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

Reinder J. Bril[a,b], Sebastian Altmeyer[c], Martijn M.H.P. van den Heuvel[b], Robert I. Davis[d], Moris Behnam[a]
[a]Mälardalen University (MDH), Sweden, [b]Technische Universiteit Eindhoven (TU/e), The Netherlands
[c]University of Amsterdam (UvA), The Netherlands, [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 FPPS and FPTS, 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 defacto 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 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*[1], and deployed in the automotive industry. FPTS may therefore be used for legacy code and viewed as an evolutionary successor of FPPS as defacto 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 $\mathcal{T}$ 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. τ_1 has the highest and τ_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 $\mathrm{hp}(\pi)$ (and $\mathrm{lp}(\pi)$) to denote the set of tasks with priorities higher than (lower than)

[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-OSEK and RTA-OS operating systems, which have been deployed in 50 to 55 million new ECUs per year since 2008 [19].

π. Similarly, we use hep(π) (and lep(π)) to denote the set of tasks with priorities higher (lower) than or equal to π.

Each task τ_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 τ_i is given by C_i/T_i, and the *utilization* U of the set of tasks $\mathcal{T}$ by $\sum_{1\leq i\leq n} U_i$. An activation of a task is also termed a *job*.

For notational convenience, we introduce $E_j(t) = \lceil t/T_j \rceil$ and $E^*_j(t) = \left(1 + \lfloor t/T_j \rfloor\right)$ to represent the maximum number of activations of τ_j in an interval $[x, x+t)$ and $[x, x+t]$, respectively, where both intervals have a length t.

B. Refined model for FPTS

In FPTS, each task τ_i has a *pre-emption threshold* θ_i, where $\pi_1 \geq \theta_i \geq \pi_i$. When τ_i is executing, it can only be pre-empted by tasks with a priority higher than θ_i. Note that we have FPPS and FPNS as special cases when $\forall_{1\leq i\leq n}\theta_i = \pi_i$ and $\forall_{1\leq i\leq n}\theta_i = \pi_1$, respectively.

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

TABLE I
NOTATIONS FOR VARIOUS SETS OF INDICES OF TASKS.

classic notations for FPPS	additional notations for FPTS
$\text{hep}(\pi) \stackrel{\text{def}}{=} \{h \mid \pi_h \geq \pi\}$	$\text{het}(\pi) \stackrel{\text{def}}{=} \{h \mid \theta_h \geq \pi\}$
$\text{lp}(\pi) \stackrel{\text{def}}{=} \{\ell \mid \pi > \pi_\ell\}$	$\text{lt}(\pi) \stackrel{\text{def}}{=} \{\ell \mid \pi > \theta_\ell\}$
$\text{hp}(\pi) \stackrel{\text{def}}{=} \{h \mid \pi_h > \pi\}$	$\text{b}(i) \stackrel{\text{def}}{=} \text{lp}(\pi_i) \setminus \text{lt}(\pi_i)$
$\text{lep}(\pi) \stackrel{\text{def}}{=} \{\ell \mid \pi \geq \pi_\ell\}$	

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 *aiT 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* τ_1 *and* τ_2*. The memory blocks of* τ_1 *map to cache set* 0, 1 *and* 2*. Only* τ_1*'s memory block mapping to cache set* 1 *is useful, i.e.* $ECB_1 = \{0, 1, 2\}$ *and* $UCB_1 = \{1\}$*. The memory blocks of* τ_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 of task* τ_1 *pre-empting task* τ_2 *is thus given as follows:*

$$|ECB_1 \cap UCB_2| \cdot BRT = |\{1, 2\}| \cdot BRT = 2 \cdot BRT.$$

The *cache utilization* U^C_i of task τ_i is given by $|\text{ECB}_i|/N$, where $|\text{ECB}_i|$ denotes the number of ECBs of τ_i and N denotes the number of cache sets. The *total cache utilization* U^C of the set of tasks $\mathcal{T}$ is given by $\sum_{1\leq i\leq n} U^C_i = \sum_{1\leq i\leq n} |\text{ECB}_i|/N$.

III. RECAP OF RESPONSE TIME ANALYSIS FOR FPPS AND FPTS

This section starts with a recapitulation of the exact schedulability analysis for FPTS, 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. FPTS with arbitrary deadlines (without CRPD)

A set $\mathcal{T}$ of tasks is schedulable if and only if for every task $\tau_i \in \mathcal{T}$ its worst-case response time R_i is at most equal to its deadline D_i, i.e. $\forall_{1\leq i\leq n} R_i \leq D_i$. 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

$$L_i = B_i + \sum_{\forall j \in \text{hep}(\pi_i)} E_j(L_i) \cdot C_j, \tag{1}$$

where B_i denotes the worst-case blocking of task τ_i, given by

$$B_i = \max\left(0, \max_{\forall b \in \text{b}(i)} C_b\right). \tag{2}$$

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 τ_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

$$S_{i,k} = \begin{cases} B_i + kC_i + \sum\limits_{\forall j \in \text{hp}(\pi_i)} E_j(S_{i,k}) \cdot C_j & \text{if } B_i > 0 \\ kC_i + \sum\limits_{\forall j \in \text{hp}(\pi_i)} E^*_j(S_{i,k}) \cdot C_j & \text{if } B_i = 0 \end{cases} \tag{3}$$

and

$$F_{i,k} = S_{i,k} + C_i + \begin{cases} \sum\limits_{\forall j \in \text{hp}(\theta_i)} \left(E_j(F_{i,k}) - E_j(S_{i,k})\right) \cdot C_j & \text{if } B_i > 0 \\ \sum\limits_{\forall j \in \text{hp}(\theta_i)} \left(E_j(F_{i,k}) - E^*_j(S_{i,k})\right) \cdot C_j & \text{if } B_i = 0 \end{cases}. \tag{4}$$

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 τ_i is now given by

$$R_i = \max_{0 \leq k < E_i(L_i)} \left(F_{i,k} - k \cdot T_i\right). \tag{5}$$

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 τ_i is given by the smallest positive solution [24, 5] of

$$R_i = C_i + \sum_{\forall j \in \mathrm{hp}(\pi_i)} E_j(R_i) \cdot C_j. \tag{6}$$

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

$$R_i = C_i + \sum_{\forall j \in \mathrm{hp}(\pi_i)} \left(E_j(R_i) \cdot C_j + \gamma_{i,j}(R_i) \right), \tag{7}$$

where $\gamma_{i,j}(R_i)$ represents the cache-related pre-emption cost due to all jobs of a higher priority pre-empting task τ_j executing within the worst-case response time of task τ_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 and UCB-Union Multiset approaches consider a combination of pre-empting and pre-empted tasks.

1) ECB-Only approach: For this case, $\gamma_{i,j}(t)$ is given by[2]

$$\gamma^{\text{ecb-o}}_{i,j}(t) = \begin{cases} \mathrm{BRT} \cdot E_j(t) \cdot \left|\mathrm{ECB}_j\right| & \text{if } \mathrm{aff}(\pi_i, \pi_j) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}, \tag{8}$$

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

$$\mathrm{aff}(\pi_i, \pi_j) \stackrel{\text{def}}{=} \mathrm{lp}(\pi_j) \cap \mathrm{hep}(\pi_i). \tag{9}$$

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

$$\gamma^{\text{M}}_{i,j}(t) \stackrel{\text{def}}{=} \mathrm{BRT} \cdot \sum_{\ell=1}^{E_j(t)} \left| \mathtt{sort}\left(M_{i,j}(t) \right) [\ell] \right|, \tag{10}$$

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

For the UCB-Only Multiset approach, the multiset $M_{i,j}(t)$ contains $E_j(R_h) \cdot E_h(t)$ copies of the *size* of UCB_h of each task $h \in \mathrm{aff}(\pi_i, \pi_j)$ affecting task τ_i and affected by task τ_j, i.e.

$$M^{\text{ucb-o}}_{i,j}(t) \stackrel{\text{def}}{=} \bigcup_{h \in \mathrm{aff}(\pi_i,\pi_j)} \left(\bigcup_{E_j(R_h) \cdot E_h(t)} \left|\mathrm{UCB}_h\right| \right). \tag{11}$$

[2] Strictly speaking, the condition $\mathrm{aff}(\pi_i, \pi_j) \neq \emptyset$ in (8) can be removed, because $\gamma^{\text{ecb-o}}_{i,j}(t)$ is only applied in a context where $i \in \mathrm{lp}(\pi_j)$. We inserted the condition to ease the comparison of FPPS (this section) and FPTS (later on).

