $o_1 = 0$ and $o_2 = 2$. The offline adversary will release the third sporadic task τ_3 either at time $o_3 = 4$ or $o_3 = 8$, depending on the scheduling decisions taken by the online algorithm. Any online scheduling algorithm has to take a scheduling decision at time 2, i.e. at τ_2's release. It can either choose to continue to execute Task τ_1 or choose to preempt it and switch to Task τ_2:

- Case 1: τ_1 continues to execute.
  In this case, the competitor chooses to release τ_3 at time 4. As τ_1 is still executed for some amount of time after time 2, τ_2 cannot complete its execution without a deadline miss: if τ_2 is executed prior to τ_3, then a preemption will necessarily occur in order for τ_3 to meet its deadline. The only solution to avoid this preemption, is to start τ_2 after τ_3's execution. In both cases, τ_2 will miss its deadline. The second case is depicted in Figure 9. So the online algorithm fails to schedule this instance of the system τ. However, as shown in Figure 10, a feasible schedule exists.
- Case 2: τ_2 is chosen to be executed.
  In this case, the adversary chooses to release τ_3 at time 8. As a consequence, τ_1 will necessarily experience two preemptions and as $C_1 + 2 \times s_1 + C_2 + C_3 = 14$ then the deadline at time 13 will be missed as shown in Figure 11. But once more, a feasible schedule exists as depicted in Figure 12.

Note that we should also consider the cases in which the online algorithm insert idle times even if there is either τ_2 or τ_3 ready to be executed. But, inserting idle times means that the processor demand will be higher for the rest of the time interval. As a consequence, the system will still be unschedulable.

To complete the proof, we have to prove that Taskset τ is feasible. This is achieved using Lemma 1 and Theorem 6. □

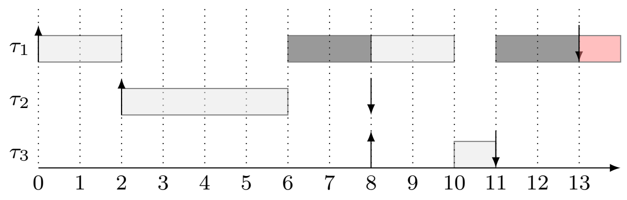


Fig. 11 Schedule constructed by the online algorithm for Theorem 5 - Case 2

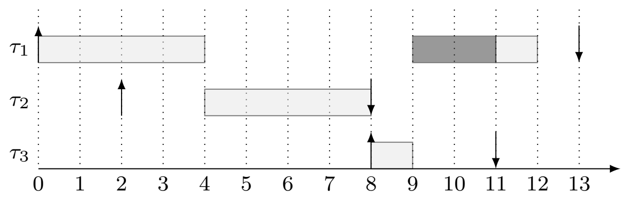


Fig. 12 Feasible schedule for Theorem 5 - Case 2

Before proving that taskset τ is feasible, we first give two direct results of Theorem 5:

Corollary 2 *Optimal online scheduling of a set of jobs accounting for preemption delays is impossible.*

Corollary 3 *Optimal online scheduling of asynchronously released periodic tasks accounting for preemption delays is impossible.*

Note that for *synchronously-released periodic tasks*, the existence of an optimal online scheduler is still an open problem.

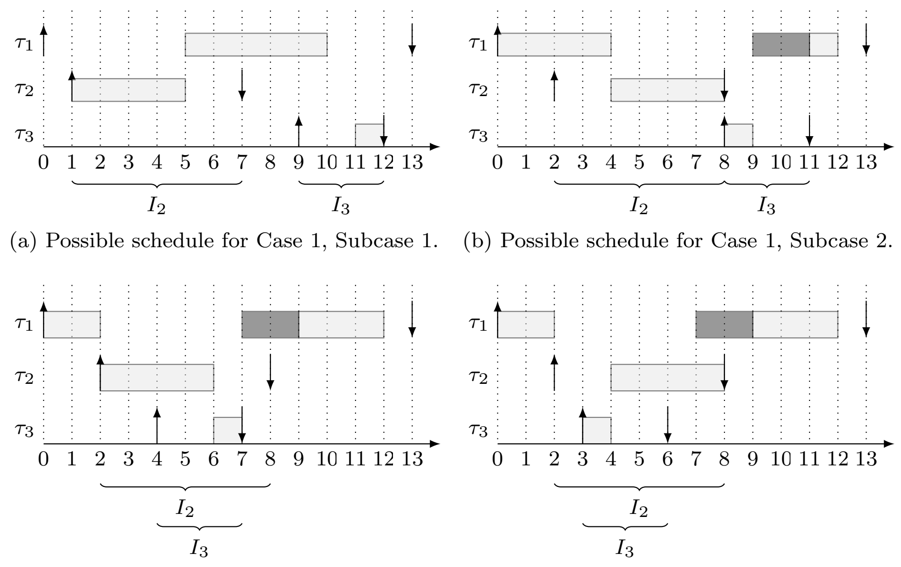
We now prove that all instances of Taskset τ are feasible, i.e. τ is feasible.

Lemma 1 *A necessary condition for τ, defined in Table 4, to be unschedulable is that one task suffer from the interference of the other two. By inter-task interference we mean that their scheduling windows (i.e. the time interval $[o_i, o_i + D_i)$) are interleaved.*

Proof To prove Lemma 1, we consider two cases.

First, we consider the case where a task is executing without any interference from another task (i.e. the scheduling windows of the three tasks are disjoined). For each task, the WCET is smaller than the relative deadline, and as there is no inter-task interference, there can be no preemption and so no additional delay. As a result, each task can be scheduled without missing a deadline.

We now deal with the case where only two tasks from τ interfere together, i.e. the scheduling windows of two tasks from τ are overlapped whereas the scheduling window of the third task is disjoined from the other two. We consider an EDF scheduler and we show that all deadlines will be met:



(a) Possible schedule for Case 1, Subcase 1. (b) Possible schedule for Case 1, Subcase 2.

(c) Possible schedules for Case 2.

Fig. 13 Different cases for the proof of Theorem 6.

- Case 1: both tasks will be executed in sequence, i.e. without any preemption, if:
  - the second released task has a deadline greater that the one of the first released task.
  - the second task is released after the first task has finished its execution.
- Case 2: the second released task preempts the first released task which has not yet finished its execution. As a consequence the first task will have to pay a preemption delay. The second task does not suffer from any preemption and meets its deadline. For the first task we consider the following three cases:
  - τ_1 and τ_2: the response time for τ_1 is equal to $C_1 + s_1 + C_2 = 11 \leq D_1 = 13$.
  - τ_1 and τ_3: the response time for τ_1 is equal to $C_1 + s_1 + C_3 = 8 \leq D_1 = 13$.
  - τ_2 and τ_3: the response time for τ_2 is equal to $C_2 + s_2 + C_3 = 6 \leq D_2 = 6$.

  In all cases, no deadline is missed.

So EDF can construct a feasible schedule when only two tasks are interfering with each other. □

Theorem 6 *Sporadic taskset τ, defined in Table 4, is feasible.*

Proof As a result of Lemma 1, the feasibility study can be limited to a time interval of length D_1 as the three tasks have to interfere with each other in order to have a possible deadline miss. We denote with I_2 (respectively I_3)

the scheduling window of Task τ_2 (resp. τ_3), i.e. the time interval of length D_2 (resp. D_3) in which τ_2 (resp. τ_3) can be scheduled: $I_2 = [o_2, o_2 + D_2]$ (resp. $I_3 = [o_3, o_3 + D_3]$). For the feasibility proof, we have to consider the following cases:

- Case 1: I_2 and I_3 are not overlapped. Without loss of generality, we assume that τ_2 is executed before τ_3. Then we consider the following two sub-cases:
  - Sub-case 1: I_2 and I_3 are separated by at least one time unit: we have $o_3 \geq o_2 + D_2 + 1 \Rightarrow o_3 + D_3 - o_2 \geq D_2 + D_3 + 1$. Executing τ_2 at the start of its interval I_2 and τ_3 at the end of I_3 leaves at least five idle time units in the middle: $(o_3 + D_3 - C_3) - (o_2 + C_2) \geq D_2 + D_3 + 1 - C_3 - C_2 = 5$. So τ_1 can be executed between τ_2 and τ_3 without any preemption as depicted in Figure 13(a).
  - Sub-case 2: I_2 and I_3 are separated by strictly less than one time unit: we have $o_3 - (o_2 + D_2) < 1 \Rightarrow o_3 + D_3 - o_2 \geq D_2 + D_3$. Executing τ_2 at the end of its interval I_2 and τ_3 at the start of I_3 leaves at least a cumulative length of seven idle time units at the beginning and at the end of the studied interval of length D_1: $D_1 - (o_3 + C_3 - (o_2 + D_2 - C_2)) > D_1 - C_3 - C_2 - 1 = 7$. So τ_1 can be executed in two parts, i.e. with one preemption, as $C_1 + s_1 = 7$, as depicted in Figure 13(b).
- Case 2: I_2 and I_3 are overlapped. We can always schedule τ_2 and τ_3 such that none of them preempt the other one. We execute τ_2 and τ_3 contiguously. So in the interval of length D_1, only five contiguous time units are required by τ_2 and τ_3 leaving eight idle time units in two parts for τ_1. As $C_1 + s_1 = 7 < 8$ we can once more construct a valid schedule in all cases as depicted in Figure 13(c).

