Integrating Cache-Related Pre-emption Delays into Analysis of Fixed Priority Scheduling with Pre-emption Thresholds

Reinder J. Britt\textsuperscript{a,b}, Sebastian Altmeurer\textsuperscript{1}, Martijn M.H.P. van den Heuvel\textsuperscript{b}, Robert I. Davis\textsuperscript{d}, Moris Behnam\textsuperscript{a}

\textsuperscript{a}Mälardalen University (MDH), Sweden, \textsuperscript{b}Technische Universiteit Eindhoven (TU/e), The Netherlands, \textsuperscript{c}University of Amsterdam (UvA), The Netherlands, \textsuperscript{d}University of York, UK

Abstract—Cache-related pre-emption delays (CRPD) have been integrated into the schedulability analysis of sporadic tasks with constrained deadlines for fixed-priority pre-emptive scheduling (FPPS). This paper generalizes that work by integrating CRPD into the schedulability analysis of tasks with arbitrary deadlines for fixed-priority pre-emption threshold scheduling (FPTS). The analysis is complemented by an optimal threshold assignment algorithm that minimizes CRPD. The paper includes a comparative evaluation of the schedulability ratios of FPTS and FPPS, for constrained-deadline tasks, taking CRPD into account.

I. INTRODUCTION

For cost-effectiveness reasons, it is preferred to use commercial off-the-shelf (COTS) programmable platforms for real-time embedded systems, rather than dedicated, application-domain specific platforms. These COTS platforms typically contain a cache to bridge the gap between the processor speed and main memory speed and to reduce the number of conflicts with other devices on the system bus. Unfortunately, caches give rise to additional delays upon pre-emptions due to cache flushes and reloads of blocks that are replaced during pre-emption. This cache-related pre-emption delay (CRPD) can have a significant impact on the computation times of tasks. For fixed-priority pre-emptive scheduling (FPPS), which is the de facto standard used in industry, CRPD has therefore been integrated into the schedulability analysis [17, 26, 32, 28, 3].

Recently, limited pre-emptive scheduling schemes received a lot of attention from academia. In particular, fixed-priority scheduling with limited pre-emptions, such as fixed-priority pre-emption scheduling with deferred pre-emption (FPDS) or co-operative scheduling [16, 14, 20] and fixed-priority scheduling with pre-emption thresholds (FPTS) [33, 31, 30, 25], are considered viable alternatives between the extremes of FPPS and fixed-priority non-pre-emptive scheduling (FPNS). Compared to FPPS, limited pre-emptive schemes can (i) reduce memory requirements [31, 23, 21] and (ii) reduce the cost of arbitrary pre-emptions [16, 14, 10]. Compared to both FPPS and FPNS, these schemes may significantly improve the feasibility of a task set [14, 31, 8, 20].

Although FPDS clearly outperforms FPTS from a theoretical perspective [18], applying FPDS in practice is still a challenge because pre-emption points have to be explicitly added in the code. Assuming strictly periodic tasks with known phasing, a single non-pre-emptive region (NPR) can significantly reduce the preemptions that can feasibly occur [29]. Alternatively, sporadic tasks with floating NPRs [34, 6] can be used; however, these require specific operating-system support and can lead to preemptions by all higher priority tasks at arbitrary points in the code which may incur substantially higher CRPD costs.

FPTS, on the other hand, can be easily applied for sporadic task systems, even without any changes to the code when pre-emption thresholds can be assigned to tasks at integration time, e.g. by means of dedicated primitives or by means of code-wrappers using standard synchronization primitives. Such support is specified by both the OSEK [1] and AUTOSAR [2] operating-system standards, in the form of internal resources\textsuperscript{1}, and deployed in the automotive industry. FPTS may therefore be used for legacy code and viewed as an evolutionary successor of FPPS as the de facto standard in industry. To the best of our knowledge, however, integration of CRPD in the schedulability analysis of sporadic tasks for FPTS has not been addressed and is therefore the topic of this paper.

The limited pre-emptive nature of FPTS gives rise to specific challenges when integrating CRPD in the analysis, in particular to prevent over-estimations of CRPD. For example, not all tasks contributing to the worst-case response time of a task can actually pre-empt the execution of a job of that task, unlike with FPPS, as illustrated by a non-pre-emptive task. Next, there does not exist an Optimal Threshold Assignment (OTA) algorithm minimizing CRPD. Finally, existing comparisons between FPPS and FPTS, e.g. [18], do not consider CRPD.

This paper presents three major contributions, i.e. (i) analysis for FPTS with CRPD (Sections IV - VII), (ii) an OTA algorithm for FPTS with CRPD that minimizes CRPD (Section VIII), and (iii) a comparative evaluation of the schedulability ratio of task sets under FPPS and FPTS, for constrained-deadline tasks, taking CRPD into account (Section IX).

II. REAL-TIME SCHEDULING MODEL

A. Basic model for FPPS

We assume a single processor and a set $T$ of $n$ independent sporadic tasks $\tau_1, \tau_2, \ldots, \tau_n$ with unique priorities $\pi_1, \pi_2, \ldots, \pi_n$. At any moment in time, the processor is used to execute the highest priority task that has work pending. For notational convenience, we assume that (i) tasks are given in order of decreasing priorities, i.e. $\tau_1$ has the highest and $\tau_n$ the lowest priority, and (ii) a higher priority is represented by a higher value, i.e. $\pi_1 > \pi_2 > \ldots > \pi_n$. We use $hp(\pi)$ (and $lp(\pi)$) to denote the set of tasks with priorities higher than (lower than) $\pi$.

\textsuperscript{1}The restriction to one internal resource per task in OSEK and AUTOSAR needs to be lifted to fully implement FPTS. In this way, FPTS is supported by ETAS’ RTA-OSSK and RTA-OS operating systems, which have been deployed in 50 to 55 million new ECUs per year since 2008 [19].

161
Similarly, we use \( \text{hep}(\pi) \) (and \( \text{lep}(\pi) \)) to denote the set of tasks with priorities higher (lower) than or equal to \( \pi \).

Each task \( \tau_i \) is characterized by a minimum inter-activation time \( T_i \in \mathbb{R}^+ \), a worst-case computation time \( C_i \in \mathbb{R}^+ \), and a (relative) deadline \( D_i \in \mathbb{R}^+ \). We assume that the constant pre-emption costs, such as context switches, are subsumed into the worst-case computation times. We feature arbitrary deadlines, i.e. the deadline \( D_i \) may be smaller than, equal to, or larger than the period \( T_i \). The utilization \( U_i \) of task \( \tau_i \) is given by \( C_i/T_i \), and the utilization \( U \) of the set of tasks \( T \) by \( \sum_{\tau \in T} U_i \). An activation of a task is also termed a job.

For notational convenience, we introduce \( E_i(t) = \left\lceil t/T_i \right\rceil \) and \( E'_i(t) = \left\lfloor 1 + t/T_i \right\rfloor \) to represent the maximum number of activations of \( \tau_i \) in an interval \([x, x+t]\) and \([x, x+t)\), respectively, where both intervals have a length \( t \).

### B. Refined model for FPTTS

In FPTTS, each task \( \tau_i \) has a pre-emption threshold \( \theta_i \), where \( \pi_i \geq \theta_i \geq \pi_i \). When \( \tau_i \) is executing, it can only be pre-empted by tasks with a priority higher than \( \theta_i \). Note that we have FPPS and FFNS as special cases when \( V_{1 \leq } V \theta_i = \pi_i \) and \( V_{1 \leq } V \theta_i = \pi_i \), respectively.

We use \( \text{het}(\pi) \) (and \( \text{lt}(\pi) \)) to denote the set of tasks with thresholds higher than or equal to (lower than) \( \pi \). Finally, we use \( \text{bit}(i) \) to denote the set of tasks that may block \( \tau_i \) due to their preemption threshold assignment. An overview of notations for sets of tasks is given in Table I. Note that for FPPS \( \text{hep}(\pi) = \text{het}(\pi), \text{lep}(\pi) = \text{lt}(\pi) \), and \( \text{bit}(i) = \emptyset \).

<table>
<thead>
<tr>
<th>Table I</th>
<th>Notations for various sets of indices of tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{hep}(\pi) )</td>
<td>( \text{het}(\pi) ) (( \theta_i \geq \pi ))</td>
</tr>
<tr>
<td>( \text{lep}(\pi) )</td>
<td>( \text{lt}(\pi) ) (( \theta_i &gt; \pi ))</td>
</tr>
<tr>
<td>( \text{bp}(\pi) )</td>
<td>( \text{bit}(i) ) (( \text{bit}(i) \neq \emptyset ))</td>
</tr>
</tbody>
</table>

### C. A model for cache-related pre-emption costs

For ease of presentation, we assume direct-mapped caches, similar to [3]. The scheduling analysis integrating CRPD is based on the concepts of evicting cache blocks (ECBs) and useful cache blocks (UCBs) [26, 4]. A memory block that may be accessed by a task is termed an ECB, as it may evict a cache block of another task. A cache block that may be (re-) used at multiple program points without being evicted by the task itself is termed a UCB. The set of UCBs and ECBs of tasks can be analyzed with, for example, a prototype version of AbsInt’s aT Timing Analyzer for ARM [22]. Similar to [3], in the current paper the sets of ECBs and UCBs are represented as sets of integers, where each integer represents a cache set.