Instead, for the ECB-Union Multiset approach, for each task $h \in \mathrm{aff}(\pi_i, \pi_j)$ the multiset $M_{i,j}(t)$ contains $E_j(R_h) \cdot E_h(t)$ copies of the size of the intersection of UCB_h and the ECBs of all tasks in $\mathrm{hep}(\pi_j)$, i.e.

$$M^{\text{ecb-u}}_{i,j}(t) \stackrel{\text{def}}{=} \bigcup_{h \in \mathrm{aff}(\pi_i,\pi_j)} \left(\bigcup_{E_j(R_h) \cdot E_h(t)} \left| \mathrm{UCB}_h \cap \left(\bigcup_{g \in \mathrm{hep}(\pi_j)} \mathrm{ECB}_g \right) \right| \right). \tag{12}$$

Note that (12) extends (11) by intersecting every UCB_h with $\left(\bigcup_{g \in \mathrm{hep}(\pi_j)} \mathrm{ECB}_g \right)$.

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

$$M^{\text{ucb}}_{i,j}(t) \stackrel{\text{def}}{=} \bigcup_{h \in \mathrm{aff}(\pi_i,\pi_j)} \left(\bigcup_{E_j(R_h) \cdot E_h(t)} \mathrm{UCB}_h \right). \tag{13}$$

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

$$M^{\text{ecb}}_j(t) \stackrel{\text{def}}{=} \bigcup_{E_j(t)} \mathrm{ECB}_j. \tag{14}$$

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

$$\gamma^{\text{ucb-u}}_{i,j}(t) \stackrel{\text{def}}{=} \mathrm{BRT} \cdot \left| M^{\text{ecb}}_j(t) \cap M^{\text{ucb}}_{i,j}(t) \right|. \tag{15}$$

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 affect the execution of a task τ_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* τ_j: the set of tasks that can execute jobs in the interval *and* can be pre-empted by task τ_j;

TABLE II

Overview of tasks that can execute and affect the execution of task τ_i in a level-i active period starting at time $t = 0$ for both FPPS with constrained deadlines and FPTS with arbitrary deadlines, assuming a task τ_b that blocks τ_i for FPTS, i.e. $b \in \mathrm{b}(i)$.

	FPPS	FPTS			
interval	$[0, R_i)$	$[0, H_i)$	$[0, L_i)$	$[0, S_{i,k})$	$[0, F_{i,k})$
execute	$\mathrm{hep}(\pi_i)$	$\{i\} \cup \mathrm{hp}(\theta_i)$	$\{b\} \cup \mathrm{hep}(\pi_i)$	see $[0, L_i)$	see $[0, L_i)$
affected by τ_j	$\mathrm{lp}(\pi_j) \cap \mathrm{hep}(\pi_i) = \mathrm{aff}(\pi_i, \pi_j)$	$\mathrm{lt}(\pi_j) \cap (\{i\} \cup \mathrm{hp}(\theta_i))$	$\mathrm{lt}(\pi_j) \cap (\{b\} \cup \mathrm{hep}(\pi_i))$	see $[0, L_i)$	see $[0, L_i)$
#-jobs	$\begin{cases} E_h(R_i) & \text{if } h \in \mathrm{hep}(\pi_i) \\ 0 & \text{otherwise} \end{cases}$	$\begin{cases} E_h(H_i) & \text{if } h \in \mathrm{hp}(\theta_i) \\ 1 & \text{if } i \\ 0 & \text{otherwise} \end{cases}$	$\begin{cases} E_h(L_i) & \text{if } h \in \mathrm{hep}(\pi_i) \\ 1 & \text{if } b \\ 0 & \text{otherwise} \end{cases}$	$\begin{cases} E_h(S_{i,k}) & \text{if } h \in \mathrm{hp}(\pi_i) \\ k & \text{if } i \\ 1 & \text{if } b \\ 0 & \text{otherwise} \end{cases}$	$\begin{cases} E_h(F_{i,k}) & \text{if } h \in \mathrm{hp}(\theta_i) \\ E_h(S_{i,k}) & \text{if } h \in \mathrm{hp}(\pi_i) \setminus \mathrm{hp}(\theta_i) \\ k+1 & \text{if } i \\ 1 & \text{if } b \\ 0 & \text{otherwise} \end{cases}$

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

The "*#-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 τ_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* τ_j in Table II; see (8). When we focus on the *pre-empted* tasks, e.g. when using the UCB-Only Multiset approach, the *#-jobs* also play a role, i.e. the multiset $M^{\text{ucb-o}}_{i,j}(t)$ in (11) contains $E_j(R_h) \cdot E_h(t)$ copies of the size of UCB_h for each task $h \in \mathrm{aff}(\pi_i, \pi_j)$ affecting τ_i and affected by τ_j.

In the next sections, the information in Table II is the basis for the extensions for FPTS with CRPD.

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

For FPPS with constrained deadlines, all pre-emptions during the response time of a job of a task may actually evict UCBs of that job. For FPTS, 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-emptive 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* $a_{i,k}$, (*absolute*) *start-time* $s_{i,k}$ and (*absolute*) *finishing time* $f_{i,k}$ of a job k of task τ_i; see Figure 1. The length of the interval $[a_{i,k}, f_{i,k})$ and $[s_{i,k}, f_{i,k})$ is termed the *response time* and the *hold time*[3] of job k of task τ_i, respectively.

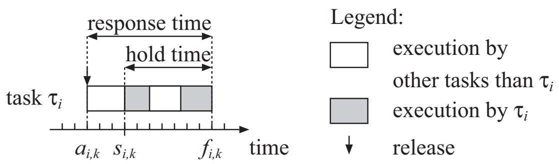

Fig. 1. The response time and hold time of job k of task τ_i.

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

$$H_i = C_i + \sum_{\forall j \in \mathrm{hp}(\theta_i)} E_j(H_i) \cdot C_j. \qquad (16)$$

The worst-case hold time H_i of a task τ_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 τ_b with $b \in \mathrm{b}(i)$, and (*iii*) the interference of tasks τ_j with $j \in \mathrm{hp}(\pi_i) \cap \mathrm{lep}(\theta_i)$ are included in R_i but not in H_i. Example 2 below illustrates (*i*) and Example 3 illustrates (*ii*) and (*iii*).

TABLE III

Task characteristics of $\mathcal{T}_2$ and worst-case response times and hold times of periodic tasks with non-constrained deadlines under FPPS without CRPD.

	T	D	C	$\pi = \theta$	R	H
τ_1	5	5	2	2	2	2
τ_2	7	9	4.2	1	8.6	8.2

TABLE IV

Task characteristics of $\mathcal{T}_3$ and worst-case response times and hold times of periodic tasks under FPTS without CRPD.

	$T = D$	C	π	θ	R	H
τ_1	6	1	4	4	3	1
τ_2	7	2	3	4	5	2
τ_3	9	2	2	3	8	3
τ_4	11	2	1	3	8	3

Example 2. *The characteristics of a set $\mathcal{T}_2$ of periodic tasks is given in Table III. 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 $\mathcal{T}_3$ of periodic tasks are given in Table IV. The worst-case hold times of all tasks are* smaller *than their worst-case response times. Task τ_1 is an example of* (ii)*, task τ_4 is an example of* (iii)*, and tasks τ_2 and τ_3 are examples of both* (ii) *and* (iii)*.*

Tasks τ_3 and τ_4 of Example 3 are particularly interesting when FPTS is extended with CRPD, because task τ_1 can be activated twice during their worst-case response time but only once during their worst-case hold time.