So Taskset τ is feasible as we can construct a schedule meeting all deadlines for every case. □

5 The *Offline* Scheduling Problem

In the previous section, we showed that no online scheduler can be optimal for the preemption delay-aware scheduling problem. So, in this section, we focus on the offline scheduling problem.

We first recall some general results on the preemption delay-aware scheduling problem given in Phavorin et al (2015b). Note that in Phavorin et al (2015b), we referred to the CRPD only. But as CRPDs are only a special case of preemption delays, all results are still valid in the general case.

Theorem 7 *(Phavorin et al (2015b)) Scheduling with preemption delays is* NP*-hard in the strong sense.*

As a result there exists no scheduling algorithm to solve the preemption delay-aware scheduling problem which takes a scheduling decision (choice of the next processed job) in a polynomial amount of time unless P = NP.

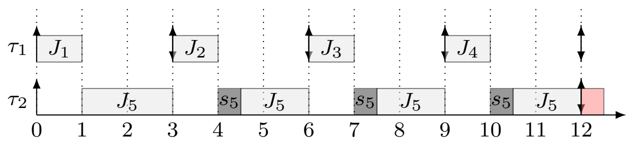


Fig. 14 RM/EDF schedule for tasks τ_1(1,3,3,0.25) and τ_2(7,12,12,0.5)

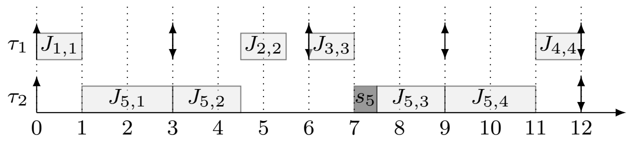


Fig. 15 Schedule produced by the offline approach for tasks τ_1(1,3,3,0.25) and τ_2(7,12,12,0.5)

Actually, the problem is still NP-hard in the weak sense even if there are only two distinct releases and deadlines and a preemption delay of one unit of time as proved in Phavorin et al (2015c). Note that an optimal algorithm exists in some special cases: for a set of jobs with either equal releases dates or equal deadlines, or when releases dates and deadlines are similarly ordered, EDF does not generate any preemption and as a result is an optimal scheduling algorithm in these cases.

As no scheduling algorithm running in polynomial time can be optimal for the preemption delay-aware scheduling problem unless P = NP, we focus on finding an offline solution using Mixed-Integer Linear Programming. We deal next with a finite set of n independent jobs noted $J_i(r_i, p_i, d_i, s_i)$. They are defined by a release date r_i, a worst-case execution time p_i, a deadline d_i and an upper bound on the preemption delay s_i. The objective is to *compute a preemptive schedule that minimizes the overall preemption delay* in order to reduce the worst-case workload. When dealing with synchronous periodic tasks, which is often the case in real-time scheduling, we consider the jobs generated by the different tasks over their hyperperiod. Consider for example two synchronous periodic tasks $\tau_1(1, 3, 3, 0.25)$ and $\tau_2(7, 12, 12, 0.5)$. As depicted in Figure 14, τ_1 generates four jobs (J_1, J_2, J_3 and J_4) over the hyperperiod of 12, whereas τ_2 issues only one job (J_5). As shown in Figure 14, this taskset is not schedulable under RM and EDF assuming that the ties are broken using task indexes. However, a feasible schedule exists as depicted in Figure 15. The offline approach introduced here allows to compute a schedule for this example as detailed in Section 5.3.

5.1 Feasible schedule property

We consider the schedule as a set of slices (represented by their index) $S = \{1, \ldots, m\}$ delimited by subsequent job release dates or deadlines. For every slice $k \in S$, b_k (respectively e_k) denotes the slice starting time (resp. ending time). The first slice begins by the earliest job release date whereas Slice m ends by the latest job deadline. In the example depicted in Figure 15, the schedule is made up of four slices: $[r_1, d_1) = [0, 3)$, $[r_2, d_2) = [3, 6)$, $[r_3, d_3) = [6, 9)$ and $[r_4, d_4) = [9, 12)$. The set of jobs that can be executed in a slice j is denoted by $\mathcal{J}_j, 1 \leq j \leq m$ where $\mathcal{J}_j = \{J_k | r_k < e_j \wedge d_k > b_j\}$. In any feasible schedule, each job must be scheduled in a time interval delimited by its release date and its deadline. We note S_i the set of slices in which Job J_i can be executed: $S_i = \{k | b_k < d_i \wedge e_k > r_i\}$. Note that a slice in S_i cannot start before J_i's release date or end after J_i's deadline as slices are delimited by subsequent job release dates or deadlines. S_i^1 is the first slice of S_i and $\bar{S}_i = S_i \setminus S_i^1$ the set of slices for Job J_i except the first one.

Our approach uses the following property:

Property 1 *There exists a feasible schedule in which a job resumes at most once in every slice.*

Proof Suppose we have a feasible schedule $\mathcal{S}$ in which a job J_i resumes twice in a slice j. We note $J_{i,j}^1$ and $J_{i,j}^2$ the two non-contiguous subjobs of τ_i. Consider Schedule $\mathcal{S}'$ in which, after some subjob permutations, $J_{i,j}^1$ and $J_{i,j}^2$ have been put consecutively. Such a reordering cannot increase the occupied time of Slice j as subjobs have only been reordered and not split (i.e. preempted), so no additional preemption delay is introduced. Moreover, having $J_{i,j}^1$ and $J_{i,j}^2$ now contiguous actually suppresses a preemption delay. Finally, the reordering does not jeopardize schedulability as a job deadline can only occur at the end of a slice. So, if $\mathcal{S}$ is a valid schedule, then $\mathcal{S}'$ is still one.

By repeating these permutations to every job in every slice we can get an optimal schedule in which a job resumes at most once in every slice. □

A direct consequence of Property 1 is that every job executed in a slice can pay at most one preemption delay. Thus, the previous property limits the number of schedule patterns to be considered in order to define an optimal offline schedule in which all job deadlines are met.

5.2 Mathematical program

We define a mixed-integer linear program (MILP) to solve the problem of scheduling with preemption delays. The objective function is to *minimize the overall preemption delay among feasible schedules.*

Jobs will be scheduled in slices and we call a job-piece the job part executed in a given slice. $J_{i,j}$ denotes the job-piece of J_i executed in slice $j \in S_i$. For example, as shown in Figure 15, J_5 is made up of four job-pieces: $J_{5,1}$ in $[0, 3)$, $J_{5,2}$ in $[3, 6)$, $J_{5,3}$ in $[6, 9)$ and $J_{5,4}$ in $[9, 12)$.

The main variables of our MILP formulation are:

- $t_{i,j} \in \mathbb{R}$, corresponding to the starting time of Job-piece $J_{i,j}$,
- $p_{i,j} \in \mathbb{R}$, standing for the execution requirement of Job-piece $J_{i,j}$,

Notation	Type	Description
		Input data
S	Set	Set of slice indexes
S_i	Set	Slice indexes of J_i
$\bar{S}_i$	Set	Slices of J_i except the first one
S_i^1	Set	Index of the first slice of J_i
$\mathcal{J}_j$	Set	Jobs in Slice j
p_i	real	Job J_i's processing time
s_i	real	Job J_i's preemption delay
b_j	real	starting time of Slice j
e_j	real	ending time of Slice j
		Output variables
$t_{i,j}$	real	starting time of Job-piece $J_{i,j}$
$p_{i,j}$	real	processing time of Job-piece $J_{i,j}$
$\Delta_{i,j}$	binary	preemption delay incurred by $J_{i,j}$
		Internal variables
$a_{i,j}$	binary	condition $p_{i,j} > 0$
$a'_{i,j}$	binary	condition $\sum p_{i,k} > 0$
$b_{i,j}$	binary	condition $t_{i,j} > t_{i,j-1} + p_{i,j-1} + s_i\Delta_{i,j-1}$
$y_{i,k,j}$	binary	job-piece disjunctive constraints

Table 5 Data and variables for the MILP

– $\Delta_{i,j} \in \{0, 1\}$, indicating if Job-piece $J_{i,j}$ resumes in Slice j after a preemption.

