Feasibility interval and sustainable scheduling simulation with CRPD on uniprocessor platform

Hai Nam Tran, Stéphane Rubini, Jalil Boukhobza, Frank Singhoff

Feasibility Interval and Sustainable Scheduling Simulation with CRPD on Uniprocessor Platform

Hai Nam Tran, Stéphane Rubini, Jalil Boukhobza, Frank Singhoff

Lab-STICC, CNRS UMR 6285, Univ. Brest, France

Abstract

The use of hardware caches became essential in modern embedded systems to address the speed gap between processor and memory. In such systems, cache-related preemption delay (CRPD) may represent a significant proportion of task execution time. Addressing this delay in scheduling simulation of these systems stays an open and under-examined problem. Assumptions are often made to simplify the computation model used in simulation and capture the worst-case effect. Nevertheless, they can introduce situations in which scheduling simulation is considered not only pessimistic but also non-sustainable. In this article, we discuss the problem and propose a less pessimistic CRPD computation model that allows sustainable scheduling simulation regarding the capacity parameter. With the proposed model, a system that is schedulable with simulated worst-case execution times remains so when these parameters are reduced. These results improve the applicability of scheduling simulation in the early verification stage for systems with caches. Experiments conducted with our CRPD computation model show a 5% to 12% improvement of schedulability task set coverage and a 30% to 50% reduction of preemption cost with regard to existing CRPD computation models. An integration in a scheduling simulator and a performance evaluation are also realized for the proposed model.

Keywords: real-time embedded systems, scheduling simulation, cache related preemption delay

1. Introduction

The popularization of cache memory in the embedded market comes from two motives. First, over the past two decades, processor speeds have increased dramatically. The use of high performance embedded processors means that memory access time has become a significant bottleneck which requires the use of cache memory. Second, more and more commercial off-the-shelf components are used in real-time embedded system (RTES) design as they allow significant cost savings [1]. Indeed, most commercial processors incorporate cache memories.

Cache memory may enhance the whole performance in term of response time; however, it can also lead to execution time variability due to the varia-

tion of preemption cost in preemptive scheduling context. Multiple tasks could share this hardware resource which may lead to cache-related preemption delay (CRPD) being introduced. By definition, CRPD is the delay added to the execution time of a preempted task because it has to reload cache blocks evicted by a preemption [2]. The work in [3] has shown that CRPD can represent up to 40% of the Worst-Case Execution Time (WCET) of a program.

One of the most distinctive traits of RTES is that they have to respect critical timing constraints. In these systems, the schedulability of each task must be determined by applying scheduling analysis methods. As we have seen that the effect of CRPD is not negligible, it is critical to account for this delay in scheduling analysis. In this article, we discuss an approach to address CRPD in simulation-based scheduling analysis.

Scheduling simulation is a popular analysis method used in early verification to provide a mean to detect timing constraint violations and evaluate the schedulability of a RTES. The advantage of simulation is that it allows system designers to perform fast prototyping with certain levels of accuracy. In a study conducted in [4], it was observed that the cost of solving a fault detected at the early verification step can be 16 times less than solving it later in the development life-cycle. In order to perform scheduling simulation, one needs to define an abstract model of the targeted system including its software and hardware components as well as their interactions. Then, the schedule over a given interval of time is computed and properties such as timing constraint violations are evaluated [5].

Three properties are important to assess the applicability and the validity of scheduling simulation: *sustainability*, *feasibility interval*, and *pessimism*.

- First, a given scheduling policy and/or a schedulability test is sustainable if a system that is schedulable under its worst-case specification remains so when its behavior is better than the worst-case [6]. This property allows us to guarantee the schedulability of a system by analyzing or testing only the worst-case scenario instead of all possible cases, which is nearly impossible to achieve in practice.

- Second, scheduling simulation is a valid schedulability test for a given system under the assumption that the feasibility interval is known. By definition, the feasibility interval is the minimum interval needed to verify the schedulability of a system [7]. If no deadline is missed during this interval, we can guarantee that no deadline will ever be missed. This property allows us to guarantee the schedulability of a system by simulating it during a limited interval of time, under the assumption that it is sustainable.

- Finally, pessimistic assumptions are often taken in scheduling analysis and it is not an exception in scheduling simulation. However, if they are too pessimistic, it leads to a conclusion that a system requires significantly more computing resources to be schedulable than it is in practice.

Problem statement and contribution: In the context of an RTES with caches, as far as we know, the applicability and the validity of scheduling simulation is still an open and under-examined problem. The assumptions, which are made to simplify the computation of CRPD and capture the worst-case effect, can introduce a situation in which scheduling simulation is *non-sustainable* [8], with an *unknown feasibility interval* and *pessimistic*. These three issues significantly limit the use of scheduling simulation as a verification method for systems with caches.

We need a formalization of scheduling simulation with CRPD to systematically address these issues. To the best of our knowledge, scheduling simulation is only used in previous studies [9, 10, 11] to illustrate the effect of CRPD. The authors also used simulation in their experiments; however, the method to perform the simulation has not been formalized. Another problem to consider is the lack of support for cache memory and CRPD in existing scheduling simulators developed in academia [12, 13, 14]. Indeed, the development of open-source scheduling simulators is not keeping pace with analytical-based approaches. The most recent efforts to account for cache memory are SimSo [15] and Cheddar [16] which do not address the problems regarding sustainability, feasibility interval and pessimism.

In this article, we first introduce a *formalization of scheduling simulation* that allows us to systematically describe and address these issues related to CRPD. Then, we propose a new CRPD computation model called $\mathcal{C}^{\text{on-lim}}$. In this model, based on an observation regarding task execution and preemption cost at different program points in [17], we define a constraint that bounds the CRPD by the executed capacity of a task. We show that when this assumption holds, scheduling simulation is *sustainable* and *less pessimistic*. In addition, this assumption also helps us to exhibit *a proof of the feasibility interval*. Finally, $\mathcal{C}^{\text{on-lim}}$ leads to the implementation of a CRPD-aware scheduling simulation methodology which is freely available as a part of Cheddar scheduling simulator [18].

Organization: the rest of the article is organized as follows. Section 2 discusses related work and positions our contribution. Section 3 formalizes the concept of scheduling simulation and presents the system models as well as assumptions taken in this work. Section 4 presents our application of the formalization to existing CRPD computation models. Our proposition of an improved computation model is discussed in section 5. Section 6 describes the sustainability problem in scheduling simulation and proves that our proposed model can guarantee sustainable scheduling simulation regarding the capacity parameter. Then, section 7 provides a feasibility interval proof. Section 8 evaluates the pessimism of the proposed CRPD computation model with synthetic task sets. Section 9 presents the implementation of a CRPD-aware scheduling simulator in Cheddar [18], an open source scheduling analyzer, which is freely available for researchers and practitioners. Finally, section 10 concludes the article and discusses future work.

2. Related Work

In this section, we discuss prior work on scheduling analysis of RTES with caches as well as position our contribution. Several solutions were introduced to consider parameters which capture cache effects in scheduling analysis. In [19], the authors classified these solutions in two categories: cache-aware scheduling and CRPD-aware scheduling.

In cache-aware scheduling, cache accesses are considered precisely and the cache state at each instant has to be known during the system execution, thus, the exact sequence of memory blocks accessed during the task execution must be known [19]. Because of this requirement, cache-aware scheduling analysis is impossible to achieve in practice, except for very simple systems.

In CRPD-aware scheduling, the cache effect is assessed through induced preemption delays [19]. Upper-bounds on the additional delays due to the cache every time a task resumes after a preemption, are precomputed by applying static analysis. Contrary to the cache-aware scheduling problem, the task memory accesses and the cache state at each instant are not required. Thus, CRPD-aware approaches are more applicable in simulation. Our work is positioned in the context of CRPD-aware scheduling.

Static analysis is mostly preferred to compute upper-bounds on the CRPD because it is difficult to determine the worst cases by measurement as stated in [20]. The bounds on the CRPD can be computed using Evicting Cache Blocks (ECBs) [2] or Useful Cache Blocks (UCBs) [21]. By definition, ECBs are blocks used by a preempting task that might override some cache locations used by the preempted task [2]. UCBs are blocks used by a task that are reused later on and will have to be reloaded if evicted from the cache due to preemption [21]. These bounds are incorporated in schedulability analysis to ensure predictability by analytical means or scheduling simulation. Thus, CRPD-aware approaches can be further classified as either analytical-based or simulation-based ones. In order to provide a complete insight of CRPD-aware approaches and better position our contributions, related work in both domains are presented below.

2.1. Analytical-based approaches

Researchers focus on fixed-task priority scheduling and incorporate CRPD in the response time analysis by considering either the preempting task [2], the preempted one [21] or both of them [22, 9, 10]. In [2] and [21], CRPD is only computed by considering either ECBs or UCBs. Later in [22, 9, 10], tighter upper-bounds on CRPD are computed by exploiting both ECBs and UCBs so that a higher schedulability task set coverage is achieved. Later, in [23], the authors proposed an extension of CRPD analysis for Earliest Deadline First (EDF) scheduling based on the processor demand bound function feasibility test.

In another line of work, the authors in [24] present a method to argue about the WCET of each task with the exact cost due to CRPD and provide a safe estimation of the maximal number of preemptions the task may experience within a hyper-period, which is equal to the least common multiplier of all task

periods [25]. However, the authors considered a fixed cost for preemption cost thus they did not account for a precise CRPD computation.

The methods presented above only consider the CRPD to assure predictability, and no change was made to the scheduling policy itself. Certain techniques can be used to either eliminate or limit the effect of CRPD. In [26], the authors studied cache partitioning to increase the predictability by eliminating CRPD. They also showed that the decrease in CRPD does not compensate for the increase in the worst-case execution time of tasks when the cache is partitioned. In [17], the author aimed to reduce CRPD per preemption by applying the effective preemption points technique. Each job of a task is modeled by a sequence of non-preemptive basic blocks, and preemption is allowed only at basic block boundaries. A memory layout optimization technique is also proposed in [27, 28] to achieve the same objective. In another line of work, the authors aim to effectively reduce the number of preemptions by preemption threshold scheduling [29]. In [30], the authors presented the deferred preemption model. In this model, each job of a task is modeled by a sequence of non-preemptive regions separated by a fixed preemption point. It allows a task to run for a period of time without being preempted up to a specific limit.

The problem of computing an optimal scheduling with preemption delays is NP-hard as proved in [8] and neither fixed-task nor fixed-job priority-based scheduling algorithms can be optimal for this problem. The authors also described an offline scheduling approach to synthesize an optimal scheduling table taking into account CRPD by solving a Mixed Integer Linear Programming (MILP) problem [20].

2.2. Scheduling simulation based approaches

A significant amount of effort has been put in the development of scheduling simulators [12, 13, 14, 31, 18] since the seminal work of [32]. The support for cache as well as CRPD is not available in existing scheduling simulators such as MAST [12], STORM [13] and YARTISS [14] that are developed in academia. In [31], the authors present SimSo — a scheduling simulation tool that supports cache sharing on multi-processor systems. It takes into account the impact of caches through statistical models and also the direct overheads such as context switches and scheduling decisions. The memory behavior of a program is modeled based on the Stack Distance Profile (SDP) — the distribution of the stack distances for all the memory accesses of a task, where a stack distance is by definition the number of unique cache lines accessed between two successive accesses to a same line [33]. However, it is difficult to see how the usage of SDP can guarantee the worst-case behavior of preemption during simulation. To the best of our knowledge, it is unclear how to investigate the sustainability and compute the feasibility interval with such model.

Indeed, very few studies have focused on the effect of preemption delay by simulation. In some cases, the authors used simulation in their experiments; however, the method to perform the simulation has not been formalized. Our previous work in [16] presents a CRPD-aware scheduling simulator that is compliant with existing analytical methods in [21], [2], [34] and [28]; however, the

problems regarding sustainability, feasibility interval and pessimism are not addressed. In this article, we aim to address these issues by presenting a formal and detailed description of CRPD-aware scheduling simulation.

The main advantage of the analytical approaches above is that they require less computational complexity and are sustainable with regard to task capacities, periods and deadlines [19]. The limitation is that, comparing to scheduling simulation, analytical approaches are more pessimistic and have a lower schedulability task set coverage. Scheduling simulation, despite of its popular use in the industry, has been proved to be a non sustainable schedulability test with existing CRPD computation models [19]. Thus, in the studies above, scheduling simulation was used as a necessary condition or a baseline to evaluate and compare different approaches in terms of schedulability task set coverage. In this article, we show that scheduling simulation with our CRPD computation model is less pessimistic comparing to existing simulations and results in a higher number of schedulable task sets. It complements existing work on the scheduling simulation aspect by providing a better baseline that yields a higher percentage of schedulability task set coverage. In addition, scheduling simulation can also be used to record and analyze properties such as the numbers of preemptions and various scheduling events that are not observable in static analysis. Furthermore, we show that under certain restricted scenarios, scheduling simulation also provides a sufficient condition for schedulability.

3. Formalization of CRPD-aware scheduling simulation

In this section, we first formalize the concept of scheduling simulation. Then, we identify the characteristics that make a simulation CRPD-aware. The system model, notations used and assumptions taken in this work are presented along.