[3]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-emptive by letting them share a so-called *pseudo-resource*. Our *hold time* is the same as the *resource hold time* of the pseudo-resource.

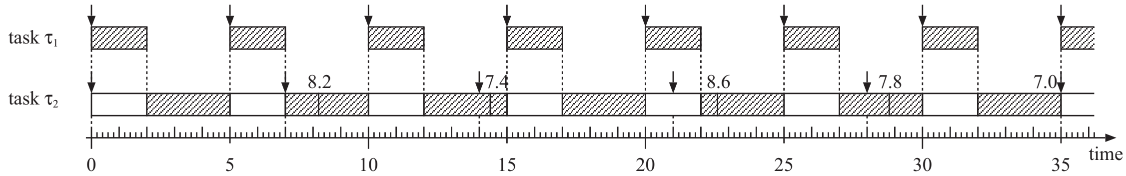

Fig. 2. Timeline for $\mathcal{T}_2$ for an entire hyper period (i.e. $\mathrm{lcm}(T_1, T_2) = 35$) with a simultaneous release of τ_1 and τ_2 at time $t = 0$. The numbers to the top right corner of the boxes denote the response times of the respective job activations.

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 $B_i \neq 0$ 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 τ_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 τ_i itself.

Example 4. *We reconsider $\mathcal{T}_2$ 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 τ_b that blocks τ_i under FPTS, i.e. $b \in \mathrm{b}(i)$, 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 τ_b is taken into account. Similarly, exactly k and $k + 1$ jobs of τ_i are taken into account when determining $S_{i,k}$ and $F_{i,k}$, respectively.

Example 5. *We reconsider $\mathcal{T}_2$ of Example 2. The worst-case finalization time $F_{2,0}$ of the first job of τ_2 is equal to 8.2. Because $E_2(8.2) = 2$, (11) would include 2 jobs of τ_2 in $M^{ucb\text{-}o}_{2,1}(8.2)$ rather than 1. To prevent this pessimism, we explicitly take the number of jobs of τ_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 $j \in hp(\pi_i)$ and assume a level-i active period starting at time $t = 0$ with a simultaneous release of τ_i and τ_j. Let $S_{i,k}$ denote the worst-case start time of job k of τ_i in that level-i active period and be derived by (3). Now the following equality holds:*

$$\forall_{j \in hp(\pi_i)} E^*_j(S_{i,k}) = E_j(S_{i,k}). \tag{17}$$

Proof. *The term $E^*_j(S_{i,k})$ represents the maximum number of activations of τ_j in the interval $[0, S_{i,k}]$. When $\exists_{m \in \mathbb{N}} S_{i,k} = m \cdot T_j$, task τ_j is activated at time $S_{i,k}$. This would imply that τ_i cannot start at $S_{i,k}$, which contradicts the definition of $S_{i,k}$. We therefore conclude that $\nexists_{m \in \mathbb{N}} S_{i,k} = m \cdot T_j$. 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.*

$$F_{i,k} = S_{i,k} + C_i + \sum_{\forall j \in hp(\theta_i)} \Big(E_j(F_{i,k}) - E_j(S_{i,k})\Big) \cdot C_j. \tag{18}$$

Similarly, Lemma 2 shows that $\gamma_{i,j}(t)$ 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 $\gamma_{i,k}(t)$ for the case $B_i = 0$, $\gamma_{i,k}(t)$ can be based on $E_j(t)$ rather than $E^*_j(t)$.*

Proof. *A solution for the recurrent relation for $S_{i,k}$ is found when $S^{(\ell)}_{i,k} = S^{(\ell+1)}_{i,k}$ for two subsequent iterations. For $S^{(\ell)}_{i,k}$ there are two cases, either $E_j(S^{(\ell)}_{i,k}) = E^*_j(S^{(\ell)}_{i,k})$ or $E_j(S^{(\ell)}_{i,k}) \neq E^*_j(S^{(\ell)}_{i,k})$.*

Let $E_j(S^{(\ell)}_{i,k}) = E^*_j(S^{(\ell)}_{i,k})$, i.e. $\nexists_{m \in \mathbb{N}} S^{(\ell)}_{i,k} = m \cdot T_j$. As a result, it doesn't matter whether $E_j(t)$ or $E^*_j(t)$ is used in $\gamma_{i,k}(t)$.

Now let $E_j(S^{(\ell)}_{i,k}) \neq E^*_j(S^{(\ell)}_{i,k})$, i.e. $\exists_{m \in \mathbb{N}} S^{(\ell)}_{i,k} = m \cdot T_j$. As a result, an additional activation of τ_j will be taken into account when determining $S^{(\ell+1)}_{i,k}$, irrespective of using either $E_j(t)$ or $E^*_j(t)$ in $\gamma_{i,k}(t)$. Together, these two cases prove the lemma. □

We therefore conclude that, apart from the number of job activations of τ_b, the information in Table II also holds for τ_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 $\mathcal{T}_3$ of Example 3. Without CRPD, the blocking of τ_2 due to τ_3 and τ_4 is the same because $C_3 = C_4$, i.e. $B_2 = \max(0, \max\{C_3, C_4\}) = 1$. The blocking including CRPD may be different, however, due to different UCBs of τ_3 and τ_4 and the ECBs of τ_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 τ_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 $1 \leq i < n$, is contained in a level-n active period [14]. Hence termination for L_n guarantees termination for L_i for all $1 \leq i < n$. Next, the lowest priority task τ_n cannot be blocked. As a result, when L_n exceeds the least common multiple (LCM) of the periods of the task set $\mathcal{T}$, the iterative procedure will not terminate. This is because at the LCM the activation pattern is repeated and if L_n 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_n 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, $S_{i,k}$ and $F_{i,k}$ for FPTS with a term $\gamma_{i,j}(t)$, we must adapt $\gamma^{\text{ecb-o}}_{i,j}(t)$ by considering the tasks affected by τ_j (see the row *affected by* τ_j in Table II). As shown in Table II, the tasks being affected by pre-emptions are the same for the intervals $[0, L_i)$, $[0, S_{i,k})$, and $[0, F_{i,k})$, 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

$$\text{aff}(\pi_i, \pi_j) \stackrel{\text{def}}{=} \text{lt}(\pi_j) \cap \text{hep}(\pi_i). \tag{19}$$

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 τ_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 $\gamma_{i,j}(t)$, similar to the extension of R_i in (7), i.e.

$$L_i = \sum_{\forall j \in \text{hep}(\pi_i)} \Big(E_j(L_i) \cdot C_j + \gamma_{i,j}(L_i) \Big). \tag{20}$$

For the ECB-Only approach, we can subsequently reuse (8) for $\gamma^{\text{ecb-o}}_{i,j}(t)$ 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.

$$L_i = \max_{\forall b \in \text{b}(i)} \left(C_b + \sum_{\forall j \in \text{hep}(\pi_i)} \Big(E_j(L_i) \cdot C_j + \gamma_{i,j,b}(L_i) \Big) \right). \tag{21}$$

A subscript "b" has been introduced in $\gamma_{i,j,b}(t)$ to capture the CRPD related to the blocking task τ_b. For the ECB-Only approach, $\gamma^{\text{ecb-o}}_{i,j,b}(t)$ is defined as

$$\gamma^{\text{ecb-o}}_{i,j,b}(t) = \begin{cases} \text{BRT} \cdot E_j(t) \cdot \left|\text{ECB}_j\right| & \text{if } \text{aff}(\pi_i, \pi_j) \neq \emptyset \lor b \in \text{lt}(\pi_j) \\ 0 & \text{otherwise} \end{cases}. \tag{22}$$