All notations are summarized in Table 5. The main difference with the approach proposed in Phavorin et al (2015a) is that $p_{i,j}$ corresponds not only to the part of the job WCET executed in Slice j but also to some potential delay due to a preemption. So the preemption delay is not bound to be executed in exactly one slice anymore.

Using these notations, the objective function can be written as:

$$\min \sum_{i=1}^{n} \sum_{j \in S_i} s_i \Delta_{i,j}$$

Note that the schedule computed by the offline approach is not necessarily work-conserving.

Equations 1 to 16 define the constraints for the mixed-integer linear program. Those constraints are divided in different categories presented hereafter.

5.2.1 Processing time constraints

The first set of constraints ensures that all job-pieces of a job put together execute for exactly the execution time of that job plus the total delay due to all the preemptions the job experiences during its execution:

$$\sum_{j \in S_i} p_{i,j} = p_i + \sum_{j \in S_i} s_i \Delta_{i,j} \qquad 1 \le i \le n \qquad (1)$$

5.2.2 Slice constraints

All job-pieces $J_{i,j}$ are executed inside Slice j, i.e. start and complete their execution in the interval $[b_j, e_j)$. An arbitrary small value ϵ is used to forbid a job-piece to start at time e_j.

$$t_{i,j} + p_{i,j} \le e_j \qquad 1 \le i \le n, j \in S_i \tag{2}$$

$$t_{i,j} \ge b_j \qquad 1 \le i \le n, j \in S_i \tag{3}$$

$$t_{i,j} \le e_j - \epsilon \qquad 1 \le i \le n, j \in S_i \tag{4}$$

Job-pieces executed inside a given slice j do not exceed the interval size:

$$\sum_{i \in \mathcal{J}_j} p_{i,j} \le e_j - b_j \qquad j \in S \tag{5}$$

5.2.3 Job-piece disjunctive constraints

Inside every slice, two job-pieces cannot be executed simultaneously. In Slice j, for every pair of job-pieces $J_{i,j}$ and $J_{k,j}$ we have either:

$$t_{i,j} + p_{i,j} \le t_{k,j}$$

or

$$t_{k,j} + p_{k,j} \le t_{i,j}$$

The previous disjunctive constraints can be linearized using a big value M and a binary variable $y_{i,k,j}$, $i < k$, set to 1 by the solver if $J_{i,j}$ is executed before $J_{k,j}$ in Slice j, 0 otherwise:

$$t_{i,j} + p_{i,j} \le t_{k,j} + (1 - y_{i,k,j}) \times M \qquad j \in S_i \cap S_k \tag{6}$$

$$t_{k,j} + p_{k,j} \le t_{i,j} + y_{i,k,j} \times M \qquad j \in S_i \cap S_k \tag{7}$$

5.2.4 Preemption penalty constraints

The binary variable $\Delta_{i,j}$ is set to 1 if Job-piece $J_{i,j}$ is subjected to a preemption delay in Slice j, to 0 otherwise.

For the first slice of every job, there cannot be any preemption:

$$\Delta_{i,j} = 0 \qquad 1 \le i \le n, j \in S_i^1 \tag{8}$$

In every other slice j, a preemption is paid by Job J_i (i.e. $\Delta_{i,j} = 1$) if and only if:

1. J_i is executed in Slice j: $p_{i,j} > 0$
2. J_i has already started its execution previous to j: $\sum_{k<j} p_{i,k} > 0$
3. J_i'executions in Slices $j-1$ and j are not contiguous:
   $t_{ij} > t_{i,j-1} + p_{i,j-1}$

task	job	slice	$t_{i,j}$	$p_{i,j}$	$\Delta_{i,j}$
τ_1	J_1	1	0	1	0
τ_1	J_2	2	4.5	1	0
τ_1	J_3	3	6	1	0
τ_1	J_4	4	11	1	0
τ_2	J_5	1	1	2	0
τ_2	J_5	2	3	1.5	0
τ_2	J_5	2	7	1.5	1
τ_2	J_5	4	9	2	0

Table 6 Output variables computed by the solver

.

In all other scenarios, the solver will always choose to set $\Delta_{i,j} = 0$ to minimize the objective function and thus no preemption delay is paid by the corresponding job-pieces.

The previous three conditions can be linearized by introducing binary variables $a_{i,j}$, $a'_{i,j}$ and $b_{i,j}$:

$$p_{i,j} \leq a_{i,j} \times M \qquad 1 \leq i \leq n, j \in \bar{S}_i \qquad (9)$$

$$\sum_{k \in S_i, k<j} p_{i,k} \leq a'_{i,j} \times M \qquad 1 \leq i \leq n, j \in \bar{S}_i \qquad (10)$$

$$t_{i,j} - (t_{i,j-1} + p_{i,j-1}) \leq b_{i,j} \times M \qquad 1 \leq i \leq n, j \in \bar{S}_i \qquad (11)$$

According to these three new binary variables, $\Delta_{i,j}$ is defined by the logical result of $a_{i,j} \wedge a'_{i,j} \wedge b_{i,j}$ which can be linearized by computing $\Delta_{i,j} = \min(a_{i,j}, a'_{i,j}, b_{i,j})$:

$$\Delta_{i,j} \leq a_{i,j} \qquad 1 \leq i \leq n, j \in \bar{S}_i \qquad (12)$$

$$\Delta_{i,j} \leq a'_{i,j} \qquad 1 \leq i \leq n, j \in \bar{S}_i \qquad (13)$$

$$\Delta_{i,j} \leq b_{i,j} \qquad 1 \leq i \leq n, j \in \bar{S}_i \qquad (14)$$

$$\Delta_{i,j} \geq a_{i,j} + a'_{i,j} + b_{i,j} - 2 \qquad 1 \leq i \leq n, j \in \bar{S}_i \qquad (15)$$

$$\Delta_{i,j} \geq 0 \qquad 1 \leq i \leq n, j \in \bar{S}_i \qquad (16)$$

5.3 Application example

We now detail an application of our offline approach on the two-tasks example introduced beforehand and depicted in Figure 14.

For the MILP, we consider the different jobs $J_i(r_i,p_i,d_i,s_i)$ from τ_1 and τ_2 over the hyperperiod which is equal to 12. On that interval of time, τ_1 issues four jobs $J_1(0,1,3,0.2)$, $J_2(3,1,6,0.2)$, $J_3(6,1,9,0.2)$ and $J_4(9,1,12,0.2)$, whereas τ_2 issues only one job $J_5(0,7,12,0.5)$. Release dates and deadlines define four slices in the schedule: $[0,3)$, $[3,6)$, $[6,9)$ and $[9,12)$. J_1 must be executed in the first slice, J_2 in the second one, J_3 in the third one and J_4 in the last one, whereas J_5 can be executed in all of them. As a result there are eight job-pieces for this example.

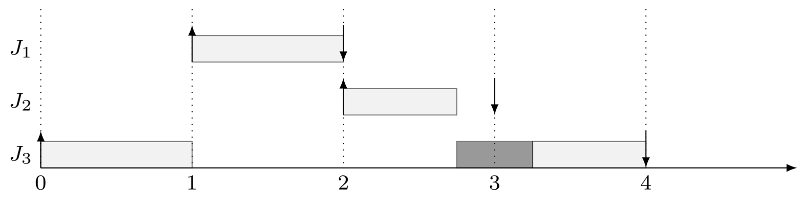


Fig. 16 Example of a feasible system deemed unschedulable by the approach presented in Phavorin et al (2015a). The system is made of three jobs $J_1(1, 1, 2, 0.25)$, $J_2(2, 0.75, 3, 0.25)$ and $J_3(0, 1.75, 4, 0.5)$.

For this example, we set $\epsilon = 0.01$ and $M = 100$. The complete MILP has thirty-seven variables (twenty-four output variables and thirteen internal ones) and seventy constraints. Using the *CPLEX* 12.6.1 solver from IBM, we compute an optimal solution to the problem with a value of 0.5 for the objective function. The output variables computed by the solver are presented in Table 6. The results computed by the solver are interpreted the following way. If $\Delta_{i,j} = 0$ (as for J_1 in Slice 2), then J_i does not pay any preemption delay in Slice j and $p_{i,j}$ corresponds exactly to the normal execution time for J_i in j. If $\Delta_{i,j} = 1$ (as for J_5 in Slice 3), then J_i pays first a preemption delay of length s_i in j and then executes normally for $p_{i,j} - s_i$. Note that if $p_{i,j} < s_i$, then the rest of the preemption delay is paid in the next slice where J_i is executed. The corresponding schedule is depicted in Figure 15. Note that the schedule constructed by the offline approach is not necessarily work-conserving: for this example, as depicted in Figure 15, the processor is left idle at time 5.5 even if Job J_5 is ready to be executed.