Works in this domain [12, 13, 14, 31, 18], including ours, adopt assumptions with regard to system models, scheduler and simulation interval that are well-established in the real-time scheduling theory. The main goal of scheduling simulation is to verify the schedulability and detect the violations of timing constraints. As an exhaustive exploration with all possibilities of input parameters and timing properties is not possible, simplified models of *tasks* and underlying hardware *architectures* are used. In addition, timing properties are only verified in the worst case scenarios. Furthermore, we need to define a *scheduler* that provides an algorithm or a policy for ordering the execution of tasks on processors according to some predefined criteria [35]. Lastly, the simulation is done in a limited interval of time which is generally proved to be sufficient to guarantee the schedulability.

Definition 1 (Scheduling simulation). *Scheduling simulation is a simulation of a task set $\mathcal{T}$ on an architecture model $\mathcal{M}$ under a scheduler $\mathcal{S}$ over a limited interval of time $\mathcal{F}$*

Assumptions of this work regarding $\mathcal{T}, \mathcal{M}, \mathcal{S}$ and $\mathcal{F}$ are presented below. Classical notations of real-time systems are used to express our models:

- A task set consists of n *independent periodic* tasks, $\mathcal{T} = \{\tau_1, ..., \tau_n\}$. A task is defined by a quintuple: (C_i, T_i, D_i, Π_i, O_i). The parameter C_i, called the capacity, denotes the worst-case execution time (WCET) of task τ_i when it executes non-preemptively starting from an empty cache. T_i, called the period, denotes the fixed interval between two successive releases of τ_i. D_i is the deadline of τ_i. In this article, we assume that tasks have constrained deadlines (i.e. $\forall i : D_i \leq T_i$).

  A release of a task τ_i at time t is called a job and denoted as $\tau_i[t]$. The first release of a task is at time $t = O_i$. We assume relative deadlines [36] which means that all deadlines are relative to the release time of jobs. For instance, the deadline of $\tau_i[t]$ is at time $t + D_i$. If $\forall \tau_i : O_i = 0$, the task set is called synchronous; otherwise, the task set is called asynchronous. All timing parameters are assumed to be non-negative integers, i.e. they are multiples of some elementary time intervals (for example the "CPU tick", the smallest indivisible CPU time unit).

  A task is assigned a priority level Π_i. We assume that the larger the value, the higher the priority level. In addition, we assume that when a task completes execution, its instructions in the cache are completely evicted. The problem of persistent cache block (PCB), which was described in [37], is not addressed in this work. We are aware that taking into account PCBs can result in less pessimistic simulations by allowing PCB-based WCET reductions. In this case, we can assume that subsequent jobs of a task can reuse the UCBs from previously released jobs. The main challenge is to prove that this WCET reduction is safe and finding a lower-bound. This problem has not been addressed in the existing work, and is not in the scope of this article.

- We assume a fixed-priority preemptive (FPP) work-conserving scheduler S. Work-conserving means that whenever a task completes earlier than its WCET, the scheduler does not idle up to the theoretical WCET of the task but executes other tasks.

- We assume a uniprocessor architecture model $\mathcal{M}$ with one level direct-mapped instruction cache shared amongst tasks. The processor has a single processing unit that is used to execute tasks. We assume this cache architecture because it is easier to capture the worst-case effect of CRPD than complex ones which include data cache or multi-level of caches. In the sequel, the term data is used to refer to a task's data in the instruction cache.

- The interval $\mathcal{F}$ is chosen such that it is the minimum known interval needed to verify the schedulability of task set $\mathcal{T}$ with regard to $(\mathcal{M}, \mathcal{S}, \mathcal{C})$, which is also called the *feasibility interval*. By definition, it is sure that no deadline will ever be missed if and only if, when we only keep the task requests made in $\mathcal{F}$, all their respected deadlines are met [7].

CRPD is taken into account in scheduling simulation through the assessment of preemption delay. Without considering the effect of preemption delay, the duration that a job occupies the processor is equal to or less than the task capacity. However, when CRPD is taken into account, a job may occupy the processor longer than the task capacity. Thus, there is a difference between the capacity of a task and the execution time of its jobs. We define the two terms as follows.

Definition 2 (Capacity). *The capacity C_i of a task τ_i is its statically analyzed worst-case execution time. It does not include the delay added due to CRPD at runtime.*

Definition 3 (Execution time). *The execution time E_i^t of a job $\tau_i[t]$ consists of the duration of time during which the job occupies the processor and the delay added due to CRPD at runtime.*

To facilite the later discussion on the subject of sustainability, we provide the definition of two terms "reduced capacity" and "executed capacity". The first term was used in the seminal work of [6]. Even though one can argue that the capacity is statically analyzed thus cannot be reduced, this term was popularly used and accepted in sustainability analysis. The second term is useful in demonstrating the effect of CRPD.

Definition 4 (Reduced capacity). *A task τ_i has a reduced capacity if it only requires $C_i' < C_i$ units of time to complete its execution. It does not include the delay added due to CRPD at runtime.*

Definition 5 (Executed capacity). *The executed capacity of a job of task τ_i refers to the portion of its capacity that has been executed, without any CRPD taken into account.*

For each job, the execution time must be computed by both the corresponding task's capacity and the CRPD, which is taken into account in scheduling simulation by considering two types of events:

- Task execution events, which are denoted as $Exec(\tau_i, t)$, are raised when task τ_i executes on the processor at time t.
- Preemption events, which are denoted as $Prem(\tau_i, \tau_j, t)$, are raised when task τ_i is preempted directly or indirectly by higher priority task τ_j at time t. In our work, the notion of preemption is applied to both direct and nested preemptions, which are defined as follows.

  Definition 6 (Direct Preemption [21]). *τ_i is said to be directly preempted by a higher priority task τ_j released at time t if τ_i is executing at time $t-1$ and is not completed at time t.*

Definition 7 (Nested Preemption [10]). *τ_i is said to be indirectly preempted by a higher priority task τ_j released at time t if τ_i is not completed at time t and is not executing at time $t-1$*

The reason τ_i is not executing at time t is because it was preempted by another higher priority task before time t. In this case, τ_j indirectly preempts τ_i by preempting the task that preempted τ_i; hence we call this a nested preemption. When a task preempts several lower priority tasks, we need to handle several preemption events.

The method of handling the two types of events is defined by a CRPD computation model. It consists of assumptions regarding cache access profile and events that allow us to study the effect of CRPD during scheduling simulation.

Definition 8 (CRPD computation model). *A CRPD computation model $\mathcal{C}$ describes a method of computing the CRPD added to the execution time of a task when it resumes after a preemption. A model is defined by $\mathcal{C} = \{\Gamma, \mathcal{H}\}$. Γ is the cache access profile which describes the cache utilization of tasks. $\mathcal{H}$ is an algorithm to handle preemption events and task execution events.*

Definition 9 (CRPD-aware scheduling simulation). *A CRPD-aware scheduling simulation is a scheduling simulation with a CRPD computation model $\mathcal{C}$*

In the next section, we apply the presented formalization to the two CRPD computation models that are used in the literature.

4. Formalization of CRPD computation models

In this section, we present two types of CRPD computation models: an offline computed CRPD called $\mathcal{C}^{\text{off}}$ in which CRPD is computed before simulation time, and an online computed CRPD called $\mathcal{C}^{\text{on}}$ in which CRPD is computed during simulation time.

4.1. $\mathcal{C}^{off}$: an offline CRPD computation model

The CRPD that a task experiences due to a preemption is computed offline before simulation time and does not change regardless of cache state during the simulation. This offline CRPD computation model, denoted $\mathcal{C}^{\text{off}} = \{\Gamma, \mathcal{H}\}$, can be defined as follows:

- $\Gamma = \{(\gamma_i) \mid \forall \tau_i \in \mathcal{T}\}$. γ_i is the CRPD added to the execution time of task τ_i whenever τ_i is preempted regardless of the preempting task. We can consider that γ_i is provided as a prerequisite or computed by applying the UCB analysis method presented in [21].
- $\mathcal{H}$: whenever a task τ_i is preempted by a higher priority task, γ_i is added to the execution time of task τ_i.

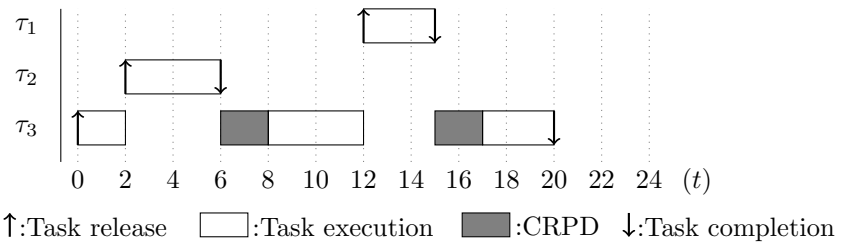

Figure 1: CRPD is computed offline and does not depend on the preempting task. τ_3 experiences 2 units of CRPD every time it is preempted.

This model is pessimistic because preempting tasks may not evict the data in the cache of the preempted task. A schedule provided by applying $\mathcal{C}^{\text{off}}$ is depicted in Figure 1. To keep the example short and simple, we only give an incomplete schedule without detailed task parameters. In this example, $\mathcal{C}^{\text{off}}$ assumes that task τ_3 experiences 2 units of CRPD every time it is preempted even if for some preempting tasks, the CRPD is smaller. In addition, as pointed out in [17], the maximum amount of CRPD is related to the amount of useful information that have to be reloaded into the cache. Indeed, if a task is preempted shortly after it starts, it has not yet loaded all of the UCBs and will therefore not experience the maximum CRPD. In other words, if a task is preempted during the period that it is reloading the evicted cache blocks, it is pessimistic to add the offline computation CRPD to the remaining execution time.

Furthermore, the pessimism also depends on the method of computing the CRPD offline for each task. For example, we can assume that either all or part of the data of τ_i in the cache are evicted and need to be reloaded.

4.2. $\mathcal{C}^{on}$: an online CRPD computation model

CRPD is computed based on the cache access profiles presenting the sets of UCBs and ECBs of each task. Then, during scheduling simulation, we evaluate the number of UCBs that have to be reloaded when a task resumes after preemptions. $\mathcal{C}^{\text{on}} = \{\Gamma, \mathcal{H}\}$ can be defined as follows.

- $\Gamma = \{(\text{UCB}_i, \text{ECB}_i) \,|\, \forall \tau_i \in \mathcal{T}\}$: for each task τ_i, a set of UCBs and ECBs are statically computed before simulation. They are denoted UCB_i and ECB_i, respectively.
  - UCB_i is computed by applying data flow analysis on a task's control flow graph presented in [21]. This can be achieved by using a static analysis tool such as aiT [1]. For a task, the sets of UCBs of each basic block are computed. UCB_i is then determined by the basic block with the highest number of UCBs. The usage of only a single set of

[1] http://www.absint.com/ait

UCBs is indeed a simplification. However, it has been shown in [10] that the correctness of CRPD computation is not impacted. In [38], the authors exploit the fact that for the i-th preemption, only the i-th highest number of UCB has to be considered. However, as shown in [10] and [17], a significant reduction typically only occurs at a high number of preemptions.

- ECB_i is computed by taking into account cache blocks accessed in the execution of a task. For a task, we assume that the position and size of assembly instructions of each basic block in the memory are known. Knowing these data and the cache associativity, which is direct-mapped in our case, we can compute the set of ECBs.

- $\mathcal{H}$: event handler, which is implemented in the simulator for the two events $Prem(\tau_i, \tau_j, t)$ and $Exec(\tau_i, t)$, must be able to keep track of the remaining UCBs in the cache of a task and compute the CRPD based on this number.

  The following simulation data is managed and updated by the simulator during the scheduling simulation interval:

  - UCB_i^t: denotes the set of UCBs of τ_i in the cache at time t.
  - γ_i^t: denotes the delay added to the execution time of τ_i when it resumes at time t.
  - BRT: is a constant block reload time. We assume that the cache *block reload time* can be upperbounded by a constant and is denoted as BRT. This is a strong assumption even though it is popularly taken in prior works [21, 2, 34]. For modern architectures it could be challenging to determine a correct bound on the BRT due to timing anomalies.

  Then $\mathcal{H}$ can be expressed by two event handlers.

  - For each $Prem(\tau_i, \tau_j, t)$, UCB_i^t is updated as follows:

$$\mathrm{UCB}_i^t = \mathrm{UCB}_i^{t-1} - (\mathrm{UCB}_i^{t-1} \cap \mathrm{ECB}_j) \tag{1}$$

    When a task τ_i is preempted by a higher priority task τ_j at time t, we remove $\mathrm{UCB}_i^{t-1} \cap \mathrm{ECB}_j$ cache blocks from the set of UCBs of τ_i in the cache. The preemption events mentioned here include both direct and nested preemptions. By tracking the sets of UCBs of tasks in the cache and updating them at the preemption event, nested preemptions are systematically taken into account.

