CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Conference Paper

Response time analysis of hard real-time tasks sharing software transactional memory data under fully partitioned scheduling

António Barros
Patrick Meumeu Yomsi
Luis Miguel Pinho

CISTER-TR-160408

2016/05/23

Response time analysis of hard real-time tasks sharing software transactional memory data under fully partitioned scheduling

António Barros, Patrick Meumeu Yomsi, Luis Miguel Pinho

CISTER Research Center

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: amb@isep.ipp.pt, pamyo@isep.ipp.pt, lmp@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

Software transactional memory (STM) is a synchronisation paradigm which improves the parallelism and composability of modern applications executing on a multi-corearchitecture. However, to abort and retry a transaction multipletimes may have a negative impact on the temporal characteristicsof a real-time task set. This paper addresses this issue: It providesa framework in which an upper-bound on the worst-case responsetime of each task is derived, assuming that tasks are scheduledby following either the Non-Preemptive During Attempt (NPDA),Non-Preemptive Until Commit (NPUC) or Stack Resource Policyfor Transactional Memory (SRPTM) policy.

Response time analysis of hard real-time tasks sharing software transactional memory data under fully partitioned scheduling

António Barros, Patrick Meumeu Yomsi and Luís Miguel Pinho
CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto
Email: {amb; pamyo; lmp}@isep.ipp.pt

***Abstract*—Software transactional memory (STM) is a synchronisation paradigm which improves the parallelism and composability of modern applications executing on a multi-core architecture. However, to abort and retry a transaction multiple times may have a negative impact on the temporal characteristics of a real-time task set. This paper addresses this issue: It provides a framework in which an upper-bound on the worst-case response time of each task is derived, assuming that tasks are scheduled by following either the Non-Preemptive During Attempt (NPDA), Non-Preemptive Until Commit (NPUC) or Stack Resource Policy for Transactional Memory (SRPTM) policy.**

I. Introduction

Parallelism and composability are two keystone features in the development of hard real-time applications on multi-core architectures. Lock-based synchronisation policies with bounded blocking times are well understood in the context of single cores. However, they still represent a major hurdle for multi-cores. Recent developed approaches use either coarse-grained locking [1], [2] or fine-grained locking [3], [4]. Coarse-grained locking has a negative impact on parallelism whereas fine-grained locking has a negative impact on composability. Furthermore, locks are generally detached from the resources they control. This implies that tasks must explicitly cooperate to acquire the proper lock before they can operate the shared resource. These issues are exacerbated with an increase of the number of cores and shared resources. STM provides an alternative where critical sections (a.k.a. *transactions*) are defined to operate on shared memory data without explicitly requesting a lock. This improves composability. On the other front, multiple transactions can execute speculatively in parallel and conflicts can be solved on the fly. This improves parallelism. STM already offers three key benefits for non-real-time systems: (1) it scales well with an increase of the number of cores [5]; (2) it delivers a higher throughput than coarse-grained locks [5] and (3) it does not increase the design complexity as fine-grained locks [6].

Most STM implementations maintain the consistency of shared data by allowing one transaction involved in a conflict to commit while its contenders are aborted and, consequently, they have to retry. This increases the execution time of the jobs and represents a serious concern for real-time systems, especially when all the task deadlines must be met. The time overhead due to the abort-and-retry occurrences must be managed such that the worst-case response time (WCRT) of each task is bounded. To the best of our knowledge, very few research works addressed this issue in the literature.

Contribution. This paper proposes a framework wherein the WCRT analysis of a set of hard real-time tasks, sharing data through STM, is devised. To do so, we assume that conflicts are solved by committing the transactions in a chronological order of their start times as defined by the FIFO contention manager for real-time systems (FIFO-CRT) [7]. Here, the unpredictability issues are solved by following two non-preemptive scheduling strategies –namely, *Non-Preemptive During Attempt* (NPDA) and *Non-Preemptive Until Commit* (NPUC)–, and one fully-preemptive scheduling strategy –namely, *Stack Resource Policy for Transactional Memory* (SRPTM). The analysis for these three strategies were missing in the literature and this paper fills the gap.

Paper organization. The rest of this paper is structured as follows. Section II provides a selection of related work. Section III sets the system model and the assumptions adopted in this work. We summarize the specification of FIFO-CRT, NPDA, NPUC and SRPTM in Section IV. Section V demonstrates the intractability of an NPDA analysis. The WCRT analyses for NPUC and SRPTM are discussed in Section VI and Section VII, respectively. Section VIII reports on the evaluation results. Finally, Section IX concludes the paper and provides future directions.

II. Related work

Previous works on synchronisation for real-time systems addressed the execution of non-blocking critical sections. Anderson *et al.* [8] established scheduling conditions for lock-free transactions under Earliest Deadline First (EDF) and Deadline Monotonic (DM) for uniprocessor systems. For multiprocessor systems, they devised a wait-free mechanism, which guarantees an upper bound on the response time of each transaction [9]. In this latter work, an arriving transaction must help pending transactions to commit before it can proceed.