5.4 Comparison with Phavorin et al (2015a)

The approach presented in this paper is a modified version of the one introduced in Phavorin et al (2015a).

In Phavorin et al (2015a), a transformed scheduling problem is considered in order to simplify the mathematical model: every job has an execution time $p'_i = p_i - s_i$ and a preemption delay s_i is always paid when the job starts its execution for the first time. This transformation is only valid when $\forall J_i : p_i \geq s_i$ which is obviously true in real-life. However, in this paper, we adopt the same experimental plan as in Altmeyer et al (2012); Lunniss et al (2013, 2014) in order to have a fair comparative point. The s_i parameters correspond to CRPDs and are derived from ECB and UCB sets which are randomly generated for each task (and so job) has a function of the total cache utilization. As a result WCETs and CRPDs are decoupled. In some cases, in particular for tasksets with small processor utilization and large cache utilization, CRPDs may be greater than WCETs.

Moreover, in order to simplify the mathematical program, the approach presented in Phavorin et al (2015a) only considers schedules where the preemption delay paid by a job when resuming its execution in a slice fits in the slice boundaries. For each job-piece, a variable $p'_{i,j}$ was computed, standing for the execution time of Job J_i in slice j without accounting for any CRPD. As a result, potential CRPDs in every slice had to be accounted for when dealing with the MILP constraints. For example, Slice constraints 2 and 5 were written as:

$$t_{i,j} + p'_{i,j} + s_i \times \Delta_{i,j} \leq e_j \qquad\qquad 1 \leq i \leq n, j \in S_i$$

and

$$\sum_{i \in \mathcal{J}_j} p'_{i,j} + s_i \times \Delta_{i,j} \leq e_j - b_j \qquad\qquad j \in S$$

In other words, it means that a preemption delay cannot be spread over two consecutive slices. In most cases, this assumption holds. However, for a nearly full-loaded processor and potentially large preemption delays, there may be systems for which the only valid schedules do not respect this assumption. For example, as depicted in Figure 16, the only schedule for the system made of Jobs $J_1(1, 1, 2, 0.25)$, $J_2(2, 0.75, 3, 0.25)$ and $J_3(0, 1.75, 4, 0.5)$ does not respect the assumption as the preemption delay paid by J_3 when it resumes its execution in the second slice has to be spread over the third slice. So the solution proposed in Phavorin et al (2015a) cannot find a valid schedule in this case and the system is deemed unschedulable since preemption delays are not allowed to be spread over two slices. The MILP presented here overcomes this problem as potential CRPDs are now implicitly accounted for in job-piece execution times $p_{i,j}$. Preemption delays appear explicitly only in the processing time constraints to guarantee that a job is executed for exactly its WCET plus the total delay due to all the preemptions the job experiences during its execution. As a result, our new approach is able to schedule systems as the one depicted in Figure 16.

An evaluation of our new offline approach in terms of solving time is given in Section 6.

5.5 MILP generalization to enhanced preemption delay models

The efficiency of the offline approach in comparison with other scheduling solutions is very dependent on the adopted preemption delay model. Our offline approach assumes a preemption delay bound depending only on the preempted task. For this preemption delay model, our solution is optimal, as it constructs a valid schedule whenever it is possible.

But under this model, the preemption delay bound can be very pessimistic as depicted in Figure 17. The system is made of two jobs $J_1(1, 1, 2)$ and $J_2(0, 2, 3)$ which can be schedulable only if no preemption delay is paid as the processor is fully loaded. Instead of a CRPD parameter, we consider here Evicting Cache Blocks (ECBs) and Useful Cache Blocks (UCBs). ECBs and UCBs are represented by the indexes of the cache sets to which they are mapped. J_1 has two ECBs: ECB $= \{0, 1\}$. During its execution, J_1 might evict any memory

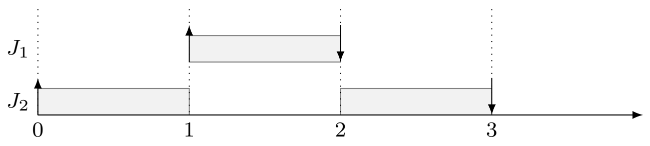


Fig. 17 Example of a system unschedulable by the offline approach because of the pessimism of the preemption delay model. The system is made of two jobs $J_1(1, 1, 2, \text{ECB} = \{0, 1\})$ and $J_2(0, 2, 3, \text{UCB} = \{2, 3\})$.

block (potentially used by another task) stored in Cache sets 0 and 1. J_2 has two UCBs, UCB $= \{2, 3\}$. If preempted, J_2 might have to reload two memory blocks if preempting tasks access Cache sets 2 and 3 during their execution. For this example, we only consider CRPDs and assume all other preemption delays to be either negligible or already accounted for in the WCETs. As J_2 has two UCBs, s_2 is not null (as in the worst-case two memory blocks might have to be reloaded) and as a consequence the MILP fails to construct a valid schedule for this system. However, as the ECBs of J_1 do not map to the same cache locations as the UCBs of J_2, J_2 will not need to reload any useful memory block when resuming its execution after being preempted by J_1. So no CRPD is actually incurred and a feasible schedule can be constructed. As a result, considering a more precise preemption delay model would allow to find a valid schedule for this system, as no preemption cost would occur.

To handle those cases, the offline approach has to be modified to consider a more accurate preemption delay bound $s_{i,j}$, depending both on the preempted job and the preempting ones. However, designing such an MILP is a difficult matter. For example, the CRPD paid by a preempted job J_i does not only depend on the damage to the cache done by the preempting job J_j, but also on the damage due to all jobs J_k that execute while J_i being preempted. One solution is to use boolean variables to represent whether Job J_k has been executed after Job J_i's preemption in Slice j' and before J_i resumes its execution in Slice j. As a consequence, the MILP size increases very fast. Such a complexity would allow to deal only with a small number of jobs and few time slices and thus would not be suitable in practice.

6 Experiments

In this section, we evaluate the effectiveness of our offline solution, called hereafter OFF, in comparison with existing scheduling policies when CRPDs are considered, using synthetically generated tasksets. For the experiments, we only focus on CRPDs since they represent the most penalizing part of the preemption delays. We compare the schedulability of RM, EDF and OFF when varying different key parameters such as the processor utilization or the cache utilization. For the CRPD parameter, we consider successively the UCB-only, the UCB-union and the ECB-union approaches to upper bound the CRPD for each task under RM (see Altmeyer et al (2012)) and EDF (see Lunniss et al (2013)).

6.1 Experimental settings

For the experiments, we consider our offline approach alongside with RM and EDF and we follow the approach adopted in Lunniss et al (2013) for task and CRPD-related parameter generation. We deal with tasksets made of only four tasks generating at most two hundred jobs over the hyperperiod, with task periods ranging from 1ms to 10ms. These restrictions, compared to the work in Lunniss et al (2013), are made to contain the explosion of the MILP solving time, which tends to increase exponentially with the size of the inputs (i.e. numbers of jobs and slices). The different taskset parameters are generated using the following inputs:

– the taskset processor utilization U;
– the taskset cache utilization CU (corresponding to the proportion of the cache used by all tasks put together), set by default to 4 ($CU > 1$ means that all task memory blocks cannot fit together into the cache);
– the cache size in number of sets CS (which is equal to the number of lines as we deal only with DM cache), set by default to 256;
– the Block Reload Time BRT, set by default to 0.008ms;
– the maximum proportion of UCBs per task, called the re-utilization factor RF, set by default to 30%;
– the approach to bound the CRPD for RM and EDF schedulability analyses: by default we use the UCB-only approach.

Note that the default values have been chosen accordingly to the ones found in the literature (Altmeyer et al (2012); Lunniss et al (2013, 2014)). In particular, hardware settings (cache size, BRT) are chosen to model commonly used embedded configurations based for example on the ARM7 architecture[1]. As for the default value for RF, it corresponds to the values observed for the tasks from the Mälardalen benchmark[2] (Altmeyer et al (2012)). It also fits the values observed for the real-time embedded PapaBench benchmark[3] were the percentage of ECBs per task ranges from 0% for the radio_control task to 30.3% for the altitude_control task (Lunniss et al (2014)).