  - For each $Exec(\tau_i, t)$, γ_i^t is computed as follows:

$$\gamma_i^t = |\mathrm{UCB}_i - \mathrm{UCB}_i^t| \cdot \mathrm{BRT} \tag{2}$$

    In addition, because the CRPD is accounted for, we consider that τ_i has reloaded all of its UCB in the cache. Thus, we have to perform

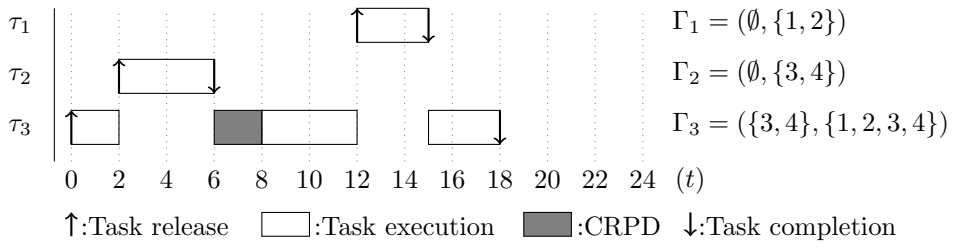

Figure 2: CRPD is computed during the schedule. Cache access profiles $\Gamma_i = (\mathrm{UCB}_i, \mathrm{ECB}_i)$ are given on the right. The CRPD added to the execution time of τ_3 is 2 when it is preempted by τ_2 and is 0 when it is preempted by τ_1.

the following assignment:

$$\mathrm{UCB}_i^t = \mathrm{UCB}_i \tag{3}$$

In this CRPD computation model, UCB_i^t is used as an internal data of the scheduling simulator to compute the CRPD instead of an exact representation of a task's data in the cache at a point in time

It is assumed that any partial execution of a task needs to load all of its UCBs into the cache. In addition, a task uses all of its ECBs. These assumptions are pessimistic considering the real execution of a task. However, to relax this assumption, one must provide information about which memory blocks are being used at a given instant. In other words, we would need a more detailed task model in which each unit of task capacity is linked to one or several memory blocks or cache blocks. In this case, CRPD could be computed based on which UCBs are being used at a given instant. However, as far as we know, there is no timing analysis tool that can provide such information. Relaxing this assumption requires a timing analysis technique, which is beyond the scope of this article.

$\mathcal{C}^{\mathrm{on}}$ is less pessimistic in the sense that it only takes into account the CRPD introduced by reloading necessary cache blocks. In Figure 2, we illustrate the schedule provided by $\mathcal{C}^{\mathrm{on}}$. As we can see, τ_3 only experiences CRPD when it is preempted by τ_2 at time $t = 2$ because τ_2 evicts two UCBs of τ_3, which are cache blocks 3 and 4. In addition, because τ_1 does not evict any UCBs of τ_3, the CRPD at time $t = 12$ is 0.

5. $\mathcal{C}^{\text{on-lim}}$ - an improved online CRPD computation model

In this section, we first analyze the limitations of the two CRPD computation models presented in section 4 and illustrate these limitations with an example. Then we discuss the cause of these limitations and propose an approach to address them. Our proposed approach is formalized into a new CRPD computation model named $\mathcal{C}^{\text{on-lim}}$.

Task	C_i	T_i	D_i	O_i	Π_i	$\mathbf{UCB_i}$	$\mathbf{ECB_i}$
τ_1	4	12	12	0	3	$\emptyset$	{1,2}
τ_2	8	24	24	0	2	{3}	{3,4}
τ_3	8	24	24	0	1	{1,2}	{1,2}

Table 1: Task set example.

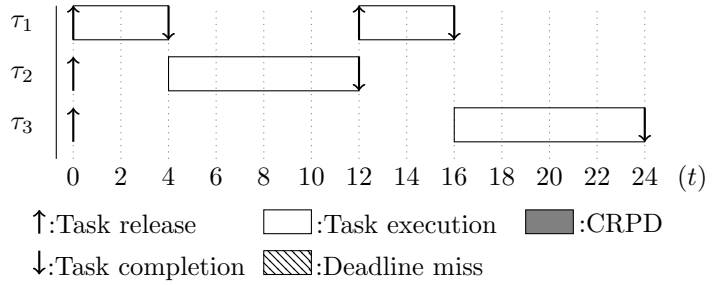

Figure 3: Schedule of the task set in Table 1 in the first 24 units of time. All deadlines are met. There is no preemption.

5.1. Limitations of previous CRPD computation models

Both CRPD computation models presented previously do not account for the executed capacity of a task when computing CRPD. Ignoring this element makes CRPD-aware scheduling simulation pessimistic and can also lead to a misconception regarding sustainability analysis, which is described later in the section.

We provide in Table 1 an example in which there is an occurrence of both pessimistic CRPD computation and non-sustainable scheduling with $\mathcal{C}^{\text{on}}$. The scheduling of this task set in the first 24 units of time is given in Figure 3, which shows that all deadlines are met. Regarding the job of task τ_3 released at $t = 0$, it experiences the interference from higher priority tasks τ_1 and τ_2.

In Figure 4, we assume that the C_2 is reduced to 7 instead of 8. Because of this change, the job of τ_2 completes at time $t = 11$. Then, τ_3 starts at time $t = 11$ and is preempted by τ_1 at time $t = 12$. Later, τ_3 resumes at time $t = 16$. Regarding $\mathcal{C}^{\text{on}}$, the CRPD added to the capacity of τ_3 at time $t = 16$ is computed as follows:

1 at time t = 11: τ_3 starts, $\text{UCB}_3^{11} = 1, 2$
2 at time t = 12: τ_3 is preempted by τ_1, all of its UCBs are evicted $\rightarrow \text{UCB}_3^{12} = \emptyset$
3 at time t = 16: $\text{UCB}_3^{16} = \emptyset$ as τ_3 does not execute from t = 12 to 16
4 at time t = 16: $\gamma_3^{16} = |\text{UCB}_3 - \text{UCB}_3^{16}| \cdot \text{BRT} = |\{1, 2\} - \emptyset| \cdot 1 = \mathbf{2}$

The added CRPD is 2 units of time and τ_3 missed its deadline. We can see that there are two problems illustrated in this figure:

1. The computed CRPD by applying $\mathcal{C}^{\text{on}}$ is 2 time units. This value is larger than the executed capacity of task τ_3, which is 1 time unit.

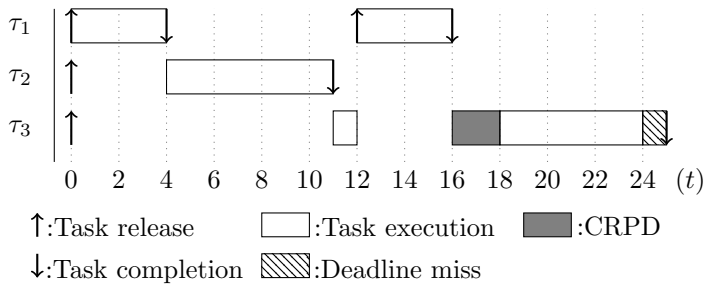

Figure 4: Non-sustainable scheduling regarding capacity parameter with $\mathcal{C}^{\text{on}}$. The capacity of τ_2 is reduced to $7 < C_2 = 8$. τ_1 preempts τ_3 at time $t = 12$

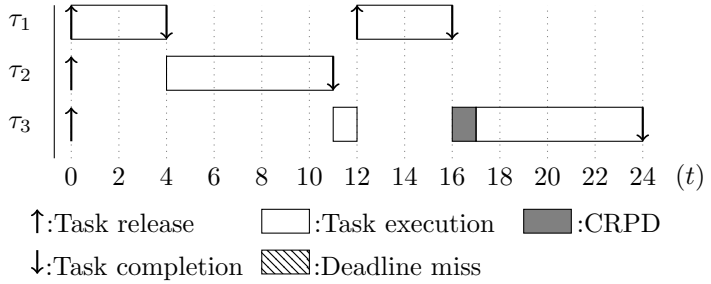

Figure 5: Sustainable scheduling regarding the capacity parameter by limiting the CRPD added. The capacity of τ_2 is $7 < C_2 = 8$. τ_1 preempts τ_3 at $t = 12$

2. The behavior of decreasing capacity of task τ_2, which is considered a better behavior, makes the task set unschedulable. Thus, scheduling simulation with $\mathcal{C}^{\text{on}}$ is non-sustainable with regard to the capacity parameter.

We proceed by presenting an improved CRPD computation model that can handle both problems in the next section.

5.2. $\mathcal{C}^{on\text{-}lim}$: an improved CRPD computation model

In this section, we first discuss the basic idea that leads to the proposition of $\mathcal{C}^{\text{on-lim}}$ CRPD computation model then present its formalization.

Our idea comes from an initial observation in [39] that the CRPD is at most be proportional to the executed capacity. As pointed out in [17], the maximum amount of CRPD is related to the amount of useful information that has to be reloaded into the cache. Indeed, if a task is preempted shortly after it starts, it has not yet loaded all of the UCBs and will therefore not experience the maximum CRPD.

Based on these observations, we design a computation model in which the computed CRPD is limited by the executed capacity of a task. As a result, the execution time gained by an early start is not neglected by the effect of CRPD, which is the case illustrated in Figure 4.

Our idea is illustrated in Figure 5. In this figure, the CRPD, which is computed at time $t = 16$, due to τ_1 preempting τ_3 is limited by the previously executed capacity of τ_3. In this case, we can obtain a feasible schedule and τ_3 can meet its deadline.

The proposed CRPD computation model named $\mathcal{C}^{\text{on-lim}}$ is defined as follows:

- $\Gamma = \{(\text{UCB}_i, \text{ECB}_i) \,|\, \forall \tau_i \in \mathcal{T}\}$: For each task, a set of UCBs and ECBs are statically computed before simulation as described in the case of $\mathcal{C}^{\text{on}}$.
- $\mathcal{H}$: the event handler for two events $Prem(\tau_i, \tau_j, t)$ and $Exec(\tau_i, t)$ is similar to $\mathcal{C}^{\text{on}}$ but with one added constraint: *the interval of time that a task spends to load memory blocks into the cache cannot be larger than the interval of time in which it has been executed.* In other words, if task τ_i executes non-preemptively in an interval of time Δ, there cannot be more than $\min(|\text{UCB}_i|, \left\lfloor \frac{\Delta}{\text{BRT}} \right\rfloor)$ UCBs loaded into the cache. As a result, we can state that if task τ_i executes non-preemptively in an interval of time Δ and loaded ρ_i UCBs into the cache, we have:

$$\rho_i \cdot \text{BRT} \leq \Delta \tag{4}$$

  In $\mathcal{C}^{\text{on-lim}}$, we introduce notation ρ_i^t that denotes the number of UCBs of τ_i in the cache at time t. When τ_i executes non-preemptively in an interval of time $[t, t + \Delta)$, the number of UCBs in the cache at time $t + \Delta$ is computed as follows:

$$\rho_i^{t+\Delta} = \min(|\text{UCB}_i|, \rho_i^t + \left\lfloor \frac{\Delta}{\text{BRT}} \right\rfloor) \tag{5}$$

  In Equation 5, in case τ_i starts execution at time t, there is no UCB loaded and $\rho_i^t = 0$. In case τ_i resumes execution at time t after it is preempted, $\rho_i^t > 0$ unless all UCBs loaded before t are evicted. We also take into account the fact that $\rho_i^{t+\Delta} \leq |\text{UCB}_i|$ with the min operator.

  Event handler $\mathcal{H}$ can be presented as follows:

  - For each $Prem(\tau_i, \tau_j, t)$, UCB_i^t is updated as in $\mathcal{C}^{\text{on}}$:

$$\text{UCB}_i^t = \text{UCB}_i^{t-1} - (\text{UCB}_i^{t-1} \cap \text{ECB}_j) \tag{6}$$

  - Event $Exec(\tau_i, t)$ is handled as follows. When τ_i resumes execution at time t after being preempted, we compute the preemption cost γ_i^t.

$$\gamma_i^t = \min(|\text{UCB}_i - \text{UCB}_i^{t-1}|, \rho_i^{t-1}) \cdot \text{BRT} \tag{7}$$

$$\rho_i^t = \max(0, \rho_i^{t-1} - |\text{UCB}_i - \text{UCB}_i^t|) \tag{8}$$

    Equation 7 computes the preemption cost with two upper-bounds deduced by the number of UCBs evicted and the number of UCBs

loaded into the cache. Equation 8 updates the number of UCBs that remain in the cache.

We assume that if τ_i is not executed at time t, the number of UCBs loaded does not change.

$$\rho_i^t = \rho_i^{t-1} \tag{9}$$

As stated in Equation 8, ρ_i^t is updated at the time a task resumes execution. It conforms with our approach of computing the preemption cost at the time a task resumes execution. If ρ_i^t is updated at preemption events, Equation 7 is not valid anymore.