The worst-case block-reload time (BRT) is given by a constant. Example 1 shows the relation between the ECBs of a task (ECB\(_i\)), the UCBs of a task (UCB\(_i\)) and the BRT.

**Example 1.** We assume a direct-mapped cache with 4 cache sets and two tasks \( \tau_1 \) and \( \tau_2 \). The memory blocks of \( \tau_1 \) map to cache set 0 and 1 and \( \tau_2 \)’s memory block mapping to cache set 1 is useful, i.e. ECB\(_1\) = \{0,1\} and UCB\(_1\) = \{1\}. The memory blocks of \( \tau_2 \) map to cache set 1, 2, and 3 and all three are useful, i.e. ECB\(_2\) = \{1,2,3\} and UCB\(_2\) = \{1,2,3\}. The cache-related pre-emption cost of task \( \tau_1 \) pre-empting task \( \tau_2 \) is thus given as follows:

\[
\text{ECB}_1 \cap \text{UCB}_1 \cap \text{BRT} = [1,2) \cap \text{BRT} = 2 \cdot \text{BRT}.
\]

The cache utilization \( U^C \) of task \( \tau_i \) is given by \( |\text{ECB}_i \cap \text{N}|/|N| \), where \( |\text{ECB}_i| \) denotes the number of ECBs of \( \tau_i \) and \( N \) denotes the number of cache sets. The total cache utilization \( U^C \) of the set of tasks \( T \) is given by \( \sum_{\tau \in T} U^C_i = \sum_{\tau \in T} |\text{ECB}_i|/|N| \).

### III. Recap of response time analysis for FPPS and FPTTS

This section starts with a recapitulation of the exact schedulability analysis for FPTTS, as presented in [25]. Next, that analysis is specialized for FPPS with constrained deadlines, i.e. for cases with \( D_i \leq T_i \), and extended with CRPD [3].

#### A. FPTTS with arbitrary deadlines (without CRPD)

A set \( T \) of tasks is schedulable if and only if for every task \( \tau_i \in T \) its worst-case response time \( R_i \) is at most equal to its deadline \( D_i \), i.e. \( \sum_{\tau \in T} R_i \leq D_i \). To determine \( R_i \), we need to consider the worst-case response times of all jobs in a so-called level-i active period [14]. The worst-case length \( L_i \) of that period is given by the smallest positive solution of

\[
L_i = B_i + \sum_{j : \text{hep}(j)} E_j(L_i) \cdot C_j,
\]

where \( B_i \) denotes the worst-case blocking of task \( \tau_i \), given by

\[
B_i = \max \left( 0, \max_{j : \text{bit}(j)} C_j \right).
\]

\( L_i \) can be found by fixed point iteration that is guaranteed to terminate for all \( i \) when \( U < 1 \) [14].

For a job \( k \) of \( \tau_i \), with \( 0 \leq k < E_i(L_i) \), the worst-case start time \( S_{i,k} \) and worst-case finalization time \( F_{i,k} \) are given by

\[
S_{i,k} = B_i + kC_i + \sum_{j : \text{lep}(j)} E_j(S_{i,k}) \cdot C_j \quad \text{if } B_i > 0
\]

\[
F_{i,k} = S_{i,k} + C_i + \sum_{j : \text{lep}(j)} E_j(S_{i,k}) \cdot C_j \quad \text{if } B_i = 0
\]

and

\[
F_{i,k} = S_{i,k} + C_i + \sum_{j : \text{lep}(j)} E_j(S_{i,k}) \cdot C_j \quad \text{if } B_i = 0
\]

Later in this paper we prove that (4) can be simplified by removing the case distinction, because \( E_j(S_{i,k}) = E_j^*(S_{i,k}) \) (see Corollary 1). Similar to \( L_i \), the values for \( S_{i,k} \) and \( F_{i,k} \) can be found by means of an iterative procedure.

The worst-case response time \( R_i \) of \( \tau_i \) is now given by

\[
R_i = \max_{0 \leq k < E_i(L)} (F_{i,k} - k \cdot T_i).
\]
B. FPPS with constrained deadlines and CRPD

FPPS is a special case of FPTS, and the analysis of FPTS can therefore be simplified for FPPS. For FPPS with constrained deadlines, the worst-case response time $R_i$ of task $r_i$ is given by the smallest positive solution [24, 5] of

$$R_i = C_i + \sum_{j \in \text{dep}(i)} E_j(R_j) \cdot C_j,$$

(6)

An upper bound for $R_i$ with CRPD [32, 3] can be found using

$$R_i = C_i + \sum_{j \in \text{dep}(i)} \left(E_j(R_j) \cdot C_j + \gamma_{i,j}(R_j)\right),$$

(7)

where $\gamma_{i,j}(R_j)$ represents the cache-related pre-emption cost due to all jobs of a higher priority pre-empting task $r_j$ executing within the worst-case response time of task $r_i$. The definition of $\gamma_{i,j}(t)$ depends on the specific approach chosen for determining these costs [3].

Integration of CRPD in the schedulability analysis of tasks has been addressed for FPPS with a focus on the pre-empting tasks [17], the pre-empted tasks [26], and by considering both the pre-empting and pre-empted tasks [32, 3]. The ECB-Only approach and UCB-Only Multiset approach focus on just the pre-empting tasks and just the pre-empted tasks, respectively. The ECB-Union approach and UCB-Union Multiset approach consider a combination of pre-empting and pre-empted tasks.

1) ECB-Only approach: For this case, $\gamma_{i,j}(t)$ is given by

$$\gamma_{i,j}^{\text{ecb}}(t) = \begin{cases} \text{BRT} \cdot E_j(t) \cdot \text{ECB} & \text{if } \text{aff(}\pi_i,\pi_j) \neq \emptyset, \\ 0 & \text{otherwise}, \end{cases}$$

(8)

where $\text{aff}(\pi_i, \pi_j)$ denotes the set of tasks that have a priority $(i)$ higher than or equal to $\pi_i$, i.e. can affect the response time of $r_i$, and $(ii)$ lower than $\pi_j$, i.e. can be pre-empted by $r_j$. For FPPS with constrained deadlines, $\text{aff}(\pi_i, \pi_j)$ is defined as

$$\text{aff}(\pi_i, \pi_j) = \text{ip}(\pi_i) \cap \text{hep}(\pi_i).$$

(9)

2) UCB-Only Multiset and ECB-Union Multiset approaches: For these approaches, $\gamma_{i,j}(t)$ is defined as

$$\gamma_{i,j}^{\text{ucb}}(t) = \text{BRT} \cdot \sum_{j \in \text{sort}(i)} \text{sort}(\text{M}_{i,j}(t)) \cdot |t|,$$

(10)

where the function $\text{sort}$ sorts the sets of the multiset $\text{M}_{i,j}(t)$ in non-increasing order of their size. Hence, the sum of the sizes of the $E_j(t)$ largest sets of the multiset $\text{M}_{i,j}(t)$ is taken and multiplied by BRT.

For the UCB-Only Multiset approach, the multiset $\text{M}_{i,j}(t)$ contains $E_j(R_j) \cdot E_k(t)$ copies of the size of the UCBs of each task $h \in \text{aff}(\pi_i, \pi_j)$ affected task $r_j$ and affected by task $r_i$, i.e.

$$\text{M}_{i,j}^{\text{ucb}}(t) = \sum_{k \in \text{hep}(\pi_j)} \left(\sum_{h \in \text{aff}(\pi_i, \pi_j)} \text{UCB}_h \cap E_j(R_j) \cdot E_k(t)\right).$$

(11)

Instead, for the ECB-Union Multiset approach, for each task $h \in \text{aff}(\pi_i, \pi_j)$ the multiset $\text{M}_{i,j}(t)$ contains $E_j(R_j) \cdot E_k(t)$ copies of the size of the intersection of the UCBs and the ECBs of all tasks in hep($\pi_i$), i.e.

$$\text{M}_{i,j}^{\text{ecb-union}}(t) = \sum_{k \in \text{hep}(\pi_j)} \left(\sum_{h \in \text{aff}(\pi_i, \pi_j)} \text{UCB}_h \cap E_j(R_j) \cdot E_k(t)\right).$$

(12)

Note that (12) extends (11) by intersecting every UCB with $\text{ECB}_h$.