Each task processor utilization u_i is generated using the UUnifast algorithm (Bini and Buttazzo (2005)). Task periods T_i are randomly chosen in $[1, 10)$ using a uniform distribution. We prefer a uniform distribution rather than a log-uniform one as in Lunniss et al (2013), since the task range is reduced to only one order of magnitude as stated before. Using the previously generated u_i and T_i, each task WCET C_i is computed as $C_i = u_i \cdot T_i$. As we limit our

[1] http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.subset.primecell.system/index.html
[2] http://www.mrtc.mdh.se/projects/wcet/benchmarks.html
[3] https://www.irit.fr/recherches/ARCHI/MARCH/rubrique.php3?id_rubrique=97

experiments to synchronously-released tasks with implicit-deadlines, we set $o_i = 0$ and $D_i = T_i$. In Altmeyer et al (2012), a task cache utilization is generated for each task in order to get the number of ECBs for the task ECB_i. As ECB_i has to be an integer, rounded computation is needed. So, the real cache utilization for the taskset might be slightly different from the one given as an input. To avoid such an issue, we prefer to generate directly each task $\#\text{ECB}_i$ as an integer value with a total sum of $CU \times CS$, using the UUnifast algorithm and a rounding technique similar to the one proposed in Lunniss et al (2012). The cache set index of the first ECB of each task S_i is randomly generated as a uniformly-distributed integer in $[0, CS - 1]$. All task ECBs are placed in a continuous group starting at S_i. Finally, the number of UCBs per task $\#\text{UCB}_i$ is randomly generated in $[0, RF \cdot \#\text{ECB}_i]$ according to a uniform distribution. All task UCBs are also placed in a continuous group starting at cache index S_i. For the sake of simplicity, we will consider hereafter solely the UCB-only approach to bound the CRPD. This approach will be detailed in the next section for RM and EDF scheduling policies. As in Altmeyer et al (2012), CRPD values s_i are not dependent on worst-case execution times C_i. As a consequence, some tasks can have a larger CRPD than their worst-case execution time.

6.2 Evaluated metrics

Through our experiments, we aim at comparing RM, EDF and OFF in terms of schedulability when varying different input parameters. We prefer to focus only on schedulability because preemption-based metrics do not seem to be relevant here as we consider only worst-case scenarios (which is typical in the hard real-time scheduling theory). In our opinion, it would be more adequate to measure the number of preemptions for a taskset or the total overhead produced over the hyperperiod on real-life scenarios (i.e. tasks with average execution times instead of WCETs).

For each experiment, we generate one thousand independent tasksets per processor utilization value. Each time, schedulability is evaluated using one of the three metrics detailed hereafter.

6.2.1 Number of schedulable tasksets

We measure the number of schedulable tasksets for each scheduling approach as a function of the total processor utilization U, all other input parameters being set to their default values.

For RM, we use the modified version of the Response Time Analysis accounting for CRPDs presented in Busquets-Mataix et al (1996a):

$$R_i = C_i + \sum_{\forall j \in \text{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot (C_j + \gamma_{i,j})$$

hp(i) being the set of tasks with priorities higher than the one of τ_i and $\gamma_{i,j}$ a bound on the CRPD experienced by τ_i each time it is preempted by a higher priority task τ_j. $\gamma_{i,j}$ can be computed using for example the UCB-only approach introduced in Lee et al (1998):

$$\gamma_{i,j} = \text{BRT} \cdot \max_{\forall k \in \text{hep}(i) \cap \text{lp}(j)} \{|\text{UCB}_k|\}$$

hep(i) being the set of tasks with priorities higher or equal to the one of τ_i and lp(j) the set of tasks with lower priority than the one of τ_j.

For EDF, we can simply use the sufficient test for periodic tasks with implicit deadlines introduced in Lunniss et al (2013):

$$\sum_{\forall \tau_i} \frac{C_i + \gamma_{D_{\max},i}}{T_i} \leq 1$$

$D_{\max}$ being the largest relative deadline in the taskset. $\gamma_{D_{\max},i}$ can be computed using for example the UCB-only approach adapted for EDF in Lunniss et al (2013):

$$\gamma_{D_{\max},i} = \text{BRT} \cdot \max_{\forall j \in,\ D_{\max} \geq D_j > D_i} \{|\text{UCB}_j|\}$$

Finally for OFF, s_i corresponds to an upper bound on the CRPD paid by Task τ_i each time it resumes after a preemption: $s_i = \text{BRT} \cdot |\text{UCB}_i|$. To evaluate the schedulability of a taskset, we use the CPLEX 12.6.1 solver from IBM to solve the MILP: if the solver finds a solution, then the taskset is deemed schedulable. To further bound the solving time issue, we set a time limit of 10 seconds for the solver. If the time limit is exceeded, then the solver keeps the best current solution (if any).

6.2.2 Weighted Schedulability

To study the impact of parameters other than U, we use the *weighted schedulability* measure introduced in Bastoni et al (2010). For each approach and each value of the chosen varying parameter p, one thousand tasksets are generated for a set Q of equally spaced processor utilization values:

$$Q = \{u | u = k \cdot 0.1, k \in [\![1, 10]\!]\}$$

Then, the results are graphically represented using the weighted schedulability $W_\ell(p)$ which combines the data of all tasksets for every processor utilization value in Q:

$$W_\ell(p) = \frac{\sum_{\forall U \in Q} U \cdot S_\ell(U,p)}{\sum_{\forall U \in Q} U}$$

where $S_\ell(U,p)$ is the binary result of the schedulability test for a taskset with a processor utilization U and a value p for the studied parameter. The Weighted Schedulability measure allows to reduce three-dimensional plots to two dimensions only without having to give the processor utilization a fixed value.

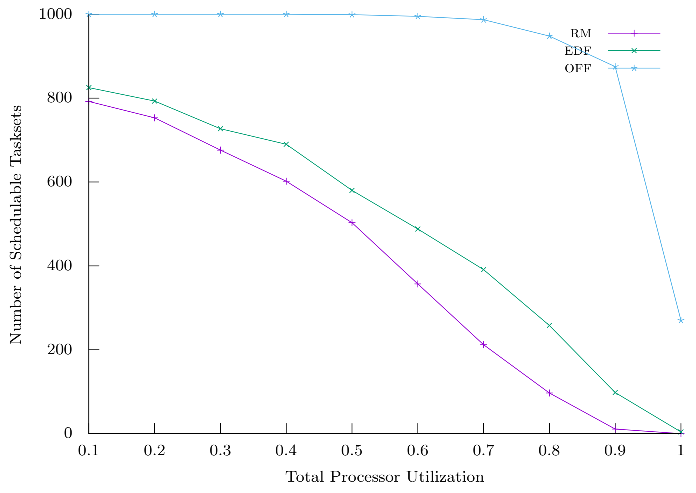


Fig. 18 Number of schedulable tasksets under RM, EDF and OFF as a function of the total processor utilization U.

6.2.3 Speedup Factors

We also study speedup factors in order to quantify the resource cost introduced when considering CRPDs in scheduling. Consider a given taskset deemed schedulable when no CRPD is considered (e.g. $U \leq 1$ for periodic tasks with implicit deadlines). The speedup factor *scf* corresponds to the factor by which the processor speed needs to be increased and the memory access time needs to be decreased for the taskset to be schedulable when CRPDs are considered. *scf* is such that a taskset, deemed schedulable when no CRPD is considered, is schedulable with $\forall \tau_i : C'_i = \frac{C_i}{scf} \wedge \text{BRT}' = \frac{\text{BRT}}{scf}$ when CRPDs are accounted for.

We compute the average speedup factors among all tasksets per experiment point for different input parameters.

6.3 Results

6.3.1 Impact of the Processor Utilization

We first consider how the total processor utilization of the taskset impacts the system schedulability. All input parameters, except for the processor utilization U, are set to their default values, i.e. $CU = 4$, $RF = 30\%$, $CS = 256$ and $\text{BRT} = 0.008$. U is varied from 0.1 to 1.0 onwards by step of 0.1.

As depicted in Figure 18, when the total processor utilization increases, then RM, EDF and OFF experience a decrease in schedulability. For small values of U ($U = 0.1$), RM and EDF successfully schedules about 80% of the tasksets whereas OFF achieves a 100% success. Some tasksets are deemed unschedulable under RM and EDF even for small values of U because of potential large

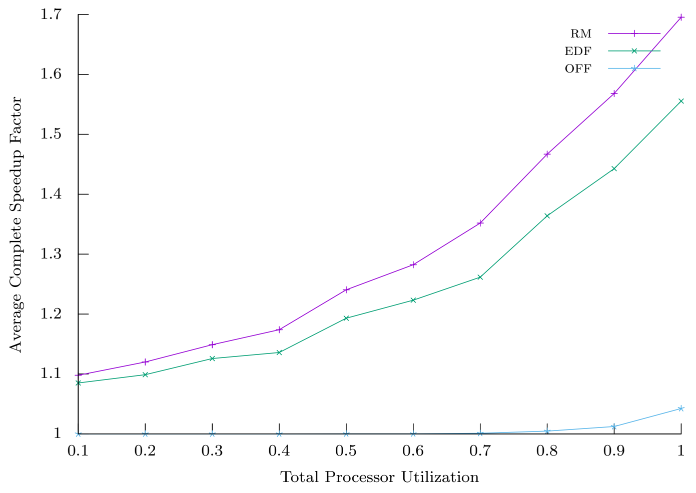


Fig. 19 Average Processor and Memory Speedup Factor for RM, EDF and OFF as a function of the total processor utilization U.