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

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

1) Tasks without blocking: Similar to L_i, we extend equation (3) for $S_{i,k}$ with a term $\gamma_{i,k}(t)$ to include CRPD, i.e.

$$S_{i,k} = kC_i + \sum_{\forall j \in \text{hp}(\pi_i)} \Big(E^*_j(S_{i,k}) \cdot C_j + \gamma_{i,j}(S_{i,k}) \Big). \tag{23}$$

Based on Lemma 2, we conclude that we can define $\gamma_{i,k}(t)$ in terms of $E_j(t)$ rather than $E^*_j(t)$. Hence, we can also reuse $\gamma^{\text{ecb-o}}_{i,k}(t)$ 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 $S_{i,k}$ with an additional subscript "b" and a term $\gamma_{i,j,b}(t)$, i.e.

$$S_{i,k,b} = C_b + kC_i + \sum_{\forall j \in \text{hp}(\pi_i)} \Big(E_j(S_{i,k,b}) \cdot C_j + \gamma_{i,j,b}(S_{i,k,b}) \Big). \tag{24}$$

For the ECB-Only approach, we can reuse $\gamma^{\text{ecb-o}}_{i,j,b}(t)$ from (22), similar to L_i.

C. Worst-case finalization time $F_{i,k}$

1) Tasks without blocking: We can extend (18) with $\gamma_{i,j}(t)$ terms complementing $E_j(F_{i,k}) \cdot C_j$ and $E_j(S_{i,k}) \cdot C_j$, i.e.

$$\begin{aligned} F_{i,k} = S_{i,k} + C_i &+ \sum_{\forall j \in \text{hp}(\theta_i)} \Big(E_j(F_{i,k}) - E_j(S_{i,k}) \Big) \cdot C_j \\ &+ \sum_{\forall j \in \text{hp}(\theta_i)} \Big(\gamma_{i,j}(F_{i,k}) - \gamma_{i,j}(S_{i,k}) \Big). \end{aligned} \tag{25}$$

Similar to L_i and $S_{i,k}$ we use (8) for $\gamma^{\text{ecb-o}}_{i,k}(t)$, with $\text{aff}(\pi_i, \pi_j)$ as defined in (19).

2) Tasks with blocking: Similar to $S_{i,k}$, we add a subscript "b" to $F_{i,k}$. Similar to the case $B_i = 0$, we expand the formula with terms for CRPD, i.e.

$$\begin{aligned} F_{i,k,b} = S_{i,k,b} + C_i &+ \sum_{\forall j \in \text{hp}(\theta_i)} \Big(E_j(F_{i,k,b}) - E_j(S_{i,k,b}) \Big) \cdot C_j \\ &+ \sum_{\forall j \in \text{hp}(\theta_i)} \Big(\gamma_{i,j,b}(F_{i,k,b}) - \gamma_{i,j,b}(S_{i,k,b}) \Big). \end{aligned} \tag{26}$$

Similar to L_i and $S_{i,k}$, we apply (22) for $\gamma^{\text{ecb-o}}_{i,j,b}(t)$. To compute $F_{i,k}$, we take the maximum value over all tasks that may block τ_i, similar to L_i and as explained in Section IV-D, i.e.

$$F_{i,k} = \max_{\forall b \in \text{b}(i)} F_{i,k,b}. \tag{27}$$

VI. FPTS with CRPD: pre-empted tasks

In this section, we consider the UCB-Only Multiset approach, i.e. we 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 $[0, H_i)$, $[0, L_i)$, $[0, S_{i,k})$ and $[0, F_{i,k})$. As a consequence, this section presents dedicated adaptations of $\gamma_{i,j}(t)$ and $M_{i,j}(t)$, for each interval. For ease 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 $\gamma_{i,j}(t)$, similar to the extension of R_i with $\gamma_{i,j}(t)$, i.e.

$$H_i = C_i + \sum_{j \in \mathrm{hp}(\theta_i)} \Big(E_j(H_i) \cdot C_j + \gamma_{i,j}(H_i) \Big). \tag{28}$$

Although we can apply $\gamma^{\mathrm{M}}_{i,j}(t)$ in (10) for the UCB-Only Multiset approach, we need to adapt the definition of $M^{\text{ucb-o}}_{i,j}(t)$ in (11) to prevent pessimism, as discussed in Sections IV-B and IV-C. Firstly, worst-case hold times are to be considered for pre-empted tasks, rather than worst-case response times. Secondly, exactly one job of task τ_i needs to be considered rather than $E_i(t)$ jobs. These two adaptations of (11) result in

$$M^{\text{ucb-o}}_{i,j}(t) = \biguplus_{h \in \mathrm{aff}'(\theta_i,\pi_j)} \left(\biguplus_{E_j(H_h)\cdot E_h(t)} \left|\mathrm{UCB}_h\right| \right) \uplus \begin{cases} \left(\biguplus_{E_j(H_i)} \left|\mathrm{UCB}_i\right| \right) & \text{if } i \in \mathrm{lt}(\pi_j) \\ \emptyset & \text{otherwise} \end{cases}. \tag{29}$$

Because task τ_i is treated in a separate clause, we need an alternative $\mathrm{aff}'(\pi_i, \pi_j)$ for $\mathrm{aff}(\pi_i, \pi_j)$ excluding $\{i\}$, i.e.

$$\mathrm{aff}'(\pi_i, \pi_j) \stackrel{\text{def}}{=} \mathrm{lt}(\pi_j) \cap \mathrm{hp}(\pi_i) = \mathrm{aff}(\pi_i, \pi_j) \setminus \{i\}. \tag{30}$$

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 $\gamma^{\mathrm{M}}_{i,j}(t)$ with a subscript b for the blocking task τ_b, with $b \in \mathrm{b}(i)$:

$$\gamma^{\mathrm{M}}_{i,j,b}(t) = \mathrm{BRT} \cdot \sum_{\ell=1}^{E_j(t)} \left| \mathtt{sort}\left(M_{i,j,b}(t) \right)[\ell] \right|. \tag{31}$$

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

$$M^{\text{ucb-o}}_{i,j,b}(t) = \biguplus_{h \in \mathrm{aff}(\pi_i,\pi_j)} \left(\biguplus_{E_j(H_h)\cdot E_h(t)} \left|\mathrm{UCB}_h\right| \right) \uplus \begin{cases} \left(\biguplus_{E_j(H_b)} \left|\mathrm{UCB}_b\right| \right) & \text{if } b \in \mathrm{lt}(\pi_j) \\ \emptyset & \text{otherwise} \end{cases}. \tag{32}$$

The pre-condition $b \in \mathrm{b}(i)$ for $M^{\text{ucb-o}}_{i,j,b}(t)$ is taken into account by the max in (21). The definition of $M^{\text{ucb-o}}_{i,j,b}(t)$ is based on $\mathrm{aff}(\pi_i, \pi_j)$, rather than $\mathrm{aff}'(\pi_i, \pi_j)$, because the number of jobs of τ_i are not known a-priori. Moreover, the definition contains the worst-case hold times of τ_h and τ_b 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 τ_b, the definitions of $\gamma^{\mathrm{M}}_{i,j,b}(t)$ and $M^{\text{ucb-o}}_{i,j,b}(t)$ are further extended for $S_{i,k}$ to consider exactly k jobs of τ_i (see Table II), i.e.

$$\gamma^{\mathrm{M}}_{i,j,k,b}(t) = \mathrm{BRT} \cdot \sum_{\ell=1}^{E_j(t)} \left| \mathtt{sort}\left(M_{i,j,k,b}(t) \right)[\ell] \right| \tag{33}$$

and

$$M^{\text{ucb-o}}_{i,j,k,b}(t) = \biguplus_{h \in \mathrm{aff}'(\pi_i,\pi_j)} \left(\biguplus_{E_j(H_h)\cdot E_h(t)} \left|\mathrm{UCB}_h\right| \right) \uplus \begin{cases} \left(\biguplus_{E_j(H_i)\cdot k} \left|\mathrm{UCB}_i\right| \right) & \text{if } i \in \mathrm{lt}(\pi_j) \\ \emptyset & \text{otherwise} \end{cases} \uplus \begin{cases} \left(\biguplus_{E_j(H_b)} \left|\mathrm{UCB}_b\right| \right) & \text{if } b \in \mathrm{lt}(\pi_j) \\ \emptyset & \text{otherwise} \end{cases}. \tag{34}$$

Similar to H_i, task τ_i is again treated by a separate clause, necessitating the usage of $\mathrm{aff}'(\theta_i, \pi_j)$ rather than $\mathrm{aff}(\theta_i, \pi_j)$. Moreover, $M^{\text{ucb-o}}_{i,j,k,b}(t)$ is based on the worst-case hold times of τ_h, τ_i, and τ_b rather than their worst-case response times.