3) ECB-Union Multiset approach: For this approach, first a multiset $\text{M}_{i,j}^{\text{ucb}}(t)$ is formed containing $E_j(R_j) \cdot E_k(t)$ copies of the UCBs of each task $h \in \text{aff}(\pi_i, \pi_j)$, i.e.

$$\text{M}_{i,j}^{\text{ecb-union}}(t) = \sum_{k \in \text{hep}(\pi_j)} \left(\sum_{h \in \text{aff}(\pi_i, \pi_j)} \text{UCB}_h \cap E_j(R_j) \cdot E_k(t)\right).$$

(13)

Apart from the cardinality operator in (11), the equations (11) and (13) are identical. Next a multi-set $\text{M}_{i,j}^{\text{ecb}}(t)$ is formed containing $E_j(t)$ copies of the ECB for each task $r_j$, i.e.

$$\text{M}_{i,j}^{\text{ecb}}(t) = \sum_{k \in \text{hep}(\pi_j)} \left(\sum_{h \in \text{aff}(\pi_i, \pi_j)} \text{UCB}_h \cap E_j(t)\right).$$

(14)

The CRPD $\gamma_{i,j}^{\text{ecb-union}}(t)$ is then given by the size of the multi-set intersection of $\text{M}_{i,j}^{\text{ecb-union}}(t)$ and $\text{M}_{i,j}^{\text{ecb}}(t)$ multiplied by BRT, i.e.

$$\gamma_{i,j}^{\text{ecb-union}}(t) = \text{BRT} \cdot \left(\text{M}_{i,j}^{\text{ecb-union}}(t) \cap \text{M}_{i,j}^{\text{ecb}}(t)\right).$$

(15)

In the remainder of this paper, we follow a similar structure for extending FPTS with CRPD. Before looking at specific approaches, we consider challenges for FPTS with CRPD (Section IV). We subsequently focus on pre-empting tasks (Section V), pre-empted tasks (Section VI), and the combination of pre-empting and pre-empted tasks (Section VII).

IV. FPTS WITH CRPD: PRELIMINARIES AND CHALLENGES

To extend the schedulability analysis of FPTS with CRPD, we must extend the corresponding formulas. For this purpose, we extend $L_i$ in (1), $S_{i,k}$ in (3) and $F_{i,k}$ in (4) with a new term $\gamma_{i,j}(t)$ in a similar way as the $R_i$ in (7) has been extended for FPPS with constrained deadlines. However, due to (i) the generalization towards arbitrary deadlines and (ii) the limited-pre-emptive nature of FPTS, it is not possible to simply extend these equations for FPTS with a term $\gamma_{i,j}(t)$ by reusing the existing approaches to determine CRPD. This section addresses preliminaries and challenges for FPTS with CRPD.

A. Distinguishing executing and affected tasks

The extension for FPPS is based on the tasks that can execute and affected the execution of a task $r_i$ in the interval under consideration. An overview of these tasks for the response interval $[0, R_i]$ is given in Table II, i.e. the table shows

- **interval**: a description of an interval under consideration;
- **execute**: the tasks that can execute jobs in the interval;
- **affected by**: the set of tasks that can execute jobs in the interval and can be pre-empted by task $r_j$.
### TABLE II
Overview of tasks that can execute and affect the execution of task \( r_i \) in a level-i active period starting at time \( i = 0 \) for both FFPS with constrained deadlines and PPTS with arbitrary deadlines, assuming a task \( r_j \) that blocks \( r_i \) for PPTS, i.e., \( b \in b(i) \).

<table>
<thead>
<tr>
<th>interval</th>
<th>FFPS</th>
<th>PPTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>([0,R_i])</td>
<td>(b(r_i))</td>
<td>(b(r_i))</td>
</tr>
<tr>
<td>([0,L_i])</td>
<td>(b(r_i))</td>
<td>(b(r_i))</td>
</tr>
<tr>
<td>([0,L_j])</td>
<td>(b(r_j))</td>
<td>(b(r_j))</td>
</tr>
<tr>
<td>([0,R_j])</td>
<td>(b(r_j))</td>
<td>(b(r_j))</td>
</tr>
</tbody>
</table>

### a-jobs: the number of job activations of a task that can execute in the interval.

The "a-jobs" in the interval \([0,R_i]\) can be immediately derived from \( R_i \), see (6). If \( R_i \leq D_i \leq T_i \), then \( E_i(R_i) = 1 \) and, as a result, task \( r_i \) can be treated as any other task.

When we focus only on the pre-empting tasks, e.g., when using the ECB-Only approach, we only need the information of the row affected by \( r_i \), see (8). When we focus on the pre-empted tasks, e.g., when using the UCBO-Only Multiset approach, the a-jobs also play a role, i.e., the multiset \( M_i^{\text{job}}(t) \) in (11) contains \( E_i(R_i) \cdot E_i(i) \) copies of the size of \( \text{UCB}_b \) for each task \( h \in \text{aff}(r_i, r_j) \) affecting \( r_i \) and affected by \( r_j \).

In the next sections, the information in Table II forms the basis for the extensions for PPTS with CRPD.

### B. Bounding the number of pre-emptions using hold times

For FFPS with constrained deadlines, all pre-emptions during the response time of a job of a task may actually evict UCBs of that job. For PPTS, however, some pre-emptions can only take place between the activation and the start of a job, and therefore do not evict UCBs of that job. An obvious example is a non-pre-empive task, where no pre-emption can take place during the actual execution of its jobs.

To prevent pessimism in the analysis when focussing on pre-empted tasks, we consider so-called hold times. To that end, we distinguish the (absolute) activation time \( a_{k,t} \), (absolute) start-time \( s_{k,t} \) and (absolute) finishing time \( f_{k,t} \) of a job \( k \) of task \( r_i \); see Figure 1. The length of the interval \([a_{k,t}, f_{k,t}]\) and \([s_{k,t}, f_{k,t}]\) is termed the response time and the hold time of job \( k \) of task \( r_i \), respectively.

![Fig. 1. The notion of hold time is inspired by the term resource hold times in [9] and the observation in [21, 23] that it is possible to make two tasks mutually non-pre-empive by letting them share a so-called pseudo-resource. Our hold time is the same as the resource hold time of the pseudo-resource.](image)

Under FFPS, the worst-case hold time \( H_i \) of a task \( r_i \) can be calculated by means of (6), i.e., by using the equation to determine the worst-case response time \( R_i \) for FFPS with constrained deadlines; see (12, 13). Under PPTS, only tasks with a priority higher than \( b_i \) can pre-empt \( r_i \). Hence, the worst-case hold time \( H_i \) (without CRPD) is given by

\[
H_i = C_i + \sum \frac{E_j(H_j)}{b_j} \cdot C_j
\]

The worst-case hold time \( H_i \) of a task \( r_i \) may be smaller than the worst-case response time \( R_i \). This is because (i) the potential delay of the execution of a job by a previous job [13], (ii) the blocking by a task \( r_k \) with \( b \in b(i) \), and (iii) the interference of tasks \( r_j \) with \( j \in \text{aff}(r_i, r_j) \) affecting \( r_i \) but not in \( H_i \). Example 2 below illustrates (i) and Example 3 illustrates (ii) and (iii).

#### Example 2. The characteristics of a set \( T_2 \) of periodic tasks

The timeline shown in Figure 2 illustrates both the worst-case hold time \( H_2 = 8.2 \) and the worst-case response time \( R_2 = 8.6 \) for the job activated at time \( t = 14 \). \( R_2 \) is larger than \( H_2 \), because \( R_2 \) includes a delay of 0.4 of the job activated at time \( t = 7 \). This illustrates (i).

#### Example 3. The characteristics of a set \( T_3 \) of periodic tasks

Tasks \( r_1 \) and \( r_2 \) of Example 3 are particularly interesting when PPTS is extended with CRPD, because task \( r_1 \) can be activated twice during their worst-case response time but only once during their worst-case hold time.

### TABLE III
Task characteristics of \( T_2 \) and worst-case response times and hold times of periodic tasks with non-constrained deadlines under FFPS without CRPD.

<table>
<thead>
<tr>
<th>( T )</th>
<th>( D )</th>
<th>( C )</th>
<th>( \pi )</th>
<th>( \theta )</th>
<th>( R )</th>
<th>( H )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_2 )</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>8.2</td>
</tr>
<tr>
<td>( T_2 )</td>
<td>7</td>
<td>9</td>
<td>4.2</td>
<td>1</td>
<td>8.6</td>
<td>8.2</td>
</tr>
</tbody>
</table>

### TABLE IV
Task characteristics of \( T_3 \) and worst-case response times and hold times of periodic tasks under PPTS without CRPD.

<table>
<thead>
<tr>
<th>( T )</th>
<th>( D )</th>
<th>( C )</th>
<th>( \pi )</th>
<th>( \theta )</th>
<th>( R )</th>
<th>( H )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_2 )</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>8.2</td>
</tr>
<tr>
<td>( T_3 )</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>8.6</td>
</tr>
<tr>
<td>( T_3 )</td>
<td>9</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>( T_3 )</td>
<td>11</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>
C. Determining the number of job activations “#-jobs”

We now show that we can derive the “#-jobs” for FPTS in Table II from the equations corresponding to the intervals, similar to FPPS. We start with the interval \([0, H_i]\). The intervals \([0, L_i]\), \([0, S_{i,k}]\), and \([0, F_{i,k}]\) are subsequently addressed for \(R \neq 0\) and \(B_i = 0\).