CRPDs: as the CRPD is independent from U for our experiments, some tasksets may experience CRPDs larger than WCETs. For some tasks, these CRPDs may not fit in the task execution window (time interval between the task release time and its deadline) causing deadline miss. For OFF, no such problem occurs as it is more likely to construct a schedule without any preemption. For large values of U ($U = 0.8$), EDF can only schedule 243 tasksets out of one thousand, whereas RM drops under one hundred schedulable tasksets. As shown in Buttazzo (2005), the number of preemptions for RM increases with the processor utilization, causing a higher overall CRPD. For EDF, the number of preemptions does not necessarily increase. But as the processor utilization is higher, there is less free CPU time and in particular task lateness (i.e. the amount of time between the completion date of a task and its deadline) might be reduced in average. So, it is more likely that a preemption delay, which postpones the task completion date, results in a deadline miss. On the contrary, OFF successfully schedule more than 95% of the tasksets for $U = 0.8$. This is because the MILP focuses on reducing the cumulated preemption delay. As a result, the average number of preemptions per job and the average proportion of the processor time occupied by CRPDs remain quite low (under 0.06 preemptions per job and 1.2% of the processor time). For $U = 1$, the only tasksets deemed schedulable are either those which can be scheduled non-preemptively or those for which a schedule can be constructed with zero-cost preemptions, i.e. tasks with zero UCB.

In Figure 19, we depict the average factor by which the CPU speed should be increased and the memory access time be decreased in order to have an unschedulable taskset become schedulable. For large values of U ($U = 0.8$), this speedup factor can increase up to 1.45 *in average* for RM (i.e. a 45% faster

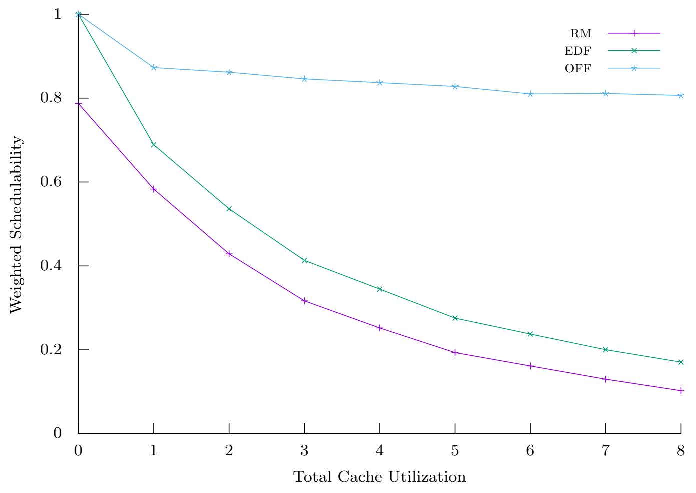


Fig. 20 Weighted Schedulability as a function of CU.

CPU and memory bus) and 1.33 *in average* for EDF (i.e. a 33% faster CPU and memory bus) whereas it remains nearly equal to 1 for OFF. Note that for OFF, the *maximum* speedup factor for any taskset never exceeds 1.83 (for $U = 1.0$) whereas it can exceed 5 for EDF and RM (for $U \geq 0.8$).

6.3.2 Impact of the Cache Utilization and the Cache Reuse

We now focus on the impact of task cache-related parameters on the system schedulability. Input parameters CS and BRT are set to their default values $CS = 256$ and BRT $= 0.008$.

First, we consider the impact of the cache utilization. RF is set to 0.3. CU is varied from 0 to 8 onwards by step of 1 and for each value of CU, we generate one thousand tasksets per processor utilization value in $Q = \{u|u = k \cdot 0.1, k \in [\![1, 10]\!]\}$.

As shown in Figure 20, when increasing the total cache utilization, the weighted schedulability of RM, EDF and OFF decreases as the CRPDs increase. For large CRPDs, it might be impossible even for OFF to construct a feasible schedule when U increases. However, for large values of U, OFF still behaves quite well even for large values of CU: for $CU = 8$, the number of tasksets deemed schedulable by OFF is more important than the number of tasksets deemed schedulable by RM or even EDF for smaller cache utilization values. Note that increasing the number of cache sets CS gives similar results. Indeed, ECBs are generated based on the total cache occupancy $CU \times CS$. Increasing either CU or CS results in a higher number of ECBs (and so of UCBs) in average.

Then, we study the impact of the cache reuse. CU is set to 4. RF is varied from 0.0 to 0.5 onwards by step of 0.05 and for each value of RF, we generate

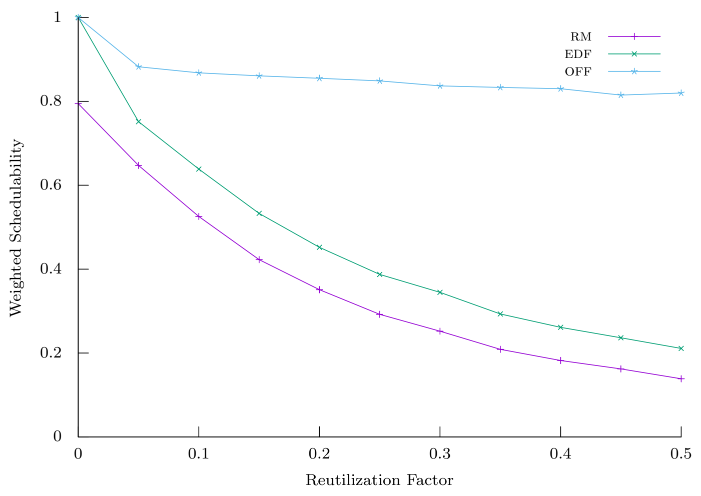


Fig. 21 Weighted Schedulability as a function of RF.

one thousand tasksets per processor utilization value in $Q = \{u | u = k \cdot 0.1, k \in [\![1,10]\!]\}$.

The results are quite similar when varying the reutilization factor, see Figure 21. As for the total processor utilization, increasing RF might result in larger CRPDs and so in potential deadline misses. Indeed, increasing RF means that the maximum proportion of UCBs per task is increased which results in a larger number of UCBs per task in average (as $\#\text{UCB}_i$ is randomly chosen between 0 and $RF \cdot \#\text{ECB}_i$ using a uniform distribution).

6.3.3 Impact of the Block Reload Time

We now focus on the impact of cache parameters on the system schedulability. Input parameters CU and RF are set to their default values $CU = 4$ and $RF = 0.3$.

As stated previously, the impact of the number of cache lines CS is quite similar to the impact of the total cache utilization CU. So, we focus hereafter on the block reload time. CS is set to 256 and BRT is varied from $2^1 = 2\mu s$ to $2^5 = 32\mu s$ onwards by step of power of 2. For each value of BRT, we generate one thousand tasksets per processor utilization value in $Q = \{u | u = k \cdot 0.1, k \in [\![1,10]\!]\}$.

As depicted in Figure 22, when increasing the Block Reload Time BRT, the schedulability of RM, EDF and OFF decreases. As the total cache utilization and the cache size remain constant ($CU = 4$ and $CS = 256$), a higher value of BRT means larger values for the CRPD parameters. In particular, RM and EDF experience a huge loss of schedulability, in particular for small values of U. OFF behaves quite well even for large values of BRT: for $\text{BRT} = 0.032$, there

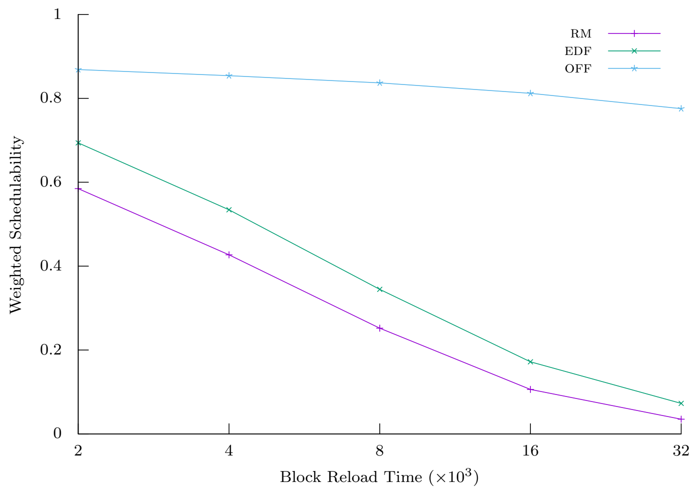


Fig. 22 Weighted Schedulability as a function of BRT.