Similar 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 $\gamma_{i,j,k,b}(t)$ as follows:

$$S_{i,k,b} = C_b + kC_i + \sum_{\forall j \in \mathrm{hp}(\pi_i)} \Big(E_j(S_{i,k,b}) \cdot C_j + \gamma_{i,j,k,b}(S_{i,k,b}) \Big). \tag{35}$$

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

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

$$\gamma^{\mathrm{M}}_{i,j,k,b}(t_s, t_f) = \mathrm{BRT} \cdot \sum_{\ell=1}^{E_j(t_f)} \left| \mathtt{sort}\left(M_{i,j,k,b}(t_s, t_f) \right)[\ell] \right|, \tag{36}$$

and

$$M^{\text{ucb-o}}_{i,j,k,b}(t_s, t_f) = \biguplus_{h \in (\mathrm{aff}'(\pi_i,\pi_j) \cap \mathrm{hp}(\theta_i))} \left(\biguplus_{E_j(H_h)\cdot E_h(t_f)} \left|\mathrm{UCB}_h\right| \right) \uplus \biguplus_{h \in (\mathrm{aff}'(\pi_i,\pi_j) \setminus \mathrm{hp}(\theta_i))} \left(\biguplus_{E_j(H_h)\cdot E_h(t_s)} \left|\mathrm{UCB}_h\right| \right) \uplus \begin{cases} \left(\biguplus_{E_j(H_i)\cdot(k+1)} \left|\mathrm{UCB}_i\right| \right) & \text{if } i \in \mathrm{lt}(\pi_j) \\ \emptyset & \text{otherwise} \end{cases} \uplus \begin{cases} \left(\biguplus_{E_j(H_b)} \left|\mathrm{UCB}_b\right| \right) & \text{if } b \in \mathrm{lt}(\pi_j) \\ \emptyset & \text{otherwise} \end{cases}. \tag{37}$$

Similar to the ECB-Only approach, $F_{i,k}$ is extended with a subscript "b" and $\gamma_{i,j,k,b}$ terms, i.e.

$$\begin{aligned} F_{i,k,b} &= S_{i,k,b} + C_i \\ &+ \sum_{\forall j \in \text{hp}(\theta_i)} \Big(E_j(F_{i,k,b}) - E_j(S_{i,k,b})\Big) \cdot C_j \\ &+ \sum_{\forall j \in \text{hp}(\theta_i)} \Big(\gamma_{i,j,k,b}(S_{i,k,b}, F_{i,k,b}) - \gamma_{i,j,k,b}(S_{i,k,b})\Big). \end{aligned} \tag{38}$$

The term $\gamma_{i,j,k,b}(S_{i,k,b})$ in (38) prevents the cache-related pre-emption costs already covered in (35) for $S_{i,k,b}$ to be accounted for twice.

We may subsequently determine $F_{i,k}$ by (27) and can derive R_i through (5) 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 multisets 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 multisets $M^{\text{ecb-u}}_{i,j,k,b}(t_s, t_f)$ and $M^{\text{ucb-u}}_{i,j,k,b}(t_s, t_f)$ 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_i 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-empting tasks for every pre-empted task individually. Similar to FPPS with CRPD, the definition of the multiset of the UCB-Only Multiset approach will be extended by intersecting the UCBs of every affected task with $\left(\bigcup_{g \in \text{hep}(\pi_j)} \text{ECB}_g\right)$, e.g. from (37) for $M^{\text{ucb-o}}_{i,j,k,b}(t_s, t_f)$ we derive

$$\begin{aligned} &M^{\text{ecb-u}}_{i,j,k,b}(t_s, t_f) = \\ &\quad \biguplus_{h \in (\text{aff}'(\pi_i,\pi_j) \cap \text{hp}(\theta_i))} \left(\biguplus_{E_j(H_h) \cdot E_h(t_f)} \left| \text{UCB}_h \cap \left(\bigcup_{g \in \text{hep}(\pi_j)} \text{ECB}_g \right) \right| \right) \\ &\cup \biguplus_{h \in (\text{aff}'(\pi_i,\pi_j) \setminus \text{hp}(\theta_i))} \left(\biguplus_{E_j(H_h) \cdot E_h(t_s)} \left| \text{UCB}_h \cap \left(\bigcup_{g \in \text{hep}(\pi_j)} \text{ECB}_g \right) \right| \right) \\ &\cup \begin{cases} \left(\biguplus_{E_j(H_i) \cdot (k+1)} \left| \text{UCB}_i \cap \left(\bigcup_{g \in \text{hep}(\pi_j)} \text{ECB}_g \right) \right| \right) & \text{if } i \in \text{lt}(\pi_j) \\ \emptyset & \text{otherwise} \end{cases} \\ &\cup \begin{cases} \left(\biguplus_{E_j(H_b)} \left| \text{UCB}_b \cap \left(\bigcup_{g \in \text{hep}(\pi_j)} \text{ECB}_g \right) \right| \right) & \text{if } b \in \text{lt}(\pi_j) \\ \emptyset & \text{otherwise} \end{cases} \end{aligned} \tag{39}$$

The equations for $\gamma^{\text{M}}_{i,j,k,b}(t_s, t_f)$ in (36), $F_{i,k,b}$ 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 $M^{\text{ucb}}_{i,j,k,b}(t_s, t_f)$ is formed. Similar to FPPS with CRPD, the definition for $M^{\text{ucb}}_{i,j,k,b}(t_s, t_f)$ can be derived from (37) for $M^{\text{ucb-o}}_{i,j,k,b}(t_s, t_f)$ by removing all cardinality operators, i.e.

$$\begin{aligned} M^{\text{ucb}}_{i,j,k,b}(t_s, t_f) = &\biguplus_{h \in (\text{aff}'(\pi_i,\pi_j) \cap \text{hp}(\theta_i))} \left(\biguplus_{E_j(H_h) \cdot E_h(t_f)} \text{UCB}_h \right) \\ &\cup \biguplus_{h \in (\text{aff}'(\pi_i,\pi_j) \setminus \text{hp}(\theta_i))} \left(\biguplus_{E_j(H_h) \cdot E_h(t_s)} \text{UCB}_h \right) \\ &\cup \begin{cases} \left(\biguplus_{E_j(H_i) \cdot (k+1)} \text{UCB}_i \right) & \text{if } i \in \text{lt}(\pi_j) \\ \emptyset & \text{otherwise} \end{cases} \\ &\cup \begin{cases} \left(\biguplus_{E_j(H_b)} \text{UCB}_b \right) & \text{if } b \in \text{lt}(\pi_j) \\ \emptyset & \text{otherwise} \end{cases}. \end{aligned} \tag{40}$$