The schedule of task set in Table 1 with $\mathcal{C}^{\text{on-lim}}$ is shown in Figure 5. In this figure, the capacity of τ_2 is reduced to $7 < C_2 = 8$. τ_1 preempts τ_3 at time $t = 12$. Regarding $\mathcal{C}^{\text{on-lim}}$, when taking into account parameter ρ_3, we have:

1 at time t = 16: $\text{UCB}_3^{16} = \emptyset$, $\rho_3 = 1$
2 at time t = 16: $\gamma_3^{16} = \min(|\text{UCB}_3 - \text{UCB}^1 5_3|, \rho_3) \cdot \text{BRT} = \mathbf{1}$

From this example, we can see that the advantage of $\mathcal{C}^{\text{on-lim}}$ is two-fold. First, it is less pessimistic compared to classical CRPD computation models. In addition, the observed scheduling remains schedulable despite of a decrease in the capacity of τ_2.

By taking into account ρ_i^t in the computation of γ_i^t, we can prove the following theorem.

Theorem 1. *The added CRPD cannot be larger than the executed capacity of task τ_i. In other words, if τ_i executes in $n-1$ discrete intervals $[t_x, t_x + \Delta_x), x \in (0, 1, ..., n-1)$ and experiences preemption costs $\gamma_i^{t_y}, y \in (1, ..., n)$, we have:*

$$\sum_{y=1}^{n} \gamma_i^{t_y} \leq \sum_{x=0}^{n-1} \Delta_x \tag{10}$$

Proof. Because of the length of the proof, we only present a prove sketch in this section, and the complete proof is presented in Section 10.1.

The basic idea is to prove the theorem by induction. In the base case, we assume τ_i starts at t_0 and executes non-preemptively in interval $[t_0, t_0 + \Delta_0)$ then be preempted at $t_0 + \Delta_0$ by higher priority tasks. It resumes at time t_1 and executes non-preemptively in interval $[t_1, t_1 + \Delta_1)$. Later, it is preempted at time $t_1 + \Delta_1$ and resumes at time t_2. In the first interval, τ_i starts from an empty cache. In the second interval, τ_i can have UCBs in the cache at time t_1. We need to prove the following equations:

$$\gamma_i^{t_1} \leq \Delta_0 \tag{11}$$

$$\gamma_i^{t_1} + \gamma_i^{t_2} \leq \Delta_0 + \Delta_1 \tag{12}$$

The inductive step is established by assuming τ_i executes in n discrete intervals $[t_x, t_x + \Delta_x), x \in (0, 1, ..., n)$ and experiences preemption costs $\gamma_i^{t_y}, y \in (1, ..., n+1)$ and we have:

$$\sum_{x=1}^{n} \gamma_i^{t_x} \leq \sum_{y=0}^{n-1} \Delta_y \tag{13}$$

then, we prove that:

$$(\sum_{x=1}^{n} \gamma_i^{t_x}) + \gamma_i^{t_{n+1}} \leq (\sum_{y=0}^{n-1} \Delta_y) + \Delta_n \tag{14}$$

□

To conclude, in this section, we describe and prove an important property of the improved CRPD computation model $\mathcal{C}^{\text{on-lim}}$. In the next three sections, we discuss in detail the three properties: sustainability, feasibility interval and pessimism of $\mathcal{C}^{\text{on-lim}}$, respectively. Sustainability and feasibility interval are proved by mathematical means in Section 6 and 7 while pessimism is evaluated by experimental means with synthetic task sets in Section 8.

6. Sustainability analysis of scheduling simulation with $\mathcal{C}^{\text{on-lim}}$

In this section, we recall the definition of sustainability, discuss sustainability analysis of classical CRPD computation models and analyze sustainability of scheduling simulation with $\mathcal{C}^{\text{on-lim}}$.

Definition 10 (Sustainability [6]). *A given scheduling policy and/or a schedulability test is sustainable if any system that is schedulable under its worst-case specification remains so when its behavior is better than the worst-case. The term "better" means that the parameters of one or more individual task(s) are changed in any, some, or all of the following ways: (1) reduced capacity, (2) larger period and (3) larger relative deadline.*

These changes are originally considered "better" behaviors because of the following observation. We assume that a job of τ_i released at time t and has a deadline at $t + D_i$. In preemptive scheduling context, a job of τ_i released at time t experiences the interference from higher priority tasks, denoted I_i^t, in the interval $[t, t + D_i)$. This interference is execution time of higher priority taks, which includes their capacities and CRPD. The job of τ_i is feasible if the following condition is satisfied [40]:

$$C_i + I_i^t \leq D_i \tag{15}$$

- Reduced capacity means a decrease in the value of C_i. In addition, it can be a reduction in the capacities of higher priority tasks, thus I_i^t is also decreased.
- Larger periods decreases I_i^t by reducing the number of higher priority tasks released in the interval $[t, t + D_i)$.
- Larger relative deadlines increases D_i.

All the changes are supposed to make the feasibility condition easier to be satisfied.

A schedulability test, such as scheduling simulation or worst-case response time analysis, must be aware of unpredictable changes in task parameters even if these are considered better behaviors. Indeed, for all realistic systems, considerable variability in execution times are to be expected.

The problem related to CRPD in sustainability analysis for fixed-priority preemptive scheduling context can be defined as follows. Two parameter changes (1) reduced capacities and (2) larger periods could decrease I_i^t by Δ. However, as shown by examples in the next sections, these two changes can increase the number of preemptions, which later lead to an increase of CRPD by γ. If $\gamma > \Delta$ due to CRPD, these parameter changes, which are considered a better scenario, may increase the interference and makes the task set to be unschedulable. Later in the section, we also discuss the third parameter change (3) increasing deadlines and its impact in the FPP scheduling context.

6.1. Reduced capacity

We prove that scheduling simulation with $\mathcal{C}^{\text{on-lim}}$ is sustainable regarding the capacity parameter.

Theorem 2. *Assuming $\mathcal{C}^{on\text{-}lim}$, a decrease of Δ in the execution time of higher priority tasks can only leads to a maximum increase of γ the execution time of a job of lower priority task where $\gamma \leq \Delta$.*

Proof. We prove this theorem by reasoning. We assume that in the new schedule, due to a decrease of Δ in the execution time of higher priority tasks, a jobs of τ_i executes in n additional intervals $[t_x, t_x + \Delta_x)$ with $\Delta = \sum_{x=0}^{n} \Delta_x$. Without losing generality, we assume that n intervals are discrete but arranged consecutively in the new schedule. The CRPD added due to these intervals can be isolated by assuming τ_i starts execution from an empty cache at time t_1. This reasoning can be backed up by the fact that τ_i already had to pay for any CRPD happens outside of these intervals in the original schedule. Applying Theorem 1, we have:

$$\gamma = \sum_{y=1}^{n+1} \gamma_i^{t_y} \leq \sum_{x=0}^{n} \Delta_x = \Delta \tag{16}$$

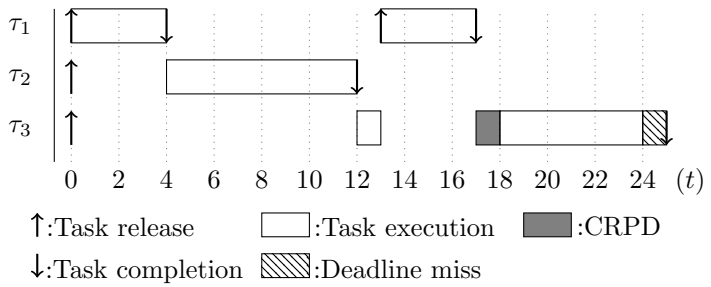

Figure 6: Non-sustainable scheduling simulation regarding period parameter with $\mathcal{C}^{\text{on-lim}}$. The period of τ_1 is increased to $13 > T_1 = 12$. τ_1 preempts τ_3 at $t = 13$. τ_3 missed its deadline at $t = 24$.

We now prove that a decrease in the execution time of higher priority tasks does not create additional interference to lower priority tasks. The decrease in execution time is always larger than or equal to the CRPD introduced by the possible increase in the number of preemptions.

Theorem 3. *Scheduling simulation with $\mathcal{C}^{on\text{-}lim}$ is sustainable with regard to the capacity parameter.*

Proof. Suppose that a system is deemed schedulable; i.e., for all jobs of all tasks, the feasibility condition defined in Equation 15 is satisfied.

We evaluate a job of task τ_i following the feasibility condition. A decrease in the execution time of higher priority tasks can introduce a new interference denoted $I_i'^t$. We have $I_i'^t = I_i^t - \Delta + \gamma$, where Δ is the decrease in execution time and γ is the CRPD introduced by this change. We have $\gamma \leq \Delta$ according to Theorem 2. Thus, $I_i'^t \leq I_i^t$. To conclude, the following equation holds.

$$C_i + I_i'^t \leq C_i + I_i^t \leq D_i \tag{17}$$

This means that if τ_i can meet the deadline with $C_i + I_i^t$, it can still meet the deadline with the new interference $I_i'^t$. We conclude that any job of task τ_i is still feasible. □

6.2. Larger period

The change in period can unfortunately create additional preemptions and jeopardize the schedulability even with $\mathcal{C}^{\text{on-lim}}$.

Theorem 4. *Scheduling simulation with $\mathcal{C}^{on\text{-}lim}$ is not sustainable with regard to the period parameter.*

Proof. We prove this theorem by using a counter example. Changing period of tasks can lead to unschedulable task sets even with $\mathcal{C}^{\text{on-lim}}$. This problem is illustrated in Figure 6.

In this figure, the period of task τ_1 in Table 1 is changed to 13. As a result, at time $t = 12$, τ_3 can execute. At time $t = 13$, τ_3 is preempted by τ_1 and there is one unit of preempted cost added to the capacity of τ_3. Finally, τ_3 missed the deadline at time $t = 24$. □

We can observe that the change in the period of τ_1 does not decrease the interference from higher priority tasks to the job of τ_3 released at $t = 0$, even in the case CRPD is not considered. Furthermore, it also creates one additional preemption. When CRPD is taken into account, the interference is increased. Thus, τ_3 missed its deadline at time $t = 24$. This observation above restricts the usage of scheduling simulation as a schedulability test for pure periodic task sets. In case of aperiodic task sets, scheduling simulation can only be used as a necessary condition.

6.3. Larger relative deadline

FFP scheduling with $\mathcal{C}^{\text{on-lim}}$ is sustainable with regard to the deadline parameter as the schedule generated by a FFP scheduler itself is independent from the deadlines. In other words, deadlines do not influence scheduling decisions. In FFP scheduling context, an increase in relative deadlines is simply a less stringent timing constraint if we do not reassign task priorities. In this case, larger deadlines neither decrease the execution time of tasks, nor create additional preemptions, nor change the CRPD. We do not investigate the case where task priorities are reassigned according to new deadlines. This case is indeed more complex because a new priority ordering can result in a higher number of preemptions and preemption cost.

6.4. Summary

To sum up, in this section, we have investigated the sustainability analysis of scheduling simulation with $\mathcal{C}^{\text{on-lim}}$ regarding the three task parameter changes: capacity, period and relative deadline. We have proved that scheduling simulation with $\mathcal{C}^{\text{on-lim}}$ is sustainable regarding capacity and relative deadline parameter and is not sustainable regarding period parameter. The result means that scheduling simulation with $\mathcal{C}^{\text{on-lim}}$ is an improvement compared to classical models. It can be used to verify and guarantee the schedulability of periodic tasks where the changes in the period parameter are predictable and unsustainability is not an issue for scheduling simulation. However, it cannot be applied to task sets with sporadic tasks.

7. Feasibility interval of scheduling simulation with $\mathcal{C}^{\text{on-lim}}$

In this section, we establish a feasibility interval $\mathcal{F}$ to determine the schedulability of a periodic task set $\mathcal{T}$, running on a uniprocessor system $\mathcal{M}$ scheduled by a fixed priority preemptive scheduler $\mathcal{S}$, with regard to $\mathcal{C}^{\text{on-lim}}$.

We analyze two properties that are used to define the feasibility interval in the literature namely stabilization time and periodic behavior. As presented

in [40] and [7], a well established result on these properties regarding RTES without cache memory is that for a task τ_i, after an initial stabilization time S_i defined later in the section, the execution of τ_i is periodic in the interval P_i, which is the level-i hyper-period defined as follows:

Definition 11 (Level-i hyper-period [40]). *The level-i hyper-period P_i of task τ_i is equal to the least common multiplier of the periods of τ_i and its higher priority tasks. $P_i = lcm(T_i, (T_j \,|\, \forall \tau_i, \Pi_j > \Pi_i))$.*

Then, the feasibility interval of τ_i is $[0, S_i + P_i)$. To determine the feasibility interval of our system model, we investigate the stabilization time when CRPD is taken into account. Second, we prove the periodic behavior of the task and establish the feasibility interval. We use this approach to present the feasibility interval of synchronous periodic tasks in Section 7.1 and asynchronous periodic tasks in Section 7.2.

7.1. Synchronous periodic tasks

The feasibility interval of a synchronous task set $\mathcal{T}$ can be deduced as follows:

Theorem 5. *For a synchronous task set $\mathcal{T}$ running on $(\mathcal{M}, \mathcal{S}, \mathcal{C}^{on\text{-}lim})$, we have $\mathcal{F} = [0, H)$*

The hyper-period $H = lcm(T_i \,|\, \forall \tau_i)$ is the level-i hyper-period of the lowest priority task. The basic idea behind the proof of this theorem is utilizing the repetition property of the synchronous release at time 0 and H to deduce the schedulability condition.