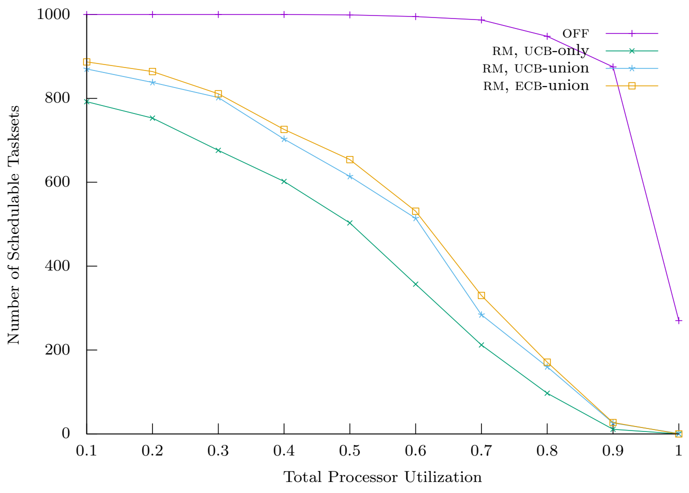
are more tasksets deemed schedulable by OFF than there are for RM or even EDF for BRT $= 0.008$.

6.3.4 Impact of the approach used to bound the CRPD

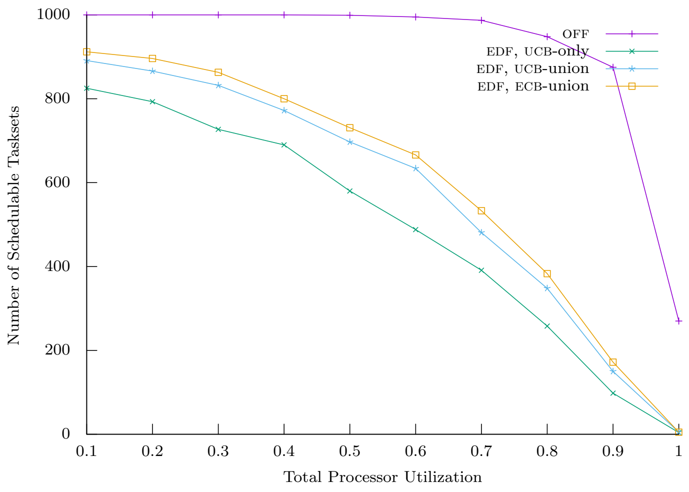
Finally, we consider the influence of the approach used to bound the CRPD on the system schedulability. For RM and EDF schedulability analyses, we consider enhanced CRPD upper bounds which take into account both the impact of the preempted and preempting tasks. As our offline approach uses a task model where the CRPD parameter only depends on the considered task (i.e. the preempted task), it is no longer optimal. So, we evaluate in this section the effectiveness of our offline approach when more precise CRPD upper bounds are used for RM and EDF schedulability analyses. Input parameters CU, CS, RF and BRT are set to their default values $CU = 4$, $CS = 256$, $RF = 0.3$ and BRT $= 0.008$.

We vary U from 0.1 to 1.0 onwards by step of 0.1 and for each value of U, we generate one thousand tasksets. For each taskset, we consider the UCB- and ECB-union approaches for both RM and EDF alongside the UCB-only approach.

As depicted in Figure 23, using enhanced CRPD bound approaches results in a higher number of schedulable tasksets. But, this number remains low in comparison with the number of tasksets deemed schedulable by the offline approach OFF. Actually, only nine tasksets (resp. three) out of all generated tasksets (i.e., ten thousand tasksets) are found schedulable by EDF (resp. DM) using the ECB-union approach and not by OFF (the results are actually similar when using the UCB-union one). The offline approach is no longer optimal but still allows to achieve a better schedulability ratio.



(a) Comparison of RM and OFF.



(b) Comparison of EDF and OFF.

Fig. 23 Number of schedulable tasksets under RM, EDF and OFF as a function of the total processor utilization for several CRPD bound approaches.

So our offline approach clearly outperforms RM and EDF even when enhanced CRPD bounds are considered. Indeed, for large values of the processor utilization, the total cache utilization, the reutilization factor or the Block Reload Time, both RM and EDF experience a huge schedulability loss whereas the offline approach still achieves a high schedulability ratio.

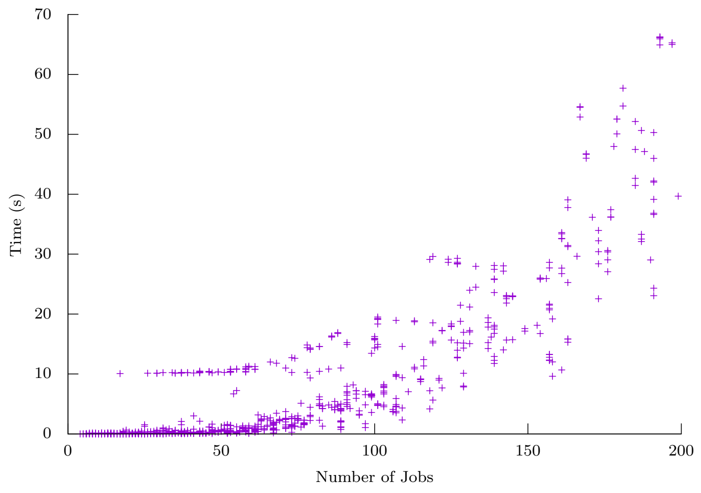


Fig. 24 Evaluation of the solving time for the offline approach OFF as a function of the number of jobs for $U = 0.8$.

6.3.5 Mathematical program solving time

We give here an insight on the solving time for the offline approach. We deal with the tasksets generated when studying the impact of U on the system schedulability (i.e. $CU = 4$, $CS = 256$, $RF = 0.3$, BRT $= 0.008$) for $U = 0.8$.

We measure the time needed to construct the inputs for the MILP and the time needed by the solver to compute a solution for each taskset as a function of the number of jobs. As depicted in Figure 24, when the number of jobs increases, the total time needed for the offline approach OFF tends to explode, rising from less than 10s in average for tasksets with twenty jobs to more than 60s for some tasksets with two hundred jobs. However, the computation time for the offline approach is not strictly correlated with the number of jobs: the time to construct the inputs for the MILP is directly related to the number of jobs and slices but the MILP solving time is highly variable from one instance to another.

7 Conclusion

In this paper, we study the problem of scheduling hard real-time tasks accounting for preemption delays. We first show that three of the most popular online scheduling algorithm, RM, DM and EDF, are no longer sustainable when subject to preemption delays. Moreover, we prove that no online scheduler can be optimal for scheduling sporadic tasks when preemption delays are considered. Because clairvoyance is needed to devise an optimal scheduling algorithm, we focus next on offline scheduling. In that context, we propose an offline solution to the preemption delay-aware scheduling problem using a mixed-integer linear program formulation.

Then, we focus on the impact of cache memory on the system schedulability and evaluate our offline solution alongside with RM and EDF when subjected to CRPDs. In particular, we study how different parameters such as the processor utilization or the cache utilization have an impact on the system schedulability. We show that both RM and EDF experience a huge schedulability loss for large values of the processor utilization, the total cache utilization or the Block Reload Time. On the contrary, our offline approach can still schedule most of the tasksets even for large processor utilizations, cache utilizations or Block Reload Times. When enhanced CRPD bounds are used for RM and EDF, our offline approach still clearly outperforms RM and EDF even if it is no longer optimal.

As real-time system behavior depends on WCET, periods, deadlines but also preemption delays (which are closely related to hardware features such as cache memories), the scheduler should consider all these parameters at the same time in order to improve schedulability. The offline approach proposed here is a step forward. A possible next step would be to devise online heuristic algorithms for the problem of scheduling with preemption delays. Note that, in this paper, we did not consider a task model with a jitter parameter. Studying the impact of the task jitter on both sustainability and schedulability would be an interesting problem to deal with in future work. Finally, some results presented in this paper, and especially the offline approach, could be extended to cope with task models using more precise CRPD parameters (i.e. taking into account the preempted and preempting tasks).