Similar to FPPS with CRPD, the definition of $\gamma^{\text{ucb-u}}_{i,j,k,b}$ is given in terms of the size of the multi-set intersection of $M^{\text{ecb}}_j(t)$ and $M^{\text{ucb}}_{i,j,k,b}(t_s, t_f)$, i.e.

$$\gamma^{\text{ucb-u}}_{i,j,k,b}(t_s, t_f) = \text{BRT} \cdot \left| M^{\text{ecb}}_j(t_f) \cap M^{\text{ucb}}_{i,j,k,b}(t_s, t_f) \right|, \tag{41}$$

where $M^{\text{ecb}}_j(t)$ is defined in (14). The equations for $F_{i,k,b}$ (38) and $F_{i,k}$ (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

$$R_i = \min(R^{\text{ecb-u}}_i, R^{\text{ucb-u}}_i), \tag{42}$$

where $R^{\text{ecb-u}}_i$ and $R^{\text{ucb-u}}_i$ are the worst-case response times of task τ_i 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 $\mathcal{T}$ 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 $\mathcal{T}$ under FPTS, if such an assignment exists. The algorithm traverses the tasks in *ascending* priority order, exploiting the property that the schedulability test for task τ_i is independent of the pre-emption thresholds of tasks with a priority higher than τ_i. For FPTS with CRPD this property does not hold. As an example, a task τ_j may affect a task τ_h, with $j, h \in \text{hp}(\pi_i)$, when the threshold θ_h of τ_h is lower than the priority π_j of τ_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 τ_i is schedulable, it remains schedulable when the pre-emption threshold θ_ℓ of a task τ_ℓ with a priority lower than task τ_i is reduced **and** θ_ℓ either was or becomes lower than priority π_i.

Our OTA algorithm (see Algorithm 1) uses an auxiliary set $\widehat{\Theta} = \{\widehat{\theta_1}, \widehat{\theta_2}, \ldots \widehat{\theta_n}\}$ of *maximum* pre-emption thresholds next to a set $\Theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$ of *assigned* pre-emption thresholds. Upon initialization, all values in $\widehat{\Theta}$ are set to the highest priority π_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 τ_i, it first assigns its maximum pre-emption threshold $\widehat{\theta_i}$ to θ_i (line 7). Next, it tests schedulability of τ_i *without* any blocking and returns *unschedulable* when the test fails (line 9). Otherwise, it tests schedulability of τ_i with blocking by considering each lower priority task τ_ℓ in isolation (lines 11-22). It decreases the maximum pre-emption threshold $\widehat{\theta_\ell}$ of τ_ℓ if-and-only-if τ_i is unschedulable due to blocking by task τ_ℓ (lines 17-19). In that case, $\widehat{\theta_\ell}$ is decreased to the highest priority of all tasks with a priority lower than τ_i, i.e. π_{i+1} of τ_{i+1}. This may increase the CRPD of tasks with a priority lower than τ_i but does not affect the schedulability of tasks with a priority higher than π_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 $\mathcal{T}$ and a priority assignment Π, 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 $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ to denote the set of priorities and $\mathcal{T}^{\mathrm{H}}_m$ to denote the subset of m highest priority tasks with $0 \leq m \leq n$, i.e. $\mathcal{T}^{\mathrm{H}}_0 = \emptyset$, $\mathcal{T}^{\mathrm{H}}_i = \{\tau_h | h \in \mathrm{hep}(\pi_i)\}$ for $1 \leq i \leq n$, and $\mathcal{T}^{\mathrm{H}}_n = \mathcal{T}$.

If the following main invariant holds for $\mathcal{T}$, then Θ contains the maximum pre-emption thresholds for which all tasks in $\mathcal{T}$ are schedulable, where $\Theta = \widehat{\Theta} \subseteq \Pi$.

Invariant 1. *Given a subset $\mathcal{T}^{\mathrm{H}}_m$ of m highest priority tasks*

1) *the set $\widehat{\Theta}$ contains the* maximum *pre-emption threshold of each task such that all tasks in $\mathcal{T}^{\mathrm{H}}_m$ meet their deadlines, i.e. $\forall_{\tau_i \in \mathcal{T}^{\mathrm{H}}_m} R_i \leq D_i$, where $\widehat{\Theta} \subseteq \Pi$.*
2) *the set Θ contains the* assigned *pre-emption threshold of τ_j if $\tau_j \in \mathcal{T}^{\mathrm{H}}_m$, i.e. $\theta_j = \widehat{\theta_j}$, and it contains the priority of τ_j if $\tau_j \notin \mathcal{T}^{\mathrm{H}}_m$, i.e. $\theta_j = \pi_j$.*

The variables in $\widehat{\Theta}$ and Θ are initialized to the highest (non-pre-emptive) priority π_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 $\mathcal{T}^{\mathrm{H}}_0$.

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

Algorithm 1: OptimalThresholdAssignment($\{\tau_1 \ldots \tau_n\}$)

Input: A task set $\mathcal{T} = \{\tau_1 \ldots \tau_n\}$ with $\{C_i, T_i, D_i, \pi_i\}, \forall \tau_i \in \mathcal{T}$.
Output: Task set schedulable and $\theta_i, \forall \tau_i \in \mathcal{T}$, where $\Theta \subseteq \Pi$.
1: **for each** τ_i **do**
2: $\widehat{\theta_i} \leftarrow \pi_1$; {Init. the max. threshold $\widehat{\theta_i}$ with the highest priority π_1.}
3: $\theta_i \leftarrow \pi_i$; {Init. the threshold θ_i with the priority π_i of τ_i.}
4: **end for**{Invariant 1 holds for $\mathcal{T}^{\mathrm{H}}_0$.}
5: **for each** τ_i (from highest to lowest priority π_i) **do**
6: {**Loop invariant:** Invariant 1 holds for $\mathcal{T}^{\mathrm{H}}_{i-1}$.}
7: $\theta_i \leftarrow \widehat{\theta_i}$; {Assign max. threshold $\widehat{\theta_i}$ to θ_i of τ_i.}
8: Compute R_i; {*without* blocking, i.e. $C_b \leftarrow 0$}
9: **if** $R_i > D_i$ **then return** *unschedulable* **end if**
10: {Invariant 2 holds for τ_i and $\mathcal{T}^{\mathrm{H}}_i$.}
11: **for each** τ_ℓ with $\ell \in \mathrm{lp}(\pi_i)$ (from highest to lowest) **do**
12: {**Loop invariant:** Invariant 2 holds for τ_i and $\mathcal{T}^{\mathrm{H}}_{\ell-1}$.}
13: {Test schedulability of τ_i when blocked by τ_ℓ based on $\widehat{\theta_\ell}$:}
14: $\theta_\ell \leftarrow \widehat{\theta_\ell}$; {Temporarily assign max. threshold $\widehat{\theta_\ell}$ to θ_ℓ of τ_ℓ.}
15: Re-compute R_i; {*with* blocking, i.e. $C_b \leftarrow C_\ell$}
16: {Establish Invariant 2 for τ_i and $\mathcal{T}^{\mathrm{H}}_\ell$.}
17: **if** $R_i > D_i$ **then** {Disallow blocking by τ_ℓ:}
18: $\widehat{\theta_\ell} \leftarrow \pi_{i+1}$;
19: **end if**
20: {Reset the threshold θ_ℓ of τ_ℓ (re-establish Invariant 1):}
21: $\theta_\ell \leftarrow \pi_\ell$;
22: **end for** {Invariant 2 holds for τ_i and $\mathcal{T}^{\mathrm{H}}_n$.}
23: **end for** {Invariant 1 holds for $\mathcal{T}^{\mathrm{H}}_n$, i.e. $\Theta = \widehat{\Theta} \subseteq \Pi \wedge \forall_{1 \leq i \leq n} R_i \leq D_i$.}
24: **return** *schedulable*;

invariant holds for $\mathcal{T}^{\mathrm{H}}_{i-1}$. First the pre-emption threshold of τ_i is assigned its maximum value, i.e. θ_i is set to $\widehat{\theta_i}$ (line 7), and the schedulability of τ_i *without* blocking is determined. If τ_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 $\mathcal{T}^{\mathrm{H}}_i$ schedulable. Otherwise 2) has been established for $\mathcal{T}^{\mathrm{H}}_i$ and the inner-loop is entered.