Proof. First, we observe that a task set is schedulable only if all deadlines of jobs released in $[0, H)$ are met before H. It means all jobs must be completed before H. Second, there is no additional interference to consider in the second synchronous release at time H and the schedule in $[H, 2H)$ is simply identical to the schedule in $[0, H)$. In addition, the cache state at the end of the interval $[0, H)$ cannot negatively influence the schedule in the interval $[H, 2H)$. We recall that the capacity parameter is computed by assuming the worst-case in which a task starts from an empty cache. Another problem to consider is PCB-based WCET reductions [37]. In other words, capacities of tasks in the interval $[H, 2H)$ can be reduced if there are PCBs remain in the cache after time H. This problem is addressed as we have proved the sustainability of scheduling simulation with regard to the capacity parameter. Thus, it is only necessary to investigate the schedule in the interval $[0, H)$ to conclude about the schedulability of synchronous tasks. □

We note that the classical feasibility interval of $[0, D_i)$ for synchronous $\mathcal{T}$ on $(\mathcal{M}, \mathcal{S})$ is not applicable to systems with cache. The work presented in [20] has shown that the instant at time 0 when all tasks are released cannot be considered as the critical instant.

7.2. Asynchronous periodic tasks

For asynchronous periodic tasks, the concept of stabilization time was introduced in [40] and [7] to study the problem of feasibility interval. Because of the asynchronous releases, there could be an interval of time, in which lower priority tasks are released and executed while higher priority tasks are not released. In this interval, the scheduling of a task is considered to be not stabilized. Stabilization time is defined as follows:

Definition 12 (Stabilization time [40]). *The stabilization time S_i of task τ_i is an instant at when τ_i is released and all higher priority tasks were released before S_i.*

The computation of S_i is inductively defined as follows:
$S_1 = O_1$,
$S_i = max(O_i, O_i + \lceil \frac{S_{i-1} - O_i}{T_i} \rceil \cdot T_i)$ $(i = 2, 3, ..., n)$.

The basic idea of the initial stabilization time can be explained as follows. For a job $\tau_i[t]$ released before the stabilization time S_i, $0 < t \leq S_i$, not all higher priority tasks are released. Because of that reason, $\tau_i[t]$ does not have to compete with these tasks. Thus, the interference caused by higher priority tasks to $\tau_i[t]$ is lower than subsequent releases of τ_i. The execution of $\tau_i[t]$ in this low interference context could not be repeated in the future.

This stabilization time can be applied to CRPD-aware scheduling as the computation of stabilization time only needs to take into account the offsets and the periods of tasks. CRPD is a factor which affects the execution time of tasks but does not affect theirs release times and the stabilization times.

Periodic Behavior

In this section, we analyze the periodic behavior of systems with cache after the initial stabilization time. We make an initial observation that the execution of an individual task in an FPP scheduling context depends only upon its own properties and higher priority tasks. Thus, we define two conditions that make the execution of τ_i periodic:

- The first condition is that τ_i is released periodically in a fixed interval. This condition is satisfied in our system model because we only take into account periodic tasks.
- The second condition is that the interference from higher priority tasks to τ_i is periodic in a fixed interval. In other words, after a fixed interval, two jobs of τ_i experience the same interference. We proceed by proving that this second condition is also satisfied when CRPD is taken into account.

Based on the two conditions, if we can prove that the job of τ_i released at time t_i after the stabilization time S_i $(t_i = O_i + m \cdot T_i, \forall m \in \mathbb{N} | t_i \geq S_i)$, which is denoted $\tau_i[t_i], t_i \geq S_i$ and the job of τ_i released at time $t_i + k \cdot P_i$, which denoted $\tau_i[t_i + k \cdot P_i], \forall k \in \mathbb{N}^*$, experience identical interferences from higher priority task, we can conclude that τ_i is periodic in interval P_i after the initial stabilization time S_i. We establish the following theorem:

Theorem 6. *For all tasks τ_i, the job $\tau_i[t_i], t_i = O_i + m \cdot T_i, \forall m \in \mathbb{N} | t_i \geq S_i$ and the job $\tau_i[t_i + k \cdot P_i], k \in \mathbb{N}^*$ experience identical interferences from higher priority tasks $\tau_0, ..., \tau_{i-1}$.*

PROOF. This theorem is proved by induction.

Base case: We assume that tasks are ordered according to their priority. Considering the highest priority task τ_1, it experiences no interference. Thus, the schedule of τ_1 is periodic from S_1 with the period $P_1 = T_1$.

Consider the second highest priority task τ_2, since the task is ordered by priority, the periodic behaviors of τ_1 cannot be affected by τ_2. Because the schedule of task τ_1 is periodic from S_1 with the period $P_1 = T_1$, τ_1 is also periodic from S_2 ($S_2 \geq S_1$) with the period $P_2 = lcm\{P_1, T_2\}$.

The interference created by the capacity of τ_1 to the two jobs $\tau_2[t_2](t_2 = O_2 + m \cdot T_2, t_2 \geq S_2)$ and $\tau_2[t_2 + k \cdot P_2]$ $(\forall m \in \mathbb{N}, \forall k \in \mathbb{N}^*)$ is periodic and identical as we assumed that task capacity is constant.

The two jobs $\tau_2[t_2]$ and $\tau_2[t_2 + k \cdot P_2]$ experience identical sequence of preempting tasks. If $\tau_2[t_2]$ is firstly preempted by τ_1 at time $t_2 + \Delta$, then $\tau_2[t_2 + k \cdot P_2]$ will be firstly preempted by τ_1 at time $(t_2 + k \cdot P_2) + \Delta, 0 \leq \Delta < D_2$. Because the sets of UCBs and ECBs of τ_1 and τ_2 are fixed and both jobs of τ_2 have executed Δ units of time, the CRPD of the two preemptions are identical. The same argument can be applied to subsequent preemptions by τ_1 to $\tau_2[t_2]$ and $\tau_2[t_2 + k \cdot P_2]$ if they exist. We can conclude that the CRPD by τ_1 that $\tau_2[t_2]$ and $\tau_2[t_2 + k \cdot P_2]$ experience are identical.

From the deductions above, we can conclude $\tau_2[t_2]$ and $\tau_2[t_2 + k \cdot P_2]$ experience identical interference.

Inductive Case: we assume that Theorem 6 is true for $\tau_1, ..., \tau_i$. The objective now is to prove that it is also true for τ_{i+1}.

From the assumption, the schedule of the task subset $\{\tau_1, ..., \tau_i\}$ is periodic from S_i with the period P_i. Since the task is ordered by priority, the periodicity of the task subset cannot be changed by τ_{i+1}. Hence, we can deduce that the schedule of the task subset is also periodic from S_{i+1} ($S_{i+1} \geq S_i$) with the period $P_{i+1} = lcm\{P_i, T_{i+1}\}$ is identical. We can have the following deductions:

- The interference created by the capacity of $\tau_1, ..., \tau_i$ to the two jobs $\tau_{i+1}[t_{i+1}]$ $(t_{i+1} = O_{i+1} + m \cdot T_{i+1}, t_{i+1} \geq S_{i+1})$ and $\tau_{i+1}[t_{i+1} + k \cdot P_{i+1}]$ $(\forall k \in \mathbb{N}^*)$ is identical.
- The CRPD created by $\tau_0, ..., \tau_i$ preempting each other to the two jobs $\tau_{i+1}[t_{i+1}]$ and $\tau_{i+1}[t_{i+1} + k \cdot P_{i+1}]$ are identical.
- The two jobs $\tau_{i+1}[t_{i+1}]$ and $\tau_{i+1}[t_{i+1} + k \cdot P_{i+1}]$ experience identical sequence of preempting tasks. Thus, the CRPD created by $\tau_0, ...\tau_i$ preempting $\tau_{i+1}[t_{i+1}]$ and $\tau_{i+1}[t_{i+1} + k \cdot P_{i+1}]$ is identical.

From these deductions, we can conclude that $\tau_{i+1}[t_{i+1}]$ and $\tau_{i+1}[t_{i+1} + k \cdot P_{i+1}]$ experience identical interference. □

As the theorem regarding periodic behavior is proved, we can now prove the following theorem about the feasibility interval:

Theorem 7. *A task τ_i is feasible if and only if the deadlines corresponding to the releases of the task in $[0, S_i + P_i)$ are met.*

PROOF. From Theorem 6, we deduce that the execution of $\tau_1, \tau_2, ..., \tau_i$ in the interval $[S_i, S_i + P_i)$ and $[S_i + k \cdot P_i, S_i + (k + 1) \cdot P_i), \forall k \in \mathbb{N}^*$ are identical. Thus, it is sufficient to check if τ_i can meet its deadlines in only one interval of time plus the interval $[0, S_i)$. □

From Theorem 7, we can conclude that for a task set of n periodic tasks, the feasibility interval is $[0, S_n + P_n)$.

Theorem 8. *For asynchronous task set $\mathcal{T}$ running on $(\mathcal{M}, \mathcal{S}, \mathcal{C}^{on\text{-}lim})$, we have $\mathcal{F} = [0, S_n + P_n)$*

In the last two sections, we have established theoretical results which show the soundness of scheduling simulation with $\mathcal{C}^{\text{on-lim}}$. In the next section, we present an evaluation of $\mathcal{C}^{\text{on-lim}}$ and provide an implementation of a CRPD-aware scheduling simulator and its performance evaluation.

8. Pessimism of scheduling simulation with $\mathcal{C}^{\text{on-lim}}$

In this section, we present experiments to compare the pessimism of CRPD computation models. They are evaluated in terms of schedulability task set coverage, number of preemptions and preemption cost. These metrics allow us to show that our CRPD computation model is less pessimistic than the existing ones.

The base configuration of our experiments is as follows. The generation of task periods and cache utilizations in our experiments is based on the existing work in [10]. Task periods are uniformly generated from 5 ms to 500 ms, as found in most automotive and aerospace hard real-time applications. Generated task sets are harmonic in order to have a low feasibility interval and scheduling simulation interval. Task deadlines are implicit, i.e. $\forall i : D_i = T_i$. Processor utilization values are generated using the UUniFast algorithm [41]. Task execution times are set based on the processor utilizations and the generated periods: $\forall i : C_i = U_i \cdot T_i$, where U_i is the processor utilization of task i. Task offsets are uniformly distributed from 1 ms to 30 ms.

Cache memory is direct mapped. The number of cache blocks is equal to 256 and block reload time is 8 μs [10]. Cache usage of each task is determined by the number of ECBs. They are generated using UUniFast algorithm for a total cache utilization of 5. For each task, the set of UCBs is generated according to a uniform distribution ranging from 0 to the number of ECBs multiplied by a reuse factor RF=0.3. If the set of ECBs generated exceeds the cache size, it is limited to the cache size. For the generation of the UCBs, the original set of ECBs is used. For $\mathcal{C}^{\text{off}}$, we assume that $\gamma_i = |\text{UCB}_i| \cdot \text{BRT}$ every time that task τ_i is preempted. For $\mathcal{C}^{\text{on}}$ and $\mathcal{C}^{\text{on-lim}}$, CRPD computation model has been already described in Section 3.

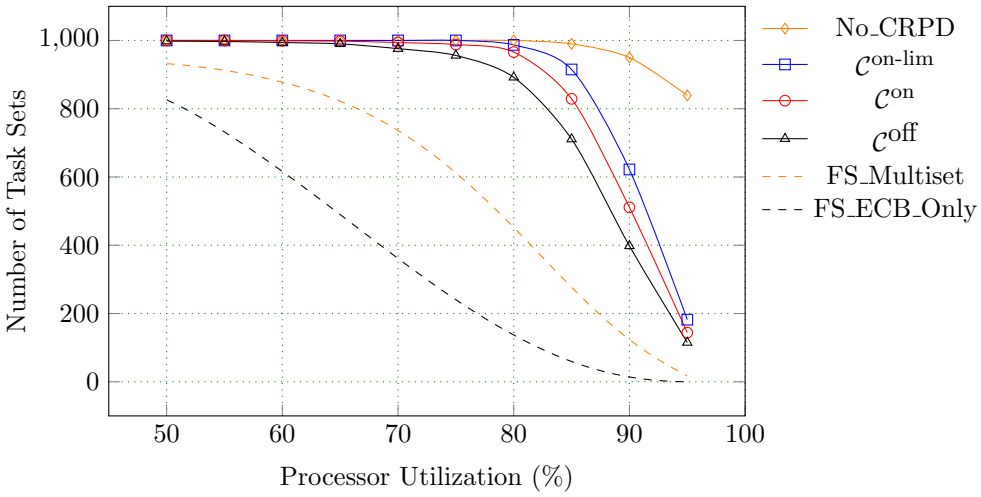

Figure 7: Schedulability task set coverage

8.1. Schedulability task set coverage

The objective of this experiment is to evaluate CRPD computation models in term of schedulability task set coverage. We performed scheduling simulations with randomly generated task sets with three CRPD computation models. Then, schedulability coverage is computed by taking into account the number of task sets found schedulable and the number of task sets generated.

$$sched\_coverage = \frac{\#task\_sets\_schedulable}{\#generated\_task\_sets}\% \tag{18}$$

As scheduling simulation is often used as a baseline to compare different response time analysis in previous research work, we also aim to show that $\mathcal{C}^{\text{on-lim}}$ achieves a higher schedulability task set coverage comparing to existing CRPD-aware feasibility tests. In our experiment, first, we generate task sets with the processor utilization (PU) varying from 50% to 90%. Task set size is fixed to 10 tasks. Second, we perform scheduling simulation over the feasibility interval to determine the schedulability of the generated task sets. In addition, in order to better illustrate the usage of simulation in schedulability task set coverage, we provide the number of task sets found schedulable by two well-known CRPD-aware feasibility tests: ECB-Only [2], and Combined Multi-set [10]. They are denoted respectively as FS_ECB_Only and FS_Multiset.