1) #-jobs for \([0, H_i]\): The “#-jobs” for the interval \([0, H_i]\) follows immediately from (16). Exactly 1 activation of \(t_i\) is taken into account. To prevent pessimism when \(T_i\) is smaller than \(H_i\), Table II contains a dedicated clause for identifying the appropriate number of job activations of task \(t_i\) itself.

Example 4. We reconsider \(T_2\) of Example 2. For that example, \(E_2(H_2) = 2\) rather than 1.

2) #-jobs for \([0, L_i]\), \([0, S_{i,k}]\), and \([0, F_{i,k}]\) when \(B_i \neq 0\): Given a task \(t_k\) that blocks \(t_i\) under FPTS, i.e. \(b \in b(t_i)\), the number of activations #-jobs in the intervals \([0, L_i]\), \([0, S_{i,k}]\), and \([0, F_{i,k}]\) in Table II can be immediately derived from (1) for \(L_i\), (3) for \(S_{i,k}\) and (4) for \(F_{i,k}\). To prevent pessimism, exactly one activation of \(t_k\) is taken into account. Similarly, exactly \(k\) and \(k + 1\) jobs of \(t_i\) are taken into account when determining \(S_{i,k}\) and \(F_{i,k}\), respectively.

Example 5. We reconsider \(T_2\) of Example 2. The worst-case finalization time \(F_{2,0}\) of the first job of \(t_2\) is equal to 8.2. Because \(E_2(8.2) = 2\), (11) would include 2 jobs of \(t_2\) in \(M_{2,0}^{\text{FPTS}}(8.2)\) rather than 1. To prevent this pessimism, we explicitly take the number of jobs of \(t_i\) into account.

3) #-jobs for \([0, L_i]\), \([0, S_{i,k}]\), and \([0, F_{i,k}]\) when \(B_i = 0\): Lemma 1 shows that the * can be removed from \(E_j(S_{i,k})\) for the case \(B_i = 0\) in (4) for \(F_{i,k}\).

Lemma 1. Let \(j \in b_p(n)\) and assume a level-i active period starting at time \(t = 0\) with a simultaneous release of \(t_i\) and \(t_j\). Let \(S_{i,k}\) denote the worst-case start time of job \(k\) of \(t_i\) in that level-i active period and be derived by (3). Now the following equality holds:

\[
\forall j \in b_p(n) \quad E_j(S_{i,k}) = E_j(S_{i,k}).
\]  

Proof. The term \(E_j(S_{i,k})\) represents the maximum number of activations of \(t_i\) in the interval \([0, S_{i,k}]\). When \(\exists_{\text{init}} S_{i,k} = m \cdot T_j\), task \(t_j\) is activated at time \(S_{i,k}\). This would imply that \(t_i\) cannot start at \(S_{i,k}\), which contradicts the definition of \(S_{i,k}\). We therefore conclude that \(\exists_{\text{init}} S_{i,k} = m \cdot T_j\). As a result, \(E_j(S_{i,k}) = E_j(S_{i,k})\), which proves the lemma.

Corollary 1. We may simplify (4) by replacing \(E_j(S_{i,k})\) by \(E_j(S_{i,k})\) and ignoring the case distinction, i.e.

\[
F_{i,k} = S_{i,k} + C_j + \sum_{j \in b_p(n)} \left( E_j(F_{i,k}) - E_j(S_{i,k}) \right) \cdot C_j.
\]  

Similarly, Lemma 2 shows that \(\gamma_{j,i}(t)\) can be defined in terms of \(E_j(S_{i,k})\) rather than \(E_j(S_{i,k})\) for the case \(B_i = 0\) in (3) when determining \(S_{i,k}\).

Lemma 2. When \(S_{i,k}\) is extended with a term \(\gamma_{j,i}(t)\) for the case \(B_i = 0\), \(\gamma_{j,i}(t)\) can be based on \(E_j(S_{i,k})\) rather than \(E_j(S_{i,k})\).

Proof. A solution for the recurrent relation for \(S_{i,k}\) is found when \(S_{i,k} = S_{i,k}^{(t)}\) for two subsequent iterations. For \(S_{i,k}^{(t)}\), there are two cases, either \(E_j(S_{i,k}^{(t)}) = E_j(S_{i,k}^{(t-1)})\) or \(E_j(S_{i,k}^{(t)}) \neq E_j(S_{i,k}^{(t-1)})\).

Let \(E_j(S_{i,k}^{(t)}) = E_j(S_{i,k}^{(t-1)})\), i.e. \(\exists_{\text{init}} S_{i,k}^{(t)} = m \cdot T_j\). As a result, it doesn’t matter whether \(E_j(t)\) or \(E_j(t)\) is used in \(\gamma_{j,i}(t)\).

Now let \(E_j(S_{i,k}^{(t)}) \neq E_j(S_{i,k}^{(t-1)})\), i.e. \(\exists_{\text{init}} S_{i,k}^{(t)} = m \cdot T_j\). As a result, an additional activation of \(t_i\) will be taken into account when determining \(S_{i,k}^{(t)}\), irrespective of using either \(E_j(t)\) or \(E_j(t)\) in \(\gamma_{j,i}(t)\). Together, these two cases prove the lemma.

We therefore conclude that, apart from the number of job activations of \(t_i\), the information in Table II also holds for \(t_i\) when \(B_i = 0\).

D. Identifying the task causing the largest blocking delay

A nice property of FPTS is that just one job of lower priority is able to cause blocking delays. In the presence of CRPD, however, the largest computation time among the blocking tasks does not necessarily result in the largest worst-case response time.

Example 6. We reconsider \(T_3\) of Example 3. Without CRPD, the blocking of \(t_3\) due to \(t_1\) and \(t_4\) is the same because \(C_3 = C_4\), i.e. \(B_3 = \max(0, \max(C_3, C_4)) = 1\). The blocking including CRPD may be different, however, due to different UCBs of \(t_3\) and \(t_4\) and the ECBs of \(t_1\). Even a smaller computation time of a blocking task may result in a larger overall blocking effect when CRPD is included.

For the case with blocking (\(B_i \neq 0\)), we therefore need a more complex procedure to compute response times. Our new procedure determines the values for \(L_i\), \(S_{i,k}\), \(F_{i,k}\), and \(R_i\) with CRPD by taking the maximum value over all tasks that may block \(t_i\).

E. Termination of the iterative procedure for \(L_i\)

Termination of the iterative procedure to determine \(L_i\) is no longer guaranteed when \(U < 1\), because the CRPD is not taken into account in the utilization \(U\). To address this problem,
we first observe that by definition every level-i active period, with \( 1 \leq i \leq n \), is contained in a level-n active period [14]. Hence termination for \( L_a \) guarantees termination for \( L_i \) for all \( 1 \leq i \leq n \). Next, the lowest priority task \( \tau_i \) cannot be blocked. As a result, when \( L_a \) exceeds the least common multiple (LCM) of the periods of the task set \( T \), the termination procedure will not terminate. This is because at the LCM the activation pattern is repeated and if \( L_a \) did not terminate at the LCM then there is pending load pushed across the LCM boundary. By integrating CRPD into the analysis, the effective utilization with CRPD is apparently larger than 1. The set is therefore considered unschedulable when \( L_a \) exceeds the LCM.

V. FPTS WITH CRPD: PRE-EMPTING TASKS

In this section, we consider the ECB-Only approach, i.e. focus only on the pre-empting tasks. Because the worst-case hold time \( H_i \) only plays a role for pre-empted tasks, we ignore \( H_i \) in this section. In order to extend the equations for \( L_i \), \( S_i \) and \( F_i \) for FPTS with a term \( \gamma_i(t) \), we must adapt \( \gamma_i^{\text{cb-o}}(t) \) by considering the tasks affected by \( \tau_j \) (see the row affected by \( \tau_j \) in Table II). As shown in Table II, the tasks being affected by pre-emptions are the same for the intervals \([0, L_i], [0, S_i], \) and \([0, F_i] \), but differ from the tasks being affected under FPPS with constrained deadlines. We therefore generalize, i.e. redefine, the set of tasks \( \text{aff}(\pi_i, \pi_j) \) for FPTS to:

\[
\text{aff}(\pi_i, \pi_j) \overset{\text{def}}{=} \text{lt}(\pi_j) \cap \text{hep}(\pi_i). \tag{19}
\]

Equation (19) for FPTS specializes to (9) for FPPS because \( \text{lp}(\pi_j) = \text{lt}(\pi_j) \) for FPPS.

To determine the worst-case response time \( R_i \) of \( \tau_i \), we can then reuse (5). In the subsections below, we consider the cases without and with blocking separately.