The inner-loop (lines 11-22) considers each task τ_ℓ with a priority lower than τ_i separately. The aim is to establish 1) for $\mathcal{T}^{\mathrm{H}}_i$, based on the following invariant.

Invariant 2. *Given a task τ_i and a subset $\mathcal{T}^{\mathrm{H}}_\ell$ with $\ell \in lep(\pi_i)$, the set $\widehat{\Theta}$ contains the* maximum *pre-emption threshold for each task, where $\widehat{\Theta} \subseteq \Pi$, such that*

1) *all tasks in $\mathcal{T}^{\mathrm{H}}_{i-1}$ are schedulable, and*
2) *τ_i is schedulable when only the set $\mathcal{T}^{\mathrm{H}}_\ell$ is considered, i.e. when all tasks in $\mathcal{T} \setminus \mathcal{T}^{\mathrm{H}}_\ell$ are ignored.*

If this invariant holds for τ_i and $\mathcal{T}$ then $\widehat{\Theta}$ contains the maximum pre-emption thresholds for which all tasks in $\mathcal{T}^{\mathrm{H}}_i$ are schedulable, where $\widehat{\Theta} \subseteq \Pi$, i.e. Invariant 1 holds for $\mathcal{T}^{\mathrm{H}}_i$.

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

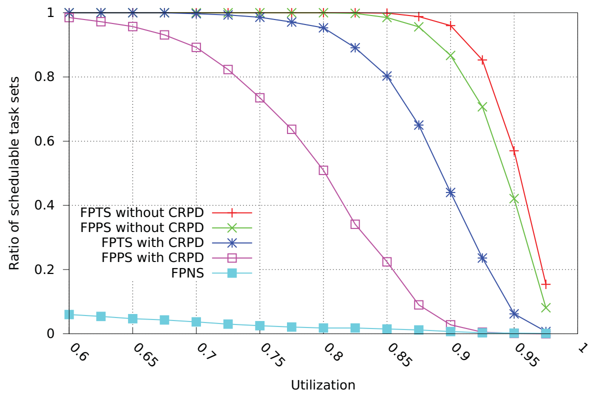

Fig. 3. Ratio of schedulable task sets versus the task sets' utilization.

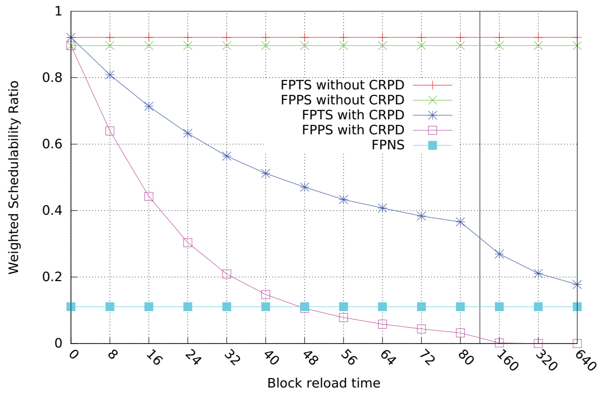

Fig. 4. Weighted schedulability ratio for varying block reload time. The vertical black line indicates a change in the scale of the x-as.

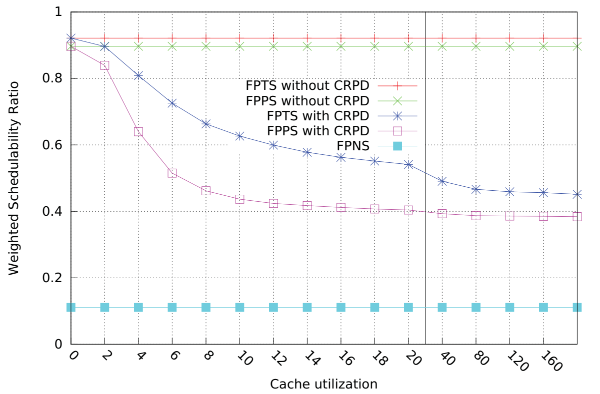

Fig. 5. Weighted schedulability ratio for varying total cache utilization. The vertical black line indicates a change in the scale of the x-as.

been established for $\mathcal{T}_\ell^{\mathrm{H}}$.

At each iteration of the outer-loop, the set $\mathcal{T}_m^{\mathrm{H}}$ of Invariant 1 is increased by one task. Similarly, at each iteration of the inner-loop, the set $\mathcal{T}_\ell^{\mathrm{H}}$ 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 $U^{\mathrm{C}} = 4$, i.e. the total number of ECBs of all tasks is $N \times U^{\mathrm{C}} = 2048$. We then select the cache utilization U_i^{C} of each task (the number of ECBs of a task, $|\mathrm{ECB}_i|$) using UUnifast [11]. 40% of a task's ECBs are also UCBs, i.e. $|\mathrm{UCB}_i| = 0.4 \cdot |\mathrm{ECB}_i|$. 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 $8\mu s$. 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% of all task sets. 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 $S_y(\mathcal{T}, p)$ be the binary result (1 if schedulable, 0 otherwise) of schedulability test y for a task set $\mathcal{T}$ and parameter value p. Then:

$$W_y(p) = \frac{\sum_{\forall \mathcal{T}} U \cdot S_y(\mathcal{T}, p)}{\sum_{\forall \mathcal{T}} U}, \tag{43}$$

where U is the utilization of task set $\mathcal{T}$. This weighted schedulability ratio reduces what would otherwise be a 3-

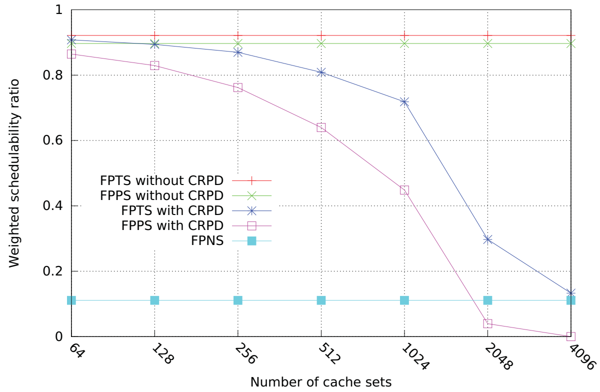

Fig. 6. Weighted schedulability ratio for varying number of cache sets.

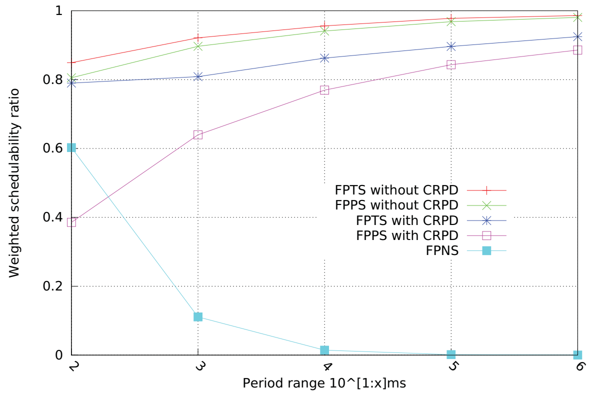

Fig. 7. Weighted schedulability ratio for varying period range.