The experimental results are shown in Figure 7. $\mathcal{C}^{\text{on-lim}}$ gives the best schedulability task set coverage comparing to $\mathcal{C}^{\text{off}}$ and $\mathcal{C}^{\text{on}}$. If we only account for task sets with PU > 65, which is the point task sets are becoming unschedulable because of the CRPD effect, the number of schedulable ones found by $\mathcal{C}^{\text{on-lim}}$, $\mathcal{C}^{\text{on}}$, and $\mathcal{C}^{\text{off}}$ are 4706, 4432, and 3965 respectively for a total of 6000 generated task sets. The corresponding schedulability task set coverages are 78%, 73%, and 66%.

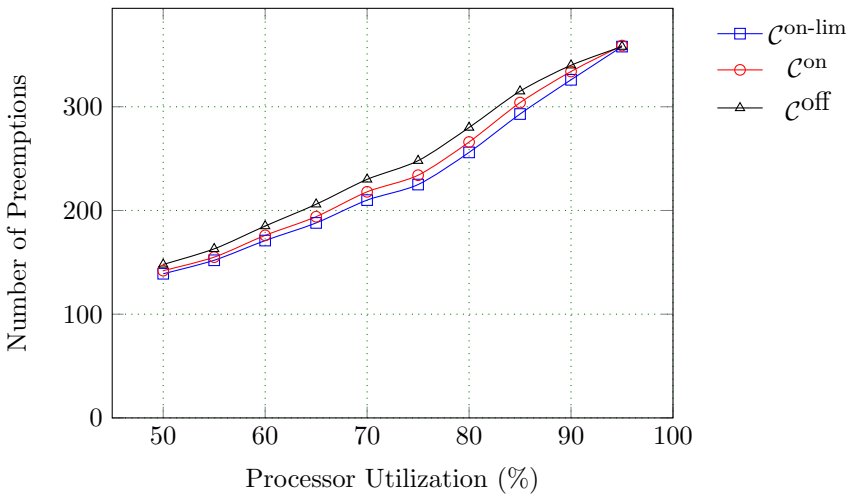

Figure 8: Average number of preemptions

We also observe that there is a gap between schedulability task set coverage by scheduling simulation and analytical methods. Our results are compared to two CRPD-aware feasibility tests namely ECB-Only [2] and Combined Multiset [10]. The evaluation conducted in [10] showed that ECB-Only is the most pessimistic test while Combined Multiset is the less pessimistic one in term of schedulability task set coverage.

Analytical methods are more pessimistic than scheduling simulation because of the following reason. CRPD-aware feasibility tests in [2, 21, 22, 9, 10] are designed with two assumptions. For a given task τ_i, the worst-case response time C_i is computed by assuming that all jobs of higher priority tasks executing within C_i preempt τ_i and CRPD is added for each preemption. The upper-bound on CRPD has been progressively reduced over the years by accounting for nested preemptions and the number of preemptions that have an actual influence on the response time. Nevertheless, it is still pessimistic as the tests ignore the observation in [17] that CRPD must be proportional to the executed capacity of the preempted task. In addition, if a task is released but not yet executed by the processor, it should not experience CRPD from higher priority tasks, which is the case in all CRPD computation models used in our simulations.

8.2. Preemption cost and number of preemptions

Two metrics that we evaluate in this experiment are the number of preemptions and the total preemption cost of three CRPD computation models. Experiment results are shown in Figures 8 and 9. First, regarding the number of preemptions, $\mathcal{C}^{\text{on-lim}}$ gives the lowest number. The reason is that it is the least pessimistic CRPD computation model thus task execution time is the shortest and resulting in less number of preemptions between tasks. Scheduling simulation with $\mathcal{C}^{\text{on-lim}}$ provides on average 7% less preemptions compared to

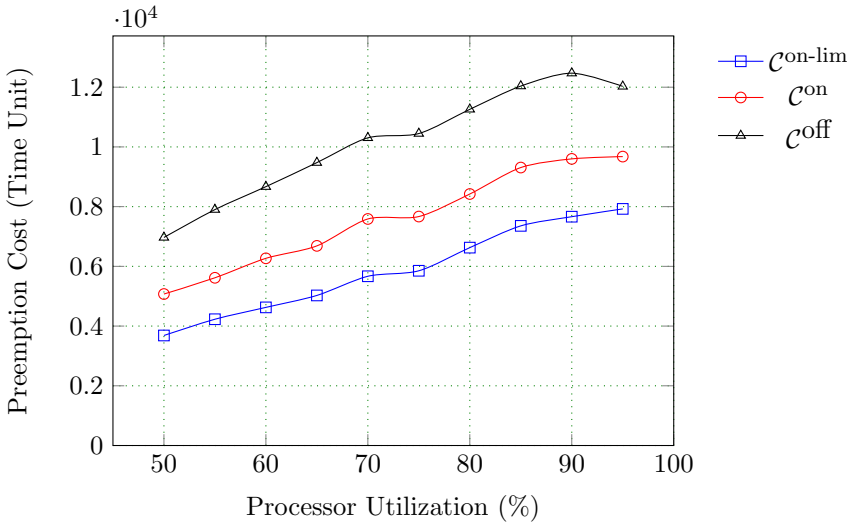

Figure 9: Average total preemption cost

$\mathcal{C}^{\text{off}}$ and 3% less preemptions compared to $\mathcal{C}^{\text{on}}$. In addition, $\mathcal{C}^{\text{on}}$ has 4% less preemption than $\mathcal{C}^{\text{off}}$.

Second, regarding the total preemption cost, the difference between the three CRPD computation models are more significant. On average, $\mathcal{C}^{\text{on-lim}}$ has 50% less preemption cost than $\mathcal{C}^{\text{off}}$ and 30% less than $\mathcal{C}^{\text{on}}$. Besides the difference regarding CRPD computation method between models, this result is also caused by our task generation strategy with small offsets. There are more preemptions that occur when tasks just execute a small proportion of their capacities. This result shows how CRPD is over-estimated in $\mathcal{C}^{\text{off}}$ and $\mathcal{C}^{\text{on}}$.

8.3. Case study: Mälardalen benchmark suite

In this section, we evaluate the different CRPD computation models using a set of programs taken from the Mälardalen [42] benchmark suite. The programs in this benchmark generally consist of a single path code followed by a main processing loop. The worst-case execution times and the cache access profiles, which are detailed in Table 2, are retrieved from [10]. The system was set up to model an ARM7 processor clocked at 100 MHz with a 2 KB direct-mapped instruction cache and a line size of 8 bytes, giving 256 cache sets, 4-byte instructions, and a BRT of 8 μs.

A task set is created from the above data by assigning periods and implicit deadlines such that all 15 tasks had equal utilization. The periods are generated by multiplying each execution time by a constant ($T_i = c \cdot C_i$) so that the PU of the task set is varied from 0.5 to 1.0 in step of 0.01. The tasks were assigned priorities in the rate monotonic priority order. As the exact placement of UCBs and ECBs is not provided in [10], we made two assumptions. ECBs are

Task	WCET	UCBs	ECBs
bs	445	5	35
minmax	504	9	79
fac	1252	4	24
fibcall	1351	5	24
insertsort	6573	10	41
loop3	13449	4	817
select	17088	15	151
qsort-exam	22146	15	170
fir	29160	9	105
sqrt	39962	14	477
ns	43319	13	64
qurt	214076	14	484
crc	290782	14	144
matmult	742585	23	100
bsort100	1567222	35	62

Table 2: Execution times and number of UCBs and ECB of tasks in the Mälardalen benchmark suite in obtained from [10]

	Breakdown Utilization
No CRPD	0.95
$\mathcal{C}^{\text{on-lim}}$	0.87
$\mathcal{C}^{\text{on}}$	0.85
$\mathcal{C}^{\text{off}}$	0.72

Table 3: Breakdown utilization for different CRPD computation models

computed by assuming that tasks are placed continuously in memory. Then, we choose to place the UCBs in a continuous group at a random location in each task.

The experiment is conducted as follows. We start with a low PU and run scheduling simulation with CRPD computation models. The PU is increased up to the point where the task set is not schedulable. It is called the breakdown utilization [10]; the maximum utilization at which a scaled version of the case study task set was deemed schedulable.

The results are provided in Table 3. Our first observation is that the difference between $\mathcal{C}^{\text{on-lim}}$ and $\mathcal{C}^{\text{on}}$ is only 2%. It can be explained by the low cache reuse factors of programs in the benchmark, which are varied from 0.11 to 0.56 with an average of 0.15. Second, $\mathcal{C}^{\text{off}}$ has a remarkable lower breakdown utilization. It shows the pessimism of $\mathcal{C}^{\text{off}}$ in case of a significant difference between the max and the min WCETs and periods. In this case, the large WCET tasks are preempted often and the CRPD is added even though the preempting tasks do not evict their UCBs in the cache.

```
1  event SCHED_START
2      for each task τ_i loop
3          τ_i.cUCB ← τ_i.UCB
4          ρ_i ← 0
5  event PREEMPTION
6      τ_j ← preempting_task
7      for each task τ_i preempted loop
8          τ_i.cUCB ← Remove(τ_i.cUCB, τ_j.ECB)
9  event RUNNING_TASK
10     τ_i ← executing_task
11     CRPD ← min((τ_i.UCB − τ_i.cUCB),ρ_i) ∗ BRT
12     ρ_i ← max((ρ_i − (τ_i.UCB − τ_i.cUCB)),0))
13     τ_i.cUCB ← τ_i.UCB
14     ρ_i ← ρ_i + 1
```

Figure 10: Event handlers update

9. CRPD-aware scheduling simulator implementation

In this section, we present our implementation of our CRPD-aware scheduling simulator in Cheddar [18], an open source scheduling analyzer, which is freely available for researchers and practitioners. In addition, we provide a performance evaluation in order to discuss some insights regarding the time it takes to simulate the scheduling with CRPD.

In Cheddar, an RTES is modeled as a set of hardware and software components. Hardware components are processors, cores and cache memories [43]. Software components are tasks and their sets of UCBs and ECBs, which are called cache access profiles. System models with computed cache access profiles are loaded into the scheduling simulator. Scheduling simulations can then be done and provide various data such as feasibility of the system, worst case response times of tasks, number of preemption, CRPD per task, total CRPD, etc. In Cheddar, we implemented the model of tasks following $\mathcal{C}^{\text{on-lim}}$ computation model. A detailed explanation of how Cheddar is extended to systems with cache can be found at [16].

The scheduling simulation in Cheddar works as follows. The scheduling is computed by three successive steps: computing priorities, managing ready tasks in queues and electing the next task to run. The elected task will receive the processor for the next unit of time.

The scheduling simulator records and handles different events raised during the simulation, such as task releases, task completions and shared resources lockings or unlockings. The result of the scheduling analysis is the set of events produced during the simulation.

We now explain the updates made to the scheduling simulator regarding $\mathcal{C}^{\text{on-lim}}$ computation model. The pseudo code of the event handlers is written in Figure 10. The notation τ_i`.cUCB` represents the set of UCBs of task τ_i in the cache. It is computed from a system model during the simulation. The function `Remove()` at line 8 is used to remove an element from this set.

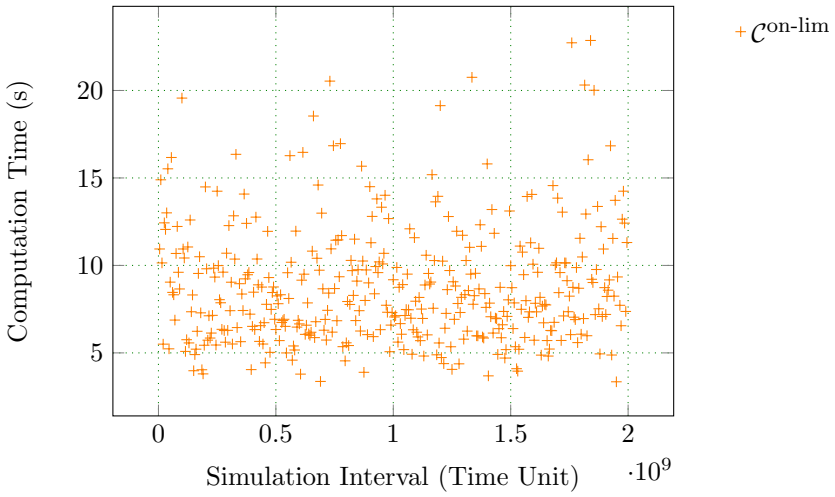

Figure 11: Computation time when simulation interval increases. Task set size = 10

At the start of the scheduling simulation, a SCHED_START event is raised. The WCET of a task is assumed to include the intrinsic cache block reloading time when a task is executed non-preemptively. On event SCHED_START, the set of UCBs of a task is assumed to be filled.