Manson *et al.* [10] proposed a data access mechanism for uniprocessor platforms – the *Preemptible Atomic Regions* (PAR) – together with its analysis to bound the response time of each job. Here, every preempted atomic region (critical section) is immediately aborted and retried later. This policy behaves as the *Abort-and-Restart* (AR) mechanism [11]. However, it is not adapted for multi-core platforms since its correctness depends on the exclusive execution of the tasks.

Fahmy *et al.* [12] provided a WCRT analysis for real-time tasks scheduled by following the Pfair algorithm [13] for multiprocessors in which STM is used as the synchronisation

mechanism. However they assume that each critical section is limited to at most two quanta, whereas it is not uncommon that this value is exceeded for real-world applications.

Sarni *et al.* [14] proposed transaction deadlines and suggested a contention manager that orders conflicting transactions according to their deadlines. In this work, the experimental results showed an improvement of the number of jobs that meet their deadlines, which is beneficial in terms of schedulability. Unfortunately, the results apply only to soft real-time systems.

Barros *et al.* [7] proposed a FIFO-based approach to serialise concurrent transactions as a way to predict the time required for a transaction to commit. In this work, they addressed the negative impact induced by preemptions on the response time of each transaction and proposed two scheduling algorithms based on EDF in which preemptions are disabled during the execution of the transactions. Later, they also proposed a fully preemptive approach for the same problem [15].

Cotard [16] developed a wait-free STM for hard real-time systems on multi-cores, in which transactions help their contenders to commit. However, this study is limited to *homogeneous* transactions[1]. The system model does not consider any specific scheduling algorithm, but transactions must not be preempted until they commit. Furthermore, update transactions with intersecting write sets must be allocated to the same core, thus making write-write conflicts impossible. This approach ensures that (i) a write-set transaction never aborts and (ii) a read-set transaction aborts at most once.

El-Shambakey *et al.* [17] proposed two contention managers that solve conflicts based on the priorities of the associated schedulers (global-RM and global-EDF). In [18], he extends these contention managers so that a transaction that has executed for a pre-defined amount of time is not aborted. However, the proposed analyses assume that each transaction can operate only one object, unfortunately. In [19], he proposed a contention management policy in which the subset of oldest non-conflicting transactions in progress are executed non-preemptively, while the recent conflicting transactions with this subset are scheduled with the lowest system priority. However, all information regarding the set of transactions in progress as well as their data sets must be known beforehand. This may not scale with an increase of the number of cores. In an approach in which all these information are not required (e.g., see [20]), each transaction must be assigned an arbitrary maximum number of aborts. Here, each conflict is solved based on the priorities of the jobs, but once a transaction exceeds its number of aborts threshold it becomes non-preemptable[2].

III. System model

Task specification. We assume that the workload is carried by a set of n sporadic tasks $\tau \stackrel{\text{def}}{=} \{\tau_1, \ldots, \tau_n\}$. Each task τ_i releases a potentially infinite number of jobs and is characterised by a worst-case execution time C_i, a relative deadline D_i, and a minimum inter-arrival time T_i. These parameters are given with the following interpretation: the j^{th} job of task τ_i executing on processor π_k, referred to as $\tau_{i,j}^k$, is characterised by its release time $r_{i,j}$ such that $r_{i,j+1} \geq r_{i,j}+T_i$ (with $i \in \{1, \ldots, n\}$ and $j \geq 1$); and an absolute deadline $d_{i,j} \stackrel{\text{def}}{=} r_{i,j} + D_i$. Furthermore, each task τ_i may perform a finite, potentially empty set of ζ_i transactions $\{\omega_i^1, \ldots, \omega_i^{\zeta_i}\}$ where ω_i^x is characterised by:

- $C_{\omega_i^x}$: the maximum time required to execute the sequential code of ω_i^x once in isolation, i.e., without any external interference from other tasks or the system itself, and try to commit.
- **The data set** ($DS_{\omega_i^x}$): the collection of shared objects that are accessed by ω_i^x. This data set can be partitioned into two subsets — the read set ($RS_{\omega_i^x}$) and the write set ($WS_{\omega_i^x}$) — where:
  - ▷ $RS_{\omega_i^x}$ is the subset of objects that are accessed by ω_i^x solely for reading.
  - ▷ $WS_{\omega_i^x}$ is the subset of objects that are modified by ω_i^x during its execution.

For the sake of readability, we assume in the rest of this section that each task τ_i carries a single transaction, say ω_i. This assumption is relaxed later in Section VI and Section VII.

Platform and Scheduler specifications. We assume that all the jobs are executed on a multi-core platform $\pi \stackrel{\text{def}}{=} \{\pi_1, \ldots, \pi_m\}$ composed of m homogeneous cores, i.e., all cores have the same computing capabilities and are interchangeable. We refer to function σ to express that a core is allocated to a task: if τ_i is assigned to π_k, then $\sigma(\tau_i) = \pi_k$. The task set τ is scheduled by following a *partitioned EDF* (P-EDF) scheduler, i.e., each task is assigned to a specific core at design time and each core schedules its subset of tasks at runtime by following the classical EDF scheduler[3].