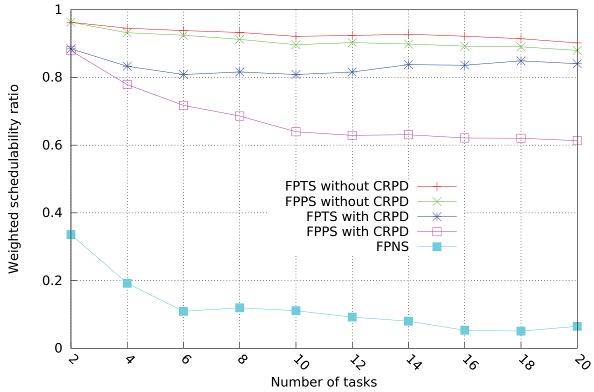

Fig. 8. Weighted schedulability ratio for varying number of tasks.

dimensional plot to 2 dimensions [7]. Weighting the individual schedulability results by task-set utilization reflects the higher placed value being able to schedule higher utilization task sets.

In the second experiment, we vary the block reload time (BRT) from $0\mu s$ to $640\mu s$. 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 FPNS (as expected). It is interesting to see that FPTS with CRPD is able to deem more task sets schedulable than FPNS, 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 FPNS, because not each 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 $8\mu s$. 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 FPNS. This is due to the limited number of cache sets, which limits the overall pre-emption costs. Hence, increasing U^C will have a negligible impact (see the scale change of the total cache utilizations from 40 and onwards). However, only by increasing simultaneously the block reload time and the total cache utilization we have been able to pull down the performance of FPTS with CRPD to the performance of FPNS.

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 increases 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 FPNS, i.e. with FPNS task sets are unaffected by the increased pre-emption costs. We recall that FPTS with CRPD still outperforms FPNS, 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 FPNS 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. This has two reasons: (*i*) as the cache utilization remains constant, the ECBs 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 (FPTS) and arbitrary deadlines. Further, we introduced an Optimal 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 FPPS 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 FPTS and FPPS with and without CRPD. Interestingly, we found that the theoretical performance advantage that FPTS has over FPPS when there are 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, FPTS still retains a performance advantage over FPNS (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 FPTS can rightly be viewed as a potential successor to FPPS as a defacto standard in industry, where it is already supported by both OSEK [1] and AUTOSAR [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

[1] OSEK/VDX operating system. Technical report, Feb. 2005. [Online], Available: http://portal.osek-vdx.org/files/pdf/specs/os223.pdf.

[2] AUTOSAR – specification of operating system, Release 4.1. Technical report, 2010. [Online], Available: http://www.autosar.org/.

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

[4] S. Altmeyer and C. Maiza. Cache-related preemption delay via useful cache blocks: Survey and redefinition. *Journal of Systems Architecture*, 57(7):707–719, Aug. 2011.

[5] N.C. Audsley, A. Burns, M.F. Richardson, and A.J. Wellings. Hard real-time scheduling: The deadline monotonic approach. In *Proc.* 8^{th} *IEEE Workshop RTOSS*, pp. 133–137, May 1991.

[6] A Baldovin, E. Mezzetti, and T. Vardanega. Limited preemptive scheduling of non-independent task sets. In *Proc. EMSOFT*, Sep. 2013.

[7] A. Bastoni, B. Brandenburg, and J. Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In *OSPERT*, pp. 33–44, July 2010.

[8] M. Bertogna, G. Buttazzo, and G. Yao. Improving feasibility of fixed priority tasks using non-preemptive regions. In *Proc.* 32^{nd} *RTSS*, pp. 251–260, Dec. 2011.

[9] M. Bertogna, N. Fisher, and S. Baruah. Static-priority scheduling and resource hold times. In *Proc.* 15^{th} *Workshop PDRTS*, pp. 1–8, Mar. 2007.

[10] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, and G. Buttazzo. Optimal selection of preemption points to minimize preemption overhead. In *Proc.* 23^{rd} *ECRTS*, pp. 217–227, July 2011.

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

[12] R.J. Bril. *Real-time scheduling for media processing using conditionally guaranteed budgets*. PhD thesis, TU/e, The Netherlands, July 2004. http://alexandria.tue.nl/extra2/200412419.pdf.

[13] R.J. Bril, G. Fohler, and W.F.J. Verhaegh. Execution times and execution jitter of real-time tasks under fixed-priority pre-emptive scheduling. Technical Report CSR 08-27, TU/e, The Netherlands, Oct. 2008.

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

[15] R.J. Bril, M.M.H.P. van den Heuvel, U. Keskin, and J.J. Lukkien. Generalized fixed-priority scheduling with preemption thresholds. In *Proc.* 24^{th} *ECRTS*, July 2012.

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

[17] J.V. Busquets-Mataix, J.J. Serrano, R. Ors, P. Gil, and A. Wellings. Adding instruction cache effects to schedulability analysis of preemptive real-time systems. In *Proc.* 2^{nd} *RTAS*, pp. 204–212, June 1996.

[18] G.C. Buttazzo, M. Bertogna, and G. Yao. Limited preemptive scheduling for real-time systems: A survey. *IEEE Transaction on Industrial Informatics (TII)*, 9(1): 3–15, Feb. 2013.

[19] Darren Buttle. Real-time in the prime-time – keynote. In *Proc.* 24^{th} *ECRTS*, page xiii, July 2012.

[20] R.I. Davis and M. Bertogna. Optimal fixed priority scheduling with deferred pre-emption. In *Proc.* 33^{rd} *RTSS*, pp. 39–50, Dec. 2012.

[21] R.I. Davis, N. Merriam, and N. Tracey. How embedded applications using an RTOS can stay within on-chip memory limits. *Proc. WiP and Industrial Experience Sessions ECRTS*, pp. 71–77, 2000.

[22] C. Ferdinand and R. Heckmann. aiT: worst case execution time prediction by static program analysis. In *IFIP*, pp. 377–384, Aug. 2004.

[23] P. Gai, G. Lipari, and M. Di Natale. Minimizing memory utilizations of real-time task sets in single and multi-processor systems-on-a-chip. In *Proc.* 22^{nd} *RTSS*, pp. 73–83, Dec. 2001.

[24] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[25] U. Keskin, R.J. Bril, and J.J. Lukkien. Exact response-time analysis for fixed-priority preemption-threshold scheduling. In *Proc.* 15^{th} *IEEE ETFA, WiP Session*, Sep. 2010.

[26] C.-G. Lee, J. Hahn, Y.-M. Seo, S.L. Min, R. Ha, S. Hong, C.Y. Park, M. Lee, and C.S. Kim. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. *IEEE Transactions on Computers*, 47(6):700–713, June 1998.

[27] W. Lunniss, S. Altmeyer, and R.I. Davis. Optimising task layout to increase schedulability via reduced cache related pre-emption delays. In *Proc.* 20^{th} *RTNS*, pp. 161–170, Oct. 2012.

[28] H. Ramaprasad and F. Mueller. Tightening the bounds on feasible preemption points. In *Proc.* 27^{th} *RTSS*, pp. 212–224, December 2006.

[29] H. Ramaprasad and F. Mueller. Bounding worst-case response time for tasks with non-preemptive regions. In *Proc.* 14^{th} *RTAS*, pp. 58–67, 2008.

[30] J. Regehr. Scheduling tasks with mixed preemption relations for robustness to timing faults. In *Proc.* 23^{rd} *RTSS*, pp. 315–326, 2002.

[31] M. Saksena and Y. Wang. Scalable real-time system design using preemption thresholds. In *Proc.* 21^{st} *RTSS*, pp. 25–34, Dec. 2000.

[32] J. Staschulat, S. Schliecker, and R. Ernst. Scheduling analysis of real-time systems with precise modeling of cache related preemption delay. In *Proc.* 17^{th} *ECRTS*, pp. 41–48, July 2005.

[33] Y. Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In *Proc.* 6^{th} *RTCSA*, pp. 328–335, Dec. 1999.

[34] G. Yao, G. Buttazzo, and M. Bertogna. Bounding the maximum length of non-preemptive regions under fixed-priority scheduling. In *Proc.* 15^{th} *RTCSA*, pp. 351–360, Aug. 2009.