A. Worst-case length \( L_i \)

1) Tasks without blocking: For the case \( B_i = 0 \), we can find an upper bound for \( L_i \) with CRPD by extending (1) with \( \gamma_i(t) \), similar to the extension of \( R_i \) in (7), i.e.

\[
L_i = \sum_{\pi_j \in \text{ aff}(\pi_i)} \left( E_j(L_i) \cdot C_j + \gamma_j(L_i) \right). \tag{20}
\]

For the ECB-Only approach, we can subsequently reuse (8) for \( \gamma_i^{\text{cb-o}}(t) \) with \( \text{aff}(\pi_i, \pi_j) \) as defined in (19).

2) Tasks with blocking: For the case \( B_i \neq 0 \), we rewrite (1) for \( L_i \) by distributing addition over the inner-max operation in equation (2) for \( B_i \) and subsequently extending the equation for CRPD as explained in Section IV-D, i.e.

\[
L_i = \max_{\pi_k} \left( C_k + \sum_{\pi_j \in \text{ aff}(\pi_i)} \left( E_j(L_i) \cdot C_j + \gamma_j(L_i) \right) \right). \tag{21}
\]

A subscript “-b” has been introduced in \( \gamma_i(L_i) \) to capture the CRPD related to the blocked task \( \tau_b \). For the ECB-Only approach, \( \gamma_i^{\text{cb-o}}(t) \) is defined as

\[
\gamma_i^{\text{cb-o}}(t) = \begin{cases} \text{BRT} \cdot E_j(t) \cdot \text{ECB} & \text{if } \text{aff}(\pi_i, \pi_j) \neq \emptyset \text{ and } b \in \text{lt}(\pi_j), \\ 0 & \text{otherwise}. \end{cases} \tag{22}
\]

Compared to (8) for FPPS, the first clause for \( \gamma_i^{\text{cb-o}}(t) \) in (22) for FPTS has been extended with \( b \in \text{lt}(\pi_j) \), because \( \tau_j \) may in that case also pre-empt task \( \tau_i \). Note that \( \text{lt}(\pi_j) \cap \text{hep}(\pi_j) \) in Table II is equal to \( \text{aff}(\pi_i, \pi_j) \cup \{\text{lt}(\pi_j) \cap \text{hep}(\pi_j)\} \) in (22).

B. Worst-case start time \( S_i \)

1) Tasks without blocking: Similar to \( L_i \), we extend equation (3) for \( S_i \) with a term \( \gamma_i(t) \) to include CRPD, i.e.

\[
S_i = kC_i + \sum_{\pi_j \in \text{ aff}(\pi_i)} \left( E_j(S_i) \cdot C_j + \gamma_j(S_i) \right). \tag{23}
\]

Based on Lemma 2, we conclude that we can define \( \gamma_i(S_i) \) in terms of \( E_i(t) \) rather than \( E_i(t) \). Hence, we can also reuse \( \gamma_i^{\text{cb-o}}(t) \) from (8) for the ECB-Only approach, with \( \text{aff}(\pi_i, \pi_j) \) as defined in (19), similar to \( L_i \).

2) Tasks with blocking: We extend \( S_i \) with an additional subscript "-b" and \( \gamma_i^{\text{cb-o}}(t) \) in terms of \( E_i(t) \) rather than \( E_i(t) \). Hence, we can also reuse \( \gamma_i^{\text{cb-o}}(t) \) from (8) for the ECB-Only approach, with \( \text{aff}(\pi_i, \pi_j) \) as defined in (19).

For the ECB-Only approach, we can reuse \( \gamma_i^{\text{cb-o}}(t) \) from (22), similar to \( L_i \).

C. Worst-case finalization time \( F_i \)

1) Tasks without blocking: We can extend (18) with \( \gamma_i(t) \) terms complementing \( E_j(F_i) \cdot C_j \) and \( E_j(S_i) \cdot C_j \), i.e.

\[
F_i = S_i + C_i + \sum_{\pi_j \in \text{ aff}(\pi_i)} \left( E_j(F_i) - E_j(S_i) \right) \cdot C_j
+ \sum_{\pi_j \in \text{ aff}(\pi_i)} \left( \gamma_j(F_i) - \gamma_j(S_i) \right). \tag{25}
\]

Similar to \( L_i \) and \( S_i \), we use (8) for \( \gamma_i^{\text{cb-o}}(t) \), with \( \text{aff}(\pi_i, \pi_j) \) as defined in (19).

2) Tasks with blocking: Similar to \( S_i \), we add a subscript "-b" to \( F_i \). Similar to the case \( B_i = 0 \), we expand the formula with terms for CRPD, i.e.

\[
F_i = S_i + C_i + \sum_{\pi_j \in \text{ aff}(\pi_i)} \left( E_j(F_i) - E_j(S_i) \right) \cdot C_j
+ \sum_{\pi_j \in \text{ aff}(\pi_i)} \left( \gamma_j(F_i) - \gamma_j(S_i) \right). \tag{26}
\]

Similar to \( L_i \) and \( S_i \), we apply (22) for \( \gamma_i^{\text{cb-o}}(t) \). To compute \( F_i \), we take the maximum value over all tasks that may block \( \tau_i \), similar to \( L_i \) and as explained in Section IV-D, i.e.

\[
F_i = \max_{\tau_b \in \text{ aff}(\pi_i)} F_i^{\text{b}}. \tag{27}
\]

VI. FPTS WITH CRPD: PRE-EMPTED TASKS

In this section, we consider the UCB-Only Multiset approach, i.e. focus on the pre-empted tasks. In this case, the row #-jobs in Table II also plays a role. As shown in Table II, a case distinction is needed to capture the tasks that are being pre-empted, and these cases differ for \([0, H_i], [0, L_i], [0, S_i] \) and \([0, F_i] \). As a consequence, this section presents dedicated adaptations of \( \gamma_i(t) \) and \( M_i(t) \), for each interval. For case
of presentation, we only consider the case where tasks may experience blocking. The other case is similar.

A. Worst-case hold time $H_i$

We can find an upper bound for $H_i$ with CRPD by extending (16) with $\gamma_i(t)$, similar to the extension of $R_i$ with $\gamma_i(t)$, i.e.

$$H_i = C_i + \sum_{j \in \text{hp}(i)} E_j(H_i) \cdot C_j - \gamma_j(H_i).$$  

(28)

Although we can apply $\gamma_i^M(t)$ in (10) for the UCB-Only Multiset approach, we need to adapt the definition of $M^{wcb-o}_i(t)$ in (11) to prevent pessimism, as discussed in Sections IV-B and IV-C.

Firstly, worst-case hold times are to be considered for preempted tasks, rather than worst-case response times. Secondly, exactly one job of task $\tau_i$ needs to be considered rather than $E_i(t)$ jobs. These two adaptations of (11) result in

$$M^{wcb-o}_i(t) = \bigcup_{h \in \text{aff}(\tau_i)} \left( \bigcup_{E_j(H_i) = 0(t)} \left[ \text{UCB}_a \right] \cup \left( \bigcup_{E_j(H_i) > 0(t)} \left[ \text{UCB}_b \right] \right) \right) \text{if } i \in \text{lt}(\tau_i),$$

$$\emptyset \text{ otherwise.}$$

(29)

Because task $\tau_i$ is treated in a separate clause, we need an alternative $\text{aff}(\tau_i)$ for $\text{aff}(\tau_i)$ excluding $(i)$, i.e.

$$\text{aff}(\tau_i, \pi_j) \overset{\text{def}}{=} \text{lt}(\pi_j) \cap \text{hp}(\pi_i) = \text{aff}(\tau_i, \pi_j) \setminus \{i\}.$$  

(30)

B. Worst-case length $L_i$

Similar to the ECB-Only approach, we can use (21) to find an upper bound for $L_i$ by extending (10) for $\gamma_i^M(t)$ with a subscript $b$ for the blocking task $\tau_b$, with $b \in \text{b}(i)$.

$$\gamma_i^M(t) = \text{BRT} \cdot \sum_{t=1}^{E_i(t)} \left[ \text{sort} \left( M_{i,b}(t) \right) \right] [f].$$

(31)

The definition of $M^{wcb-o}_i(t)$ in (11) also needs to be extended with a subscript $b$, to consider exactly one blocking job of $\tau_b$ rather than $E_b(t)$ jobs.

$$M^{wcb-o}_i(t) = \bigcup_{h \in \text{aff}(\tau_i)} \left( \bigcup_{E_j(H_i) = 0(t)} \left[ \text{UCB}_a \right] \cup \left( \bigcup_{E_j(H_i) > 0(t)} \left[ \text{UCB}_b \right] \right) \right) \text{if } b \in \text{lt}(\tau_b),$$

$$\emptyset \text{ otherwise.}$$

(32)

The pre-condition $b \in \text{b}(i)$ for $M^{wcb-o}_i(t)$ is taken into account by the max in (21). The definition of $M^{wcb-o}_i(t)$ is based on $\text{aff}(\tau_i)$, rather than $\text{aff}(\tau_i)$, because the number of jobs of $\tau_i$ are not known a-priori. Moreover, the definition contains the worst-case hold times of $\tau_b$ and $\tau_i$ rather than their worst-case response times to avoid pessimism.