When a preemption occurs, a PREEEMPTION event is raised and the simulator computes the evicted UCBs of preempted tasks using the ECBs of the preempting task. The scheduler keeps track of the number of UCBs for each task.

When a task executes, a RUNNING_TASK event is raised. The scheduler first checks whether all the UCBs of this task are loaded in the cache. If so, the task continues its execution. If not, the task reloads the evicted UCBs. The CRPD is added to the remaining capacity of the task itself. In our implementation, CRPD is not added to the capacity of the preempted tasks at the preemption point but at the instant at which those tasks resume their execution. In addition, we also update the ρ_i parameter regarding the number of evicted UCBs.

Two experiments are performed to evaluate the performance of our scheduling simulator in terms of computation time, which is the time it takes to run the simulation. Theses experiments are conducted on a PC with an Intel Core i7-8650U (1.90GHz $\times$ 8) processor, having 16 GBs of memory, and running Ubuntu 18.04.4.

9.1. Simulation interval

In the first experiment, we fix the number of tasks in a task set at 10 tasks and increase the simulation interval from 0 to 2^{31} units of time by steps of $5 \cdot 10^6$. For each value of simulation interval, a new task set is generated. For each task set, the processor utilization is varied from 50% to 90%. As the data points obtained are scattered, we choose to only display the result with

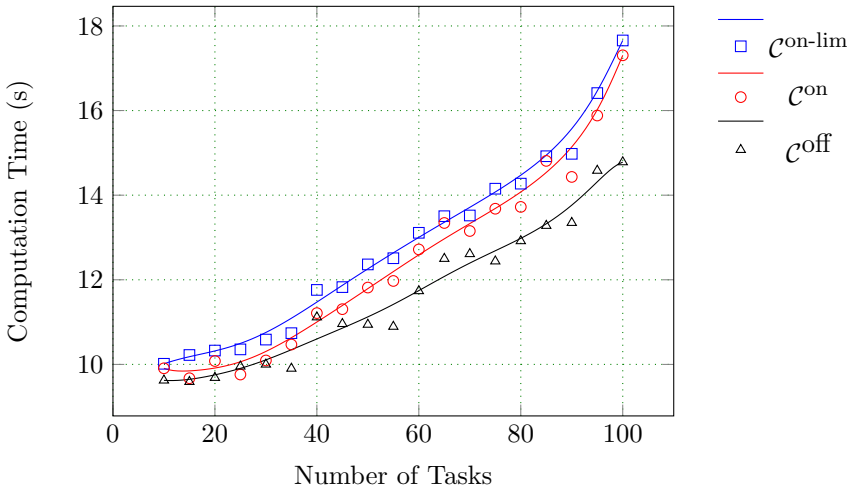

Figure 12: Computation time when task number increases. Simulation interval = 2.000.000 units of time

$\mathcal{C}^{\text{on-lim}}$. The result in Figure 11 shows that the maximum computation time is less than 25 seconds. However, we can see that the computation time does not increase linearly with the simulation interval. An explanation is that the processor utilization is also an important factors to the computation time. The simulation of task sets with low processor utilizations generates less events and thus requires less computation time.

9.2. Number of tasks

In the second experiment, we increase the number of tasks from 10 to 100 in increments of 5 and perform simulation over 2.000.000 units of time. The processor utilisation is fixed at 70% for the generated task sets. For each number of tasks, 20 task sets are generated and computation time is taken by the average. The result in Figure 12 shows that computation time increases from 10 to 18 seconds for $\mathcal{C}^{\text{on-lim}}$. Compared to the previous experiment, we can see that the increase in simulation time with regard to the number of tasks is higher than the simulation time.

We also observe that the computation time of $\mathcal{C}^{\text{on-lim}}$ is similar to $\mathcal{C}^{\text{on}}$, with an average difference of 0.39 second. The computation time of $\mathcal{C}^{\text{off}}$ is significantly less than the other two when the number of tasks is high. The maximum difference is 2.87 seconds, and the average difference is 1.17 seconds. The reason is that $\mathcal{C}^{\text{off}}$ does not require the simulator to keep track of the UCBs in the cache during simulation.

To sum up, in this section, we provide a performance evaluation of a CRPD-aware scheduling simulator by showing the computation time needed for a long period of simulation and a large number of tasks. Besides the two parameters, the computation time can also be affected by the number of preemptions because

preemptions generate additional events that must be handled by the simulator. However, the number of additional events are not significant comparing to the simulation period. For instance, the experiment in Section 8 shows a variation from 130 to 365 preemptions for a simulation period of 500,000 units of time.

10. Conclusion

In this article, we presented an improved CRPD computation model called $\mathcal{C}^{\text{on-lim}}$. We investigated two issues regarding sustainability and feasibility interval of scheduling simulation with this model. We have proved that scheduling simulation with $\mathcal{C}^{\text{on-lim}}$ is sustainable with regard to the capacity parameter. It allows us to use scheduling simulation to verify the schedulability of periodic tasks. However, with sporadic tasks, scheduling simulation has been proved to be non-sustainable and can only be used to detect the non-sustainability but not to guarantee the schedulability.

Experiments with synthetic task sets have shown that $\mathcal{C}^{\text{on-lim}}$ results in 78% schedulability task set coverage, which is 5% higher than $\mathcal{C}^{\text{on}}$ and 12% higher than $\mathcal{C}^{\text{off}}$. In addition, scheduling simulation with $\mathcal{C}^{\text{on-lim}}$ is less pessimistic in terms of number of preemptions, 7% less than $\mathcal{C}^{\text{on}}$ and 3% less than $\mathcal{C}^{\text{off}}$, and preemption cost, 50% less than $\mathcal{C}^{\text{on}}$ and 30% less than $\mathcal{C}^{\text{off}}$. CRPD computation models presented in this article have been implemented in Cheddar scheduling simulator.

As the architecture model used in this work is limited to L1 instruction cache, which is quite simple comparing to the ones supported in static analysis, future extensions must be implemented. The main challenge of taking into account the more complex cache architectures is to prove that we can preserve a feasibility interval and sustainable scheduling. Future work includes relaxing the assumption on the eviction of UCBs at the end of task execution to reduce the pessimism in our simulation by allowing PCB-based WCET reductions. In addition, we want to relax the assumption about a constant value for BRT as the difference between the best-case and the worst-case could be large in multi-core processors. Finally, a long-term objective is to address the case of multi-core processors. It requires us to take into account an additional effect named cache related migration delay and techniques to manage a shared cache amongst cores.

Acknowledgment

The authors would like to thank Sebastian Altmeyer for providing the implementation in C of the Combined Multi-set approach. This work and Cheddar are supported by Brest Métropole, Ellidiss Technologies, Région Bretagne, CD du Finistère and Campus France PESSOA programs number 27380SA and 37932TF.

Annex

10.1. Proof of Theorem 1

In this section, we prove Theorem 1 by induction.

- Base case: we assume τ_i starts at t_0 and executes non-preemptively in interval $[t_0, t_0 + \Delta_0)$ then is preempted at $t_0 + \Delta_0$ by higher priority tasks. Then, it resumes later at time t_1 and execute non-preemptively in interval $[t_1, t_1 + \Delta_1)$. Later, it is preempted at time $t_1 + \Delta_1$ and resumes at time t_2. In the first interval, τ_i starts from an empty cache. In the second interval, τ_i can have UCBs in the cache at time t_1. We need to prove the following equations:

$$\gamma_i^{t_1} \leq \Delta_0 \tag{19}$$

$$\gamma_i^{t_1} + \gamma_i^{t_2} \leq \Delta_0 + \Delta_1 \tag{20}$$

Equation 19 means that the CRPD computed at time t_1, which is $\gamma_i^{t_1}$, is less than the executed capacity of τ_i at time t_1, which is Δ_0. Equation 20 means that the total CRPD computed at time t_1 and t_2 is less than the executed capacity of τ_i at time t_2, which is $\Delta_0 + \Delta_1$.

We start by proving that $\gamma_i^{t_1} \leq \Delta_1$. Considering the preemption cost when τ_i resumes at time t_1, by applying Equation 7, we have:

$$\gamma_i^{t_1} = \min(|\mathrm{UCB}_i - \mathrm{UCB}_i^{t_1-1}|, \rho_i^{t_1-1}) \cdot \mathrm{BRT} \tag{21}$$

Because τ_i is not executed in the interval $[t_0 + \Delta, t_1 - 1)$, by applying Equation 9, we have:

$$\rho_i^{t_1-1} = \rho_i^{t_0+\Delta_0} = \min(|\mathrm{UCB}_i|, \rho_i^{t_0} + \left\lfloor \frac{\Delta_0}{\mathrm{BRT}} \right\rfloor) \tag{22}$$

In Equation 22, $\rho_i^{t_0}$ is the number of UCBs loaded into the cache at time $t = 0$ for τ_i. As we do not take into account PCBs and assume that $\rho_i^{t_0} = 0$, we have:

$$\rho_i^{t_1-1} = \min(|\mathrm{UCB}_i|, \left\lfloor \frac{\Delta_0}{\mathrm{BRT}} \right\rfloor) \tag{23}$$

By replacing $\rho_i^{t_1-1}$ in Equation 21 with Equation 23, we have:

$$\gamma_i^{t_1} = \min(|\mathrm{UCB}_i - \mathrm{UCB}_i^{t_1-1}|, \left\lfloor \frac{\Delta_0}{\mathrm{BRT}} \right\rfloor) \cdot \mathrm{BRT} \tag{24}$$

We can deduce that:

$$\gamma_i^{t_1} \leq \left\lfloor \frac{\Delta_0}{\text{BRT}} \right\rfloor \cdot \text{BRT} \leq \Delta_0 \tag{25}$$

Next, we prove that $\gamma_i^{t_1} + \gamma_i^{t_2} \leq \Delta_0 + \Delta_1$. Considering the preemption cost when τ_i resumes at time t_2, by applying Equation 7, we have:

$$\gamma_i^{t_2} = \min(|\text{UCB}_i - \text{UCB}_i^{t_2-1}|, \rho_i^{t_2-1}) \cdot \text{BRT} \tag{26}$$

Because τ_i is not executed in the interval $[t_1 + \Delta_1, t_2 - 1)$, by applying Equation 9, we have:

$$\rho_i^{t_2-1} = \rho_i^{t_1+\Delta_1} = \min(|\text{UCB}_i|, \rho_i^{t_1} + \left\lfloor \frac{\Delta_1}{\text{BRT}} \right\rfloor) \tag{27}$$

Applying Equation 8 to compute $\rho_i^{t_1}$, we have:

$$\rho_i^{t_1} = \max(0, \rho_i^{t_1-1} - |\text{UCB}_i - \text{UCB}_i^{t_1-1}|) \tag{28}$$

At this step, there are two cases. We have either $\rho_i^{t_1} = 0$ or $\rho_i^{t_1} = \rho_i^{t_1-1} - |\text{UCB}_i - \text{UCB}_i^{t_1-1}|$.

Case 1: assume that $\rho_i^{t_1} = 0$, by replacing $\rho_i^{t_1} = 0$ in Equation 27 and then $\rho_i^{t_2-1}$ in Equation 26, we have:

$$\gamma_i^{t_2} = \min\left(|\text{UCB}_i - \text{UCB}_i^{t_2-1}|, \left\lfloor \frac{\Delta_1}{\text{BRT}} \right\rfloor\right) \cdot \text{BRT} \tag{29}$$

From Equation 29, we can deduce that:

$$\gamma_i^{t_2} \leq \left\lfloor \frac{\Delta_1}{\text{BRT}} \right\rfloor \cdot \text{BRT} \tag{30}$$

Then, from Equation 30 and 21, we can deduce that:

$$\gamma_i^{t_1} + \gamma_i^{t_2} \leq \left(\left\lfloor \frac{\Delta_0}{\text{BRT}} \right\rfloor + \left\lfloor \frac{\Delta_1}{\text{BRT}} \right\rfloor\right) \cdot \text{BRT} \leq \Delta_0 + \Delta_1 \tag{31}$$

Case 2: assume that $\rho_i^{t_1} = \rho_i^{t_1-1} - |\text{UCB}_i - \text{UCB}_i^{t_1-1}|$, by replacing $\rho_i^{t_1-1}$ with the value computed in Equation 23, we have:

$$\rho_i^{t_1} = \min\left(|\text{UCB}_i|, \left\lfloor \frac{\Delta_0}{\text{BRT}} \right\rfloor\right) - |\text{UCB}_i - \text{UCB}_i^{t_1-1}| \tag{32}$$

By replacing $\rho_i^{t_1}$ in Equation 27 to deduce the bound on $\rho_i^{t_2-1}$ and then using this bound in Equation 26, we have:

$$\gamma_i^{t_2} \leq \left(\left\lfloor \frac{\Delta_0}{\text{BRT}} \right\rfloor - |\text{UCB}_i - \text{UCB}_i^{t_1-1}| + \left\lfloor \frac{\Delta_1}{\text{BRT}} \right\rfloor \right) \cdot \text{BRT} \tag{33}$$

From Equation 21, we can deduce that:

$$\gamma_i^{t_1} \leq |\text{UCB}_i - \text{UCB}_i^{t_1-1}| \cdot \text{BRT} \tag{34}$$

Taking the sum of the two equations, we can eliminate $|\text{UCB}_i - \text{UCB}_i^{t_1-1}|$ and conclude that:

$$\gamma_i^{t_1} + \gamma_i^{t_2} \leq \left(\left\lfloor \frac{\Delta_0}{\text{BRT}} \right\rfloor + \left\lfloor \frac{\Delta_1}{\text{BRT}} \right\rfloor \right) \cdot \text{BRT} \leq \Delta_0 + \Delta_1 \tag{35}$$

The base case is proved, we now prove the step case of the induction proof.