References

Altmeyer S, Davis R, Maiza C (2011) Pre-emption Cost Aware Response Time Analysis for Fixed Priority Pre-emptive Systems. Tech. Rep. YCS-2010-464, University of York, Department of Computer Science

Altmeyer S, Davis R, Maiza C (2012) Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. Real-Time Systems 48(5):499–526

Altmeyer S, Douma R, Lunniss W, Davis R (2014) OUTSTANDING PAPER: Evaluation of Cache Partitioning for Hard Real-Time Systems. In: Proceedings of the 2014 26th Euromicro Conference on Real-Time Systems (ECRTS), pp 15–26

Baruah S, Burns A (2006) Sustainable Scheduling Analysis. In: Proceedings of the 2006 27th IEEE International Real-Time Systems Symposium (RTSS), pp 159–168

Bastoni A, Brandenburg B, Anderson J (2010) Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In: Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2010), pp 33–44

Bate IJ (1998) Scheduling and timing analysis for safety critical real-time systems. PhD thesis, Department of Computer Science, University of York

Bertogna M, Xhani O, Marinoni M, Esposito F, Buttazzo G (2011) Optimal Selection of Preemption Points to Minimize Preemption Overhead. In: Proceedings of the 2011 23rd Euromicro Conference on Real-Time Systems (ECRTS), pp 217–227

Bini E, Buttazzo G (2005) Measuring the Performance of Schedulability Tests. Real-Time Systems 30(1-2):129–154

Brandenburg BB (2011) Scheduling and locking in multiprocessor real-time operating systems. PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill

Bril R, Altmeyer S, Van Heuvel M, Davis R, Behnam M (2014) Integrating Cache-Related Pre-Emption Delays into Analysis of Fixed Priority Scheduling with Pre-Emption Thresholds. In: Proceedings of the 2014 IEEE Real-Time Systems Symposium (RTSS), pp 161–172

Burns A (1995) Advances in Real-time Systems. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, chap Preemptive Priority-based Scheduling: An Appropriate Engineering Approach, pp 225–248

Burns A, Baruah S (2008) Sustainability in real-time scheduling. Journal of Computing Science and Engineering 2(1):74–97

Busquets-Mataix J, Serrano J, Ors R, Gil P, Wellings A (1996a) Adding instruction cache effect to schedulability analysis of preemptive real-time systems. In: Proceedings of the 1996 IEEE Real-Time Technology and Applications Symposium (RTAS), pp 204–212

Busquets-Mataix J, Serrano-Martin J, Ors-Carot R, Gil P, Wellings A (1996b) Adding instruction cache effect to an exact schedulability analysis of preemptive real-time systems. In: Proceedings of the 1996 Eighth Euromicro Workshop on Real-Time Systems, pp 271–276

Buttazzo G (2005) Rate Monotonic vs. EDF: Judgment Day. Real-Time Systems 29(1):5–26

Buttazzo G (2011) HARD REAL-TIME COMPUTING SYSTEMS: Predictable Scheduling Algorithms and Applications, Real-Time System Series, vol 24, 3rd edn. Springer US

Calandrino J, Anderson J (2008) Cache-Aware Real-Time Scheduling on Multicore Platforms: Heuristics and a Case Study. In: Proceedings of the 2008 Euromicro Conference on Real-Time Systems (ECRTS), pp 299–308

Cavicchio J, Tessler C, Fisher N (2015) Minimizing Cache Overhead via Loaded Cache Blocks and Preemption Placement. In: Proceedings of the 2015 27th Euromicro Conference on Real-Time Systems (ECRTS), pp 163–173

Ding H, Liang Y, Mitra T (2014) WCET-centric Dynamic Instruction Cache Locking. In: Proceedings of the Conference on Design, Automation & Test in Europe, European Design and Automation Association, 3001 Leuven, Belgium, Belgium, DATE '14, pp 27:1–27:6

Ferdinand C, Wilhelm R (1999) Efficient and Precise Cache Behavior Prediction for Real-Time Systems. Real-Time Systems 17(2-3):131–181

Fisher N, Goossens J, Baruah S (2010) Optimal online multiprocessor scheduling of sporadic real-time tasks is impossible. Real-Time Systems 45(1-2):26–71

Keskin U, Bril R, Lukkien J (2010) Exact response-time analysis for fixed-priority preemption-threshold scheduling. In: Proceedings of the 2010 IEEE Conference on Emerging Technologies and Factory Automation (ETFA), pp 1–4

Lee CG, Hahn J, Seo YM, 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 Transactions on Computers 47(6):700–713

Lee J, Shin K (2014) Preempt a Job or Not in EDF Scheduling of Uniprocessor Systems. IEEE Transactions on Computers 63(5):1197–1206

Levinthal D (2009) Performance Analysis Guide for Intel®Core^{TM}i7 Processor and Intel®Xeon^{TM}5500 processors. Tech. rep., Intel, URL `https://software.intel.com`

Liu C, Layland J (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM) 20(1):46–61, DOI 10.1145/321738.321743

Lunniss W, Altmeyer S, Davis R (2012) Optimising Task Layout to Increase Schedulability via Reduced Cache Related Pre-emption Delays. In: Proceedings of the 20th International Conference on Real-Time and Network Systems, ACM, New York, NY, USA, RTNS '12, pp 161–170

Lunniss W, Altmeyer S, Maiza C, Davis R (2013) Integrating cache related pre-emption delay analysis into EDF scheduling. In: Proceedings of the 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), pp 75–84

Lunniss W, Altmeyer S, Davis R (2014) A Comparison between Fixed Priority and EDF Scheduling accounting for Cache Related Pre-emption Delays. Leibniz Transactions on Embedded Systems 1(1):01–1–01:24

Mok A (1983) Fundamental design problems of distributed systems for the hard-real-time environment. PhD thesis, Massachusetts Institute of Technology

Pellizzoni R, Caccamo M (2007) Toward the Predictable Integration of Real-Time COTS Based Systems. In: Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International, pp 73–82

Peng B, Fisher N, Bertogna M (2014) Explicit Preemption Placement for Real-Time Conditional Code. In: Proceedings of the 2014 26th Euromicro Conference on Real-Time Systems (ECRTS), pp 177–188

Phavorin G, Richard P, Goossens J, Chapeaux T, Maiza C (2015a) Scheduling with Preemption Delays: Anomalies and Issues. In: Proceedings of the 23rd International Conference on Real Time and Networks Systems, ACM, New York, NY, USA, RTNS '15, pp 109–118

Phavorin G, Richard P, Maiza C (2015b) Complexity of scheduling real-time tasks subjected to cache-related preemption delays. In: Proceedings of the 2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA), pp 1–8

Phavorin G, Richard P, Maiza C (2015c) Complexity of scheduling real-time tasks subjected to cache-related preemption delays. Research Report no. 2, LIAS, Université de Poitiers, URL `http://www.lias-lab.fr/publications/18099/rapport\_recherche.pdf`, 10p.

Reineke J, Altmeyer S, Grund D, Hahn S, Maiza C (2014) Selfish-LRU: Preemption-aware caching for predictability and performance. In: Proceedings of the 2014 IEEE 20th Real-Time and Embedded Technology and Applications Symposium (RTAS), pp 135–144

Tomiyama H, Dutt N (2000) 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, ACM, New York, NY, USA, CODES '00, pp 67–71

Vera X, Lisper B, Xue J (2003) Data caches in multitasking hard real-time systems. In: Proceedings of the 2003 24th IEEE Real-Time Systems Symposium (RTSS), pp 154–165

Wang C, Gu Z, Zeng H (2015) Integration of Cache Partitioning and Preemption Threshold Scheduling to Improve Schedulability of Hard Real-Time Systems. In: Proceedings of the 2015 27th Euromicro Conference on Real-Time Systems (ECRTS), pp 69–79

Whitham J, Audsley N (2012) Explicit Reservation of Local Memory in a Predictable, Preemptive Multitasking Real-Time System. In: Proceedings of the 2012 IEEE 18th Real-Time and Embedded Technology and Applications Symposium (RTAS), pp 3–12, DOI 10.1109/RTAS.2012.19

Xu J, Parnas D (1993) On satisfying timing constraints in hard-real-time systems. IEEE Transactions on Software Engineering 19(1):70–84

Yao G, Buttazzo G, Bertogna M (2011) Feasibility analysis under fixed priority scheduling with limited preemptions. Real-Time Systems 47(3):198–223

Yomsi P, Sorel Y (2007) Extending Rate Monotonic Analysis with Exact Cost of Preemptions for Hard Real-Time Systems. In: Proceedings of the 2007 19th Euromicro Conference on Real-Time Systems (ECRTS), pp 280–290