C. Worst-case start time $S_{i,k}$

As well as considering exactly one job of task $\tau_b$, the definitions of $\gamma_i^M(t)$ and $M^{wcb-o}_i(t)$ are further extended for $S_{i,k}$ to consider exactly $k$ jobs of $\tau_i$ (see Table II), i.e.

$$\gamma_i^{M_{i,k,b}}(t) = \text{BRT} \cdot \sum_{t=1}^{E_i(t)} \left[ \text{sort} \left( M_{i,k,b}(t) \right) \right] [f].$$

(33)

and

$$M^{wcb-o}_{i,k,b}(t) = \bigcup_{h \in \text{aff}(\tau_i)} \left( \bigcup_{E_j(H_i) = 0(t)} \left[ \text{UCB}_a \right] \cup \left( \bigcup_{E_j(H_i) > 0(t)} \left[ \text{UCB}_b \right] \right) \right) \text{if } i \in \text{lt}(\tau_i),$$

$$\emptyset \text{ otherwise.}$$

(34)

Similar to $H_i$, task $\tau_i$ is once again treated by a separate clause, necessitating the usage of $\text{aff}(\tau_i)$ rather than $\text{aff}(\tau_i)$. Moreover, $M^{wcb-o}_{i,k,b}(t)$ is based on the worst-case hold times of $\tau_b$, $\tau_i$, and $\tau_b$ rather than their worst-case response times.

Similarly to the ECB-Only approach, a subscript "b" is added to $S_{i,k}$. Moreover, the equation of $S_{i,k}$ in (3) is extended with $\gamma_i^{M_{i,k,b}}(t)$ as follows:

$$S_{i,k,b} = C_b + kC_i + \sum_{j \in \text{hp}(\tau_i)} \left( E_j(S_{i,b,k}) \cdot C_j + \gamma_i^{M_{i,k,b}}(S_{i,k,b}) \right).$$

(35)

D. Worst-case finishing time $F_{i,k}$

As indicated in Table II, exactly $k + 1$ jobs of $\tau_i$ need to be considered for $F_{i,k}$. Moreover, we need to split the set of tasks $\text{hp}(\tau_i)$ into two subsets for $F_{i,k}$, i.e. the set $\text{hp}(\tau_i) \setminus \text{hp}(\tau_b) = \text{set of tasks that can be blocked by } \tau_i$ and the set $\text{hp}(\tau_b)$ that cannot be blocked by $\tau_i$. The former set can execute and experience pre-emptions in $[0, S_{i,b}]$, whereas the latter set can execute and experience pre-emptions in $[0, F_{i,b}]$. To take the proper number of activations of tasks in these two sets into account, we use two parameters $t_i$ and $t_f$ for $\gamma_i^{M_{i,k,b}}(t_i)$ and $M^{wcb-o}_{i,k,b}(t_f)$, i.e.

$$\gamma_i^{M_{i,k,b}}(t_i, t_f) = \text{BRT} \cdot \sum_{t=1}^{E_i(t)} \left[ \text{sort} \left( M_{i,k,b}(t_i, t_f) \right) \right] [f].$$

(36)

and

$$M^{wcb-o}_{i,k,b}(t_i, t_f) = \bigcup_{h \in \text{aff}(\tau_i)} \left( \bigcup_{E_j(H_i) = 0(t_i)} \left[ \text{UCB}_a \right] \cup \left( \bigcup_{E_j(H_i) > 0(t_i)} \left[ \text{UCB}_b \right] \right) \right) \text{if } i \in \text{lt}(\tau_i),$$

$$\emptyset \text{ otherwise.}$$

(37)

Similar to the ECB-Only approach, $F_{i,k}$ is extended with a
Let subscript “b” and \( \gamma_{i,k,b} \) terms, i.e.,

\[
F_{i,k} = S_{i,k} + C_j + \sum_{\nu \in \text{rep}(\nu)} \left( E_j(F_{i,k},h) - E_j(S_{i,k},h) \right) \cdot C_j + \sum_{\nu \in \text{rep}(\nu)} \left( \gamma_{i,k,b}(S_{i,k},h), F_{i,k,b}, \gamma_{i,k,b}(S_{i,k},h) \right).
\] (38)

The term \( \gamma_{i,k,b}(S_{i,k},h) \) in (38) prevents the cache-related preemption costs already covered in (35) for \( S_{i,k} \) being accounted for twice.

We may subsequently determine \( F_{i,k} \) by (27) and can derive \( R_t \) through (3) as before.

VII. FPTS with CRPD: PRE-EMPTING AND PRE-EMPTED TASKS

In this section, we consider the ECB-Union and UCB-Union Multiset approaches, i.e., we consider both the pre-empting and the pre-empted tasks. As described in Section III-B for FPPS with CRPD, the definitions of the multiset for the ECB-Union and UCB-Union Multiset approaches can be derived from the definition of the multiset for the UCB-Only Multiset approach. A similar derivation applies for FPTS with CRPD. We therefore only consider the definition of the multiset \( M_{i,k}^{b}(t,f) \) and \( M_{i,k}^{b}(t_f) \) for the worst-case finalization time \( F_{i,k} \) for the case with blocking. The derivation of the definitions for the case without blocking and for the worst-case length \( L_t \) and worst-case start time \( S_{i,k} \) are similar.

A. ECB-Union Multiset approach

The ECB-Union Multiset approach considers the pre-emption cost of pre-empted tasks for every pre-empted task individually. Similar to FPPS with CRPD, the definition of the multiset of the UCB-Only Multiset approach is extended by including the UCBs of every affected task with \( \bigcup_{\nu \in \text{rep}(\nu)} \text{ECB}_{\nu} \), e.g., from (37) for \( M_{i,k}^{b}(t,f) \) we derive

\[
M_{i,k}^{b}(t,f) = \bigcup_{\nu \in \text{rep}(\nu)} \left( \bigcup_{\lambda \in \text{rep}(\lambda)} \text{UCB}_{\lambda} \cap \bigcup_{\mu \in \text{rep}(\mu)} \text{ECB}_{\mu} \right).
\] (39)

The equations for \( \gamma_{i,k,b}(t_f) \) in (36), \( F_{i,k} \) in (38), and \( F_{i,k} \) in (27) can be reused for the ECB-Union Multiset approach.

B. UCB-Union Multiset approach

For the UCB-Union Multiset approach, first a multiset \( M_{i,k}^{b}(t_f) \) is formed. Similar to FPPS with CRPD, the definition for \( M_{i,k}^{b}(t_f) \) can be derived from (37) for \( M_{i,k}^{b}(t_f) \) by removing all cardinality operators, i.e.,

\[
M_{i,k}^{b}(t_f) = \bigcup_{\nu \in \text{rep}(\nu)} \left( \bigcup_{\lambda \in \text{rep}(\lambda)} \text{UCB}_{\lambda} \right)
\] (40)

Similar to FPPS with CRPD, the definition of \( \gamma_{i,k}^{b}(t_f) \) is given in terms of the size of the multi-set intersection of \( M_{i,k}^{b}(t_f) \) and \( M_{i,k}^{b}(t_f) \), i.e.,

\[
\gamma_{i,k,b}(t_f) = \text{BRT} \left( M_{i,k}^{b}(t_f) \cap M_{i,k}^{b}(t_f) \right).
\] (41)

where \( M_{i,k}^{b}(t_f) \) is defined in (14). The equations for \( F_{i,k} \) in (38) and \( F_{i,k} \) in (27) also apply for the UCB-Union Multiset approach.

C. Composite approach

The ECB-Union Multiset and UCB-Union Multiset approaches can be combined into a simple composite approach that dominates both [3]. This composite approach uses

\[
R_t = \min(R_t^{\text{ECB}}, R_t^{\text{UCB}}),
\] (42)

where \( R_t^{\text{ECB}} \) and \( R_t^{\text{UCB}} \) are the worst-case response times of task \( t \), using the ECB-Union Multiset approach and the UCB-Union Multiset approach, respectively. Since this composite approach is the most effective analysis for CRPD, we use it in our evaluation.

VIII. AN OPTIMAL THRESHOLD ASSIGNMENT ALGORITHM

In [33] an optimal threshold assignment algorithm (OTA) for a set \( T \) scheduled under FPTS without CRPD is described, which assumes that priorities of tasks are given, i.e. it finds the (minimum) pre-emption thresholds achieving schedulability of \( T \) under FPTS, if such an assignment exists. The algorithm traverses the tasks in descending priority order, exploiting the property that the schedulability test for task \( t_i \) is independent of the pre-emption thresholds of tasks with a priority higher than \( t_i \). For FPTS with CRPD this property does not hold. As an example, a task \( t_j \) may affect a task \( t_k \), with \( j,h \in \text{rep}(\pi) \), when the threshold \( \theta_j \) of \( t_j \) is lower than the priority \( \pi_t \) of \( t_j \). The algorithm subsequently presented in [31] can determine the maximum pre-emption thresholds of tasks, taking a threshold assignment for which the set is schedulable as input.