- **Step case**: assume that we have:

$$\sum_{x=1}^{n} \gamma_i^{t_x} \leq \sum_{y=0}^{n-1} \Delta_y \tag{36}$$

we have to prove that:

$$(\sum_{x=1}^{n} \gamma_i^{t_x}) + \gamma_i^{t_{n+1}} \leq (\sum_{y=0}^{n-1} \Delta_y) + \Delta_n \tag{37}$$

Considering the preemption cost when τ_i resumes at time t_{n+1}, by applying Equation 7, we have:

$$\gamma_i^{t_{n+1}} = \min(|\text{UCB}_i - \text{UCB}_i^{t_{n+1}-1}|, \rho_i^{t_{n+1}-1}) \cdot \text{BRT} \tag{38}$$

Because τ_i is not executed in the interval $[t_n + \Delta_n, t_{n+1} - 1)$, by applying Equation 9, we have:

$$\rho_i^{t_{n+1}-1} = \rho_i^{t_{n+\Delta}} = \min(|\text{UCB}_i|, \rho_i^{t_n} + \left\lfloor \frac{\Delta_n}{\text{BRT}} \right\rfloor) \tag{39}$$

Applying Equation 8 to compute $\rho_i^{t_n}$, we have:

$$\rho_i^{t_n} = \max(0, \rho_i^{t_n-1} - |\text{UCB}_i - \text{UCB}_i^{t_n-1}|) \tag{40}$$

At this step, there are two cases. We have either $\rho_i^{t_n} = 0$ or $\rho_i^{t_n} = \rho_i^{t_n-1} - |\mathrm{UCB}_i - \mathrm{UCB}_i^{t_n-1}|$.

Case 1: we assume that $\rho_i^{t_n} = 0$, from Equation 38, we can deduce that:

$$\gamma_i^{t_n+1} \leq \left\lfloor \frac{\Delta_n}{\mathrm{BRT}} \right\rfloor \cdot \mathrm{BRT} \tag{41}$$

From Equation 36, which is the assumption of the step case, and Equation 41, we can deduce that:

$$(\sum_{x=1}^{n} \gamma_i^{t_x}) + \gamma_i^{t_{n+1}} \leq (\sum_{y=0}^{n-1} \Delta_y) + \Delta_n \tag{42}$$

Case 2: we assume that $\rho_i^{t_n} = \rho_i^{t_n-1} - |\mathrm{UCB}_i - \mathrm{UCB}_i^{t_n-1}|$. We can successively deduce that $\rho_i^{t_n-1} = \min\left(|\mathrm{UCB}_i|, \rho_i^{t_n-2} + \left\lfloor \frac{\Delta_{n-2}}{\mathrm{BRT}} \right\rfloor\right)$. By applying this deduction successively until $\rho_i^{t_0}$, we have

$$\rho_i^{t_{n+1}-1} \leq \sum_{x=0}^{n-1} \left\lfloor \frac{\Delta_x}{\mathrm{BRT}} \right\rfloor - \sum_{x=0}^{n-1} |\mathrm{UCB}_i - \mathrm{UCB}_i^{t_x}| + \left\lfloor \frac{\Delta_n}{\mathrm{BRT}} \right\rfloor \tag{43}$$

By replacing the value of $\rho_i^{t_{n+1}-1}$ in Equation 38, we can deduce that:

$$\gamma_i^{t_{n+1}} \leq \left(\sum_{x=0}^{n-1} \left\lfloor \frac{\Delta_x}{\mathrm{BRT}} \right\rfloor - \sum_{x=0}^{n-1} |\mathrm{UCB}_i - \mathrm{UCB}_i^{t_x}| + \left\lfloor \frac{\Delta_n}{\mathrm{BRT}} \right\rfloor \right) \cdot \mathrm{BRT} \tag{44}$$

From Equation 7, we can also deduce that:

$$\sum_{x=1}^{n} \gamma_i^{t_x} \leq \sum_{x=0}^{n-1} |\mathrm{UCB}_i - \mathrm{UCB}_i^{t_x}| \cdot \mathrm{BRT} \tag{45}$$

From Equation 44 and Equation 45, we can compute $(\sum_{x=1}^{n} \gamma_i^{t_x}) + \gamma_i^{t_{n+1}}$, eliminate $\sum_{x=0}^{n-1} |\mathrm{UCB}_i - \mathrm{UCB}_i^{t_x}|$ and deduce Equation 42, which is what we want to prove.

References

References

[1] J. Krodel, Commercial Off-the-shelf Real-time Operating Systems and Architectural Considerations, Office of Aviation Research, Federal Aviation Administration, 2004.

[2] J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, A. Wellings, Adding instruction cache effect to schedulability analysis of preemptive real-time systems, in: Proceedings of the 2^{nd} IEEE Real-Time Technology and Applications Symposium (RTAS), 1996, pp. 204–212.

[3] R. Pellizzoni, M. Caccamo, Toward the predictable integration of real-time COTS based systems, in: 28th International Real-Time Systems Symposium (RTSS), IEEE, 2007, pp. 73–82.

[4] S. Planning, The economic impacts of inadequate infrastructure for software testing, National Institute of Standards and Technology.

[5] J. Goossens, E. Grolleau, L. Cucu-Grosjean, Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms, Real-Time Systems 52 (6) (2016) 808–832. doi:10.1007/s11241-016-9256-1.
URL https://hal.inria.fr/hal-01419704

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

[7] J. Goossens, R. Devillers, The non-optimality of the monotonic priority assignments for hard real-time offset free systems, Real-Time Systems 13 (2) (1997) 107–126.

[8] G. Phavorin, P. Richard, J. Goossens, T. Chapeaux, C. Maiza, Scheduling with preemption delays: anomalies and issues, in: Proceedings of the 23rd International Conference on Real Time and Networks Systems, ACM, 2015, pp. 109–118.

[9] J. Staschulat, R. Ernst, Scalable precision cache analysis for preemptive scheduling, ACM SIGPLAN Notices 40 (7) (2005) 157–165.

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

[11] W. Lunniss, S. Altmeyer, R. I. Davis, A comparison between fixed priority and edf scheduling accounting for cache related pre-emption delays.

[12] M. González Harbour, J. Gutiérrez García, J. Palencia Gutiérrez, J. Drake Moyano, Mast: Modeling and analysis suite for real time applications, in: Real-Time Systems, 13th Euromicro Conference on, 2001., IEEE, 2001, pp. 125–134.

[13] R. Urunuela, A. Deplanche, Y. Trinquet, Storm, a simulation tool for real-time multiprocessor scheduling evaluation, in: Proceeding of the 15th Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, 2010.

[14] Y. Chandarli, F. Fauberteau, D. Masson, S. Midonnet, M. Qamhieh, et al., Yartiss: A tool to visualize, test, compare and evaluate real-time scheduling algorithms, in: Proceedings of the 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems, 2012, pp. 21–26.

[15] M. Chéramy, P.-E. Hladik, A.-M. Déplanche, S. Dal Zilio, Simulation of real-time scheduling algorithms with cache effects, in: 6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems, 2015.

[16] H. N. Tran, F. Singhoff, S. Rubini, J. Boukhobza, Cache-aware real-time scheduling simulator: Implementation and return of experience, ACM SIGBED Rev. 13 (1) (2016) 22–28. doi:10.1145/2907972.2907975. URL http://doi.acm.org/10.1145/2907972.2907975

[17] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, G. Buttazzo, Optimal selection of preemption points to minimize preemption overhead, in: 23rd Euromicro Conference on Real-Time Systems (ECRTS), IEEE, 2011, pp. 217–227.

[18] F. Singhoff, J. Legrand, L. Nana, L. Marcé, Cheddar: a flexible real time scheduling framework, ACM SIGAda Ada Letters 24 (4) (2004) 1–8.

[19] G. Phavorin, Hard real-time scheduling subjected to cache-related preemption delays, Ph.D. thesis (Sept 2016).

[20] G. Phavorin, P. Richard, J. Goossens, C. Maiza, L. George, T. Chapeaux, Online and offline scheduling with cache-related preemption delays, Real-Time Systems (2017) 1–38.

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

[22] H. Tomiyama, N. D. Dutt, Program path analysis to bound cache-related preemption delay in preemptive real-time systems, in: Proceedings of the eighth international workshop on Hardware/software codesign, ACM, 2000, pp. 67–71.

[23] W. Lunniss, S. Altmeyer, C. Maiza, R. I. Davis, Integrating cache related pre-emption delay analysis into EDF scheduling, University of York, York, UK, Technical Report YCS-2012-478.

[24] P. M. Yomsi, Y. Sorel, Extending rate monotonic analysis with exact cost of preemptions for hard real-time systems, in: 19th Euromicro Conference on Real-Time Systems (ECRTS), 2007, IEEE, 2007, pp. 280–290.

[25] J. Y.-T. Leung, J. Whitehead, On the complexity of fixed-priority scheduling of periodic, real-time tasks, Performance evaluation 2 (4) (1982) 237–250.

[26] S. Altmeyer, R. Douma, W. Lunniss, R. I. Davis, On the effectiveness of cache partitioning in hard real-time systems, Real-Time Systems (2015) 1–46.

[27] G. Gebhard, S. Altmeyer, Optimal task placement to improve cache performance, in: Proceedings of the 7th ACM & IEEE international conference on Embedded software, 2007, pp. 259–268.

[28] W. Lunniss, S. Altmeyer, R. I. Davis, Optimising task layout to increase schedulability via reduced cache related pre-emption delays, in: Proceedings of the 20th ACM International Conference on Real-Time and Network Systems, 2012, pp. 161–170.

[29] R. J. Bril, S. Altmeyer, M. M. van den Heuvel, R. I. Davis, M. Behnam, Integrating cache-related pre-emption delays into analysis of fixed priority scheduling with pre-emption thresholds, in: 2014 IEEE Real-Time Systems Symposium, IEEE, 2014, pp. 161–172.

[30] A. Burns, Preemptive priority-based scheduling: An appropriate engineering approach, in: Advances in Real-Time Systems, chapter 10, Prentice Hall, 1994, pp. 225–248.

[31] M. Chéramy, A.-M. Déplanche, P.-E. Hladik, Simulation of Real-Time Multiprocessor Scheduling with Overheads, in: International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2013), Reykjavik, Iceland, 2013, pp. 5–14.

[32] C. L. Liu, J. W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, Journal of the ACM (JACM) 20 (1) (1973) 46–61.

[33] R. L. Mattson, J. Gecsei, D. R. Slutz, I. L. Traiger, Evaluation techniques for storage hierarchies, IBM Systems journal 9 (2) (1970) 78–117.

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

[35] N. Audsley, A. Burns, R. Davis, K. Tindell, A. Wellings, Real-time system scheduling, Springer Berlin Heidelberg, 1995.

[36] A. Burns, A. Wellings, Concurrent and Real-Time Programming in Ada, Cambridge University Press, 2007.

[37] H. Theiling, C. Ferdinand, R. Wilhelm, Fast and precise WCET prediction by separated cache and path analyses, Real-Time Systems 18 (2-3) (2000) 157–179.

[38] J. Staschulat, S. Schliecker, R. Ernst, Scheduling analysis of real-time systems with precise modeling of cache related preemption delay, in: Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS), 2005, pp. 41–48.

[39] W. R. E. Lunniss, Cache related pre-emption delays in embedded real-time systems, Ph.D. thesis, University of York (2014).

[40] N. C. Audsley, Optimal priority assignment and feasibility of static priority tasks with arbitrary start times, in: Technical Report YCS 164, Dept. Computer Science, University of York, UK, 1991.

[41] E. Bini, G. C. Buttazzo, Measuring the performance of schedulability tests, Real-Time Systems 30 (1-2) (2005) 129–154.

[42] J. Gustafsson, A. Betts, A. Ermedahl, B. Lisper, The mälardalen WCET benchmarks: Past, present and future, in: 10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

[43] C. Fotsing, F. Singhoff, A. Plantec, V. Gaudel, S. Rubini, S. Li, H. N. Tran, L. Lemarchand, P. Dissaux, J. Legrand, Cheddar architecture description language, Lab-STICC technical report.