This section presents an OTA algorithm for FPTS with CRPD, yielding the maximum pre-emption thresholds of tasks when the set is schedulable, effectively minimizing pre-emption costs. The algorithm also assumes that priorities of tasks are
given and traverses the tasks in descending priority order. It exploits the property that once a task \( t_i \) is schedulable, it remains schedulable when the pre-emption threshold \( \theta_i \) of a task \( t_j \) with a priority lower than task \( t_i \) is reduced and \( \theta_j \) either was or becomes lower than priority \( \pi_i \).

Our OTA algorithm (see Algorithm 1) uses an auxiliary set \( \Theta = \{\theta_1, \theta_2, \ldots, \theta_n\} \) of maximum pre-emption thresholds next to a set \( \Xi = \{\xi_1, \xi_2, \ldots, \xi_n\} \) of assigned pre-emption thresholds. Upon initialization, all values in \( \Theta \) are set to the highest priority \( \pi_1 \) (line 2), i.e. tasks are non-pre-emptive and therefore experience minimal CRPD. The algorithm traverses the tasks in descending priority order (lines 5-23). When it considers a task \( t_i \), it first assigns its maximum pre-emption threshold \( \theta_i \) to \( \theta_i \) (line 7). Next, it tests schedulability of \( t_i \) without any blocking and returns unschedulable when the test fails (line 9). Otherwise, it tests schedulability of \( t_i \) with blocking by considering each lower priority task \( t_j \) in isolation (lines 11-22). It decreases the maximum pre-emption threshold \( \tilde{\theta}_j \) of task \( t_j \) and only if \( t_i \) is unschedulable due to blocking by task \( t_j \) (lines 17-19). In that case, \( \theta_i \) is increased to the highest priority of all tasks with a priority lower than \( t_i \), i.e. \( \pi_{i+1} \) of all \( t_j \). This may increase the CRPD of tasks with a priority lower than \( t_i \) but does not affect the schedulability of tasks with a priority higher than \( \pi_i \). Hence, when the algorithm returns schedulable, i.e. the task set is schedulable, it has assigned the maximum pre-emption threshold to each task.

**Theorem 1.** Given a set of tasks \( T \) and a priority assignment \( \Pi \), the OTA algorithm (Algorithm 1) assigns the maximum pre-emption thresholds \( \Theta \subseteq \Pi \) to tasks achieving schedulability, if such an assignment exists.

A proof of correctness and detailed explanation of our OTA algorithm using invariants are given in the next subsection.

**A. Correctness and proof of OTA algorithm**

Our algorithm is based on two invariants, which use U = \( \{\pi_1, \pi_2, \ldots, \pi_n\} \) to denote the set of priorities and \( T_m^H \) to denote the subset of \( m \) highest priority tasks with 0 \( \leq m \leq n \), i.e. \( T_0^H = \emptyset, T_1^H = \{t_i | \pi_i \in \text{dep}(n_i)\} \) for 1 \( \leq m \leq n \), and \( T_m^H = T \).

If the following main invariant holds for \( T_r \), then \( \Theta \) contains the maximum pre-emption thresholds for which all tasks in \( T \) are schedulable, where \( \Theta = \Theta \subseteq \Pi \).

**Invariant 1.** Given a subset \( T_m^H \) of \( m \) highest priority tasks

1. the set \( \Theta \) contains the maximum pre-emption threshold of each task such that all tasks in \( T_m^H \) meet their deadlines, i.e. \( \forall t_i \in T_m^H \), where \( \Theta \subseteq \Pi \).
2. the set \( \Theta \) contains the assigned pre-emption threshold of \( t_j \) if \( t_j \in T_m^H \), i.e. \( \theta_j = \tilde{\theta}_j \), and it contains the priority of \( t_j \) if \( t_j \notin T_m^H \), i.e. \( \theta_j > \pi_j \).

The variables in \( \Theta \) and \( \Theta \) are initialized to the highest (non-pre-emptive) priority \( \pi_1 \) (line 2) and the (fully pre-emptive) priority of the corresponding task (line 3), respectively. As a result, Invariant 1 holds for the empty set \( T_0^H \).

Next, the algorithm traverses the tasks in descending priority order (lines 5-23). When a task \( t \) is considered (line 5),

**Algorithm 1: OptimalThresholdAssignment(\{t_1, \ldots, t_n\})**

**Input:** A task set \( T = \{t_1, \ldots, t_n\} \) with \( (G, T, D, m) \), \( \Pi \), \( \Theta \subseteq \Pi \).
**Output:** Task set schedulable and \( \theta_i, \pi_i \in T \), where \( \Theta \subseteq \Pi \).

1. for each \( t_i \) do
   2. \( \tilde{\theta}_i \leftarrow \pi_i \); (Init. the max. threshold \( \tilde{\theta}_i \) with the highest priority \( \pi_i \).
   3. \( \theta_i \leftarrow \pi_i \); (Init. the threshold \( \theta_i \) with the priority \( \pi_i \) of \( t_i \).
   4. end for
**Invariant 1 holds for** \( T_0^H \).
5. for each \( t_i \) (from highest to lowest priority) do
   6. **(Loop invariant: Invariant 1 holds for** \( T_i^H \)).
   7. \( \tilde{\theta}_i \leftarrow \theta_i \); (Assign max. threshold \( \tilde{\theta}_i \) to \( \theta_i \) of \( t_i \).
   8. Compute \( R_i \) (without blocking, i.e. \( C_i = 0 \)).
   9. if \( R_i > D_i \) then return unschedulable end if
   10. **(Invariant 2 holds for** \( t_i \) and \( T_i^H \)).
   11. for each \( t_j \) with \( \ell \in \text{lep}(n_j) \) (from highest to lowest) do
   12. **(Loop invariant: Invariant 2 holds for** \( t_j \) and \( T_j^H \)).
   13. (Test schedulability of \( t_i \) when blocked by \( t_j \) based on \( \tilde{\theta}_i \)).
   14. \( \tilde{\theta}_i \leftarrow \tilde{\theta}_i \); (Temporarily assign max. threshold \( \tilde{\theta}_i \) to \( \theta_i \) of \( t_i \).
   15. Re-compute \( R_i \) (with blocking, i.e. \( C_i = C_i \)).
   16. **(Establish Invariant 2 for** \( t_i \) and \( T_i^H \)).
   17. if \( R_i > D_i \) then (Disallow blocking by \( t_j \)).
   18. \( \tilde{\theta}_i \leftarrow \pi_i \).
   19. end if
   20. (Reset the threshold \( \theta_i \) of \( t_i \) (re-establish Invariant 1)).
   21. \( \theta_i \leftarrow \pi_i \).
   22. **end for**
   23. **end for**
   24. return schedulable.

Invariant 1 holds for \( T_0^H \). First the pre-emption threshold of \( t_i \) is assigned its maximum value, i.e. \( \theta_i \) is set to \( \tilde{\theta}_i \) (line 7), and the schedulability of \( t_i \) without blocking is determined. If \( t_i \) is not schedulable, then the algorithm returns unschedulable (line 9), i.e. there does not exist a pre-emption threshold assignment making the set of tasks \( T_i^H \) schedulable. Otherwise 2) has been established for \( T_i^H \) and the inner-loop is entered.

The inner-loop (lines 11-22) considers each task \( t_i \) with a priority lower than \( t_i \), separately. The aim is to establish 1) for \( T_i^H \) based on the following invariant.

**Invariant 2.** Given a task \( t_i \) and a subset \( T_i^H \) with \( \ell \in \text{lep}(n) \), the set \( \Theta \) contains the maximum pre-emption threshold for each task, where \( \Theta \subseteq \Pi \), such that

1. all tasks in \( T_i^H \) are schedulable, and
2. \( t_i \) is schedulable when only the set \( T_i^H \) is considered, i.e. when all tasks in \( T \setminus T_i^H \) are ignored.

If this invariant holds for \( t_i \) and \( T \) then \( \Theta \) contains the maximum pre-emption thresholds for which all tasks in \( T_i^H \) are schedulable, where \( \Theta \subseteq \Pi \). Invariant 1 holds for \( T_i^H \).

Before the inner-loop, Invariant 2 holds for \( t_i \) and \( T_i^H \), and when a task \( t_i \) is considered (line 11), it holds for \( t_i \) and \( T_i^H \). When \( t_i \) remains schedulable when blocked by \( t_i \), \( \theta_i \) remains unchanged. Otherwise \( \tilde{\theta}_i \) is set to the priority \( \pi_{i+1} \) of task \( t_{i+1} \), i.e. the highest priority in \( \Pi \) for which \( t_i \) is not blocked by \( t_i \). This may increase the CRPD of tasks with a priority lower than \( t_i \), but does not affect the schedulability of tasks with a priority higher than \( t_i \). Note that it doesn’t make sense to decrease the threshold of \( t_i \) to a priority higher than or equal to the priority of \( t_i \), because the CRPD experienced by \( t_i \) remains at best the same and may even increase due to additional pre-emptions during the execution of a job of \( t_i \). Invariant 2 has therefore
been established for $\mathcal{T}^H$.

At each iteration of the outer-loop, the set $\mathcal{T}^H_m$ of Invariant 1 is increased by one task. Similarly, at each iteration of the inner-loop, the set $\mathcal{T}^L_i$ of Invariant 2 is increased by one task. Hence, the algorithm terminates with either schedulable and a set of maximum pre-emption thresholds that deem the task set schedulable with the least possible CRPD or unschedulable, in which case no assignment of pre-emption thresholds achieving schedulability exists under the given priority assignment.

B. Algorithmic complexity

Algorithm 1 traverses the set of tasks (of size $n$) in descending priority order and it may then consider any lower-priority task (at most $n-1$ tasks). Hence, just like the algorithm in [33], our algorithm has $O(n^2)$ iterations. In each iteration, the response time analysis is applied, which has a pseudo-polynomial time complexity.

IX. Evaluation

We perform the same simulation studies as in [3] to compare the relative CRPD costs under FPTS, FPPS and FPNS. The results are compared with the scheduling analysis ignoring CRPD. We have therefore generated system configurations so that the results for FPTS without CRPD match those in [10, 15] and the results for FPPS with CRPD those in [3].

In our basic system configuration, we assume a cache with $N = 512$ cache sets and a total cache utilization of $U^C = 4$, i.e. the total number of ECBs of all tasks is $N \times U^C = 2048$. We then select the cache utilization $U^C_i$ of each task (the number of ECBs of a task, $|\text{ECB}_i|$) using UUniFast [11]. 40% of a task’s ECBs are also UCBs, i.e. $|\text{UCB}_i| = 0.4 \times |\text{ECB}_i|$. To compute the schedulability of a task set under CRPD, we compared the most effective approaches, i.e. the combination of the UCB-Union Multiset and the ECB-Union-Multiset, both for FPPS (see [3]) and FPTS (developed in this paper). For each experiment and for each parameter configuration, we generate a new set of 1,000 systems.

For each system, we generate $n = 10$ tasks which are assigned deadline monotonic priorities. The task deadlines are implicit, i.e. $D_i = T_i$, and the task periods $T_i$ are randomly drawn from the interval $[10, 1000]$ ms. The individual task utilizations $U_i$ (with $C_i = U_i \times T_i$) are generated using the

UUniFast algorithm [11]. The pre-emption thresholds of tasks are selected by our OTA algorithm (see Section VIII).

In our first experiment, we vary the task-set’s utilization and the block reload time is set to $5\mu s$. Figure 3 shows the ratio of task sets deemed schedulable. The relative performance improvement of FPTS compared to FPPS is strongly amplified when including the CRPD. In contrast, FPTS and FPPS without CRPD only differ in case of high task utilization (starting at $U = 0.85$) and at most by 20%. In the presence of CRPD, however, FPPS is only able to schedule half of all generated task sets at a utilization of $U = 0.8$, while FPTS is able to schedule more than 90%. FPTS only experiences a similar performance degradation at a considerably higher utilization, i.e. approximately at $U = 0.88$.

In the remaining experiments, we use as a metric the weighted schedulability ratio [7]. This metric takes a weighted average of the schedulability ratio over the entire utilization range $U \in [0, 1]$ using the utilization $(U)$ as a weight. It is defined as follows [7]. Let $S_S(\mathcal{T}, p)$ be the binary result (1 if schedulable, 0 otherwise) of schedulability test $\mathcal{T}$ for a task set $\mathcal{T}$ and parameter value $p$. Then:

$$W_S(p) = \frac{\sum_{\mathcal{T}} U \cdot S_S(\mathcal{T}, p)}{\sum_{\mathcal{T}} U},$$

where $U$ is the utilization of task set $\mathcal{T}$. This weighted schedulability ratio reduces what would otherwise be a 3-
dimensional plot to 2 dimensions [7]. Weighting the individual schedulability results by task-set utilization reflects the higher value placed on being able to schedule higher utilization task sets.

In the second experiment, we vary the block reload time (BRT) from 0µs to 640µs. Figure 4 shows the results. By increasing the BRT, we increase the CRPD and therefore penalise pre-emption. Consequently, the number of task sets deemed schedulable with FPPS with CRPD quickly drops to zero, while the performance of FPTS with CRPD converges to the performance of PPNS (as expected). It is interesting to see that FPTS with CRPD is able to deem more task sets schedulable than PPNS, even for an infinite BRT. The reason is as follows. If the sets of UCBs and ECBs of two tasks are completely disjoint (which may happen for randomly generated UCBs and ECBs of tasks), the CRPD of these two tasks pre-empting each other will remain zero. It is therefore possible that FPTS with CRPD outperforms PPNS, because not every pre-emption will be penalised.

In the third experiment, we vary the total cache utilization ($U_C$) from 0 to 160 and we reset the BRT to 8µs. Since the number of cache sets ($N$) remains the same, increasing $U_C$ means increasing the number of ECBs of tasks. Figure 5 shows again a weighted schedulability ratio. FPPS and FPTS with CRPD are both able to schedule considerably more task sets than PPNS. This is due to the fixed number of cache sets, which restricts the maximum possible pre-emption cost. At a total cache utilization of 40, each pre-emption evicts most of the cache contents which then need to be reloaded, hence further increases in cache utilization have little effect on schedulability.

In the fourth experiment, we vary the number of cache sets ($N$). Figure 6 shows the weighted schedulability ratio. As $N$ increases, the total number of ECBs being used by tasks also increases and, contrary to the third experiment, more of these ECBs fit into the cache. Hence, the pre-emption costs increase when more blocks need to be reloaded. The schedulability ratios of FPPS and FPTS with CRPD therefore decrease. FPPS will eventually be unable to schedule any tasks. The performance of FPTS, however, converges to the performance of PPNS, i.e., with PPNS task sets are unaffected by the increased pre-emption costs. We recall that FPTS with CRPD still outperforms PPNS, because, after assigning the highest possible pre-emption thresholds to tasks using our OTA, some of the remaining pre-emptions in the system may effectively come for free due to the limited overlap between the UCBs of some tasks and the ECBs of others.

In the fifth experiment, we vary the range of the task periods in steps of increasing orders of magnitude (see Figure 7). Since we generate computation times depending on the task periods, a larger range of the periods results in a larger computation time for some tasks. The performance of PPNS therefore quickly drops, because computation times of tasks with a large period may exceed the periods (and the constrained deadlines) of other tasks in the system. For the same reason, however, we may be unable to assign a pre-emption threshold to tasks with a large period and long computation time other than its regular priority. The performance of FPPS with CRPD therefore approaches the performance of FPTS with CRPD. At the other extreme, when the range of task periods is small, then FPTS with CRPD provides performance close to that of FPPS without CRPD. This is because with a small range of periods and deadlines, the OTA algorithm can set pre-emption thresholds such that most tasks cannot pre-empt each other, thus greatly reducing CRPD. Overall, FPTS provides consistently high performance irrespective of the range of task periods.

Finally, we increase the number of tasks (see Figure 8). This leads to an improved performance of FPTS with CRPD relative to FPPS with CRPD. There are two reasons for this: (i) as the
cache utilization remains constant, the ECRs per task decrease and (ii) by increasing the number of tasks, the individual task utilizations and execution times decrease, thus decreasing the potential blocking times. This gives the OTA algorithm more freedom to set pre-emption thresholds such that most tasks cannot pre-empt each other, again greatly reducing CRPD.

X. CONCLUSIONS

In this paper, we integrated analysis of cache related pre-emption delays (CRPD) into response time analysis for fixed priority scheduling of tasks with pre-emption thresholds (PPTS) and arbitrary deadlines. Further, we introduced an Optimized Threshold Assignment (OTA) algorithm that minimizes the effects of CRPD given an initial set of task priorities. The analysis we provided generalizes existing analysis for FFPS with constrained deadlines and CRPD described in [3], and covers the most effective approaches presented in that paper, in particular the ECB-Union and UCB-Union Multiset approaches.

We presented a comparative evaluation of the performance of the schedulability tests for PPTS and FFPS with and without CRPD. Interestingly, we found that the theoretical performance advantage that FFPS has over FFPS when there is no CRPD is extended significantly when CRPD are taken into account. Further, even when the overheads (block reload times) affecting CRPD are increased to very high levels, PPTS still retains a performance advantage over FFPS which it also dominates. This is due to the limited overlap between the UCBs of some tasks and the ECBs of others, meaning that some pre-emptions effectively come for free (i.e. no CRPD).

Our results indicate that PPTS can rightly be viewed as a potential successor to FFPS as a defacto standard in industry, where it is already supported by both OSEK [1] and AUTOSSAR [2] compliant operating systems.

There are a number of ways in which this work can be extended. Firstly, the layout of tasks in memory has already been shown to have a substantial effect on CRPD [27]. The combination of pre-emption thresholds and task layout provides additional opportunities for CRPD reduction. Secondly, our OTA algorithm assumes that task priorities are provided. The problem of optimally assigning both priorities and thresholds using a computationally tractable method remains open.

REFERENCES