STM specification. We assume that a collection of p STM objects $O \stackrel{\text{def}}{=} \{o_1, \ldots, o_p\}$ are located at the globally shared memory and are accessible to all tasks carrying a transaction, irrespective of the core on which they execute. We assume that the STM allows multiple simultaneous transactions in progress, and keeps record of the chronological order of their start times. Conflicts are detected and solved at commit time by a *contention manager* (CM) that selects the transaction with earliest start time stamp in the conflict to commit first. We assume that transactions cannot abort except when a conflict is detected and the CM manager decides so. A conflict occurs when two or more transactions, executing in parallel (i.e. simultaneously, in different cores) have intersecting data sets and at least one is trying to modify a commonly accessed object. We represent contentions by using a graph G in which *vertices* represent transactions and *edges* connect pairs of transactions assigned to different cores such that the write set of one transaction intersects the data set of the other. Before going any further in this paper, we introduce few concepts that are central for a good understanding of our proposed analyses.

Definition 1 (Contention group)**.** Given a contention graph G, a *contention group*, denoted as Ω_g (with $g \geq 1$), is defined as

[1] In this case, the data set of a transaction consists exclusively of the read set (*read-set transaction*) or the write set (*write-set transaction*).

[2] Conflicts between non-preemptable transactions are solved based on the time stamps of the instants at which transactions exceeded their thresholds.

[3] Here, at each time instant the job with the earliest absolute deadline is selected for execution and ties are broken in an arbitrary manner.

a set of connected vertices of G in which any two transactions are connected by a path.

Figure 1 illustrates a very simple example of contention groups in which a set of five transactions $\{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5\}$ are sharing a set of three STM objects $\{o_1, o_2, o_3\}$. In this example[4], the data sets of the transactions form two distinct contention groups: $\Omega_1 = \{\omega_1, \omega_5\}$ and $\Omega_2 = \{\omega_2, \omega_3, \omega_4\}$.

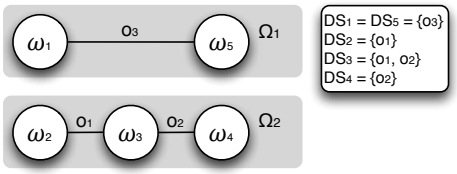


Fig. 1. Illustration of contention groups.

Definition 2 (Cores assigned to a contention group Π_g)**.** Given a contention group Ω_g, we define Π_g as the set of $m_g \leq m$ cores assigned to the execution of transactions in Ω_g. Formally, $\Pi_g = \{\pi_k \mid \sigma(\omega_i) = \pi_k,\ \omega_i \in \Omega_g\}$.

Definition 3 (Direct contender of a transaction)**.** Given a contention graph G, a *direct contender of transaction* ω_i is defined as a transaction ω_j that shares at least one STM object with ω_i and this object is modified either by ω_i or ω_j or both. Formally, that is $(WS_i \cap DS_j) \cup (DS_i \cap WS_j) \neq \emptyset$.

Definition 4 (Indirect contender of a transaction)**.** Given a contention graph G, an *indirect contender of transaction* ω_i is defined as a transaction ω_j that does not share any STM object with ω_i, but belongs to the same contention group.

Definition 5 (Independent transactions)**.** Given a contention graph G, two transactions ω_i and ω_j are said to be *independent* when they belong to two distinct contention groups.

Each instance of a transaction has a life cycle that follows the states represented in Figure 2, in which the transaction is *in progress*. When a transaction starts, it executes the transaction code and commits if no conflicts are detected, otherwise it may be aborted, retrying to commit immediately. Note: a transaction may be aborted multiple times until it successfully commits.

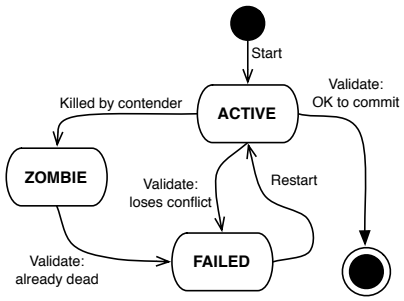


Fig. 2. Illustration of the state diagram of a transaction.

Definition 6 (Transaction overhead of a job)**.** Given a task τ_i hosting a transaction ω_i, the *transaction overhead of job* $\tau_{i,j}$, denoted as $W_{i,j}$, is defined as the time wasted in executing aborted commit attempts of ω_i. Formally, $W_{i,j}$ is given by:

$$W_{i,j} \stackrel{\text{def}}{=} A_{i,j} \cdot C_{\omega_i} \tag{1}$$

where $A_{i,j}$ represents the number of failed attempts of ω_i before it commits.

Definition 7 (Execution time of a job executing a transaction)**.** The *execution time of the* j^{th} *job* of task τ_i executing a transaction ω_i, denoted as $C_{i,j}$, is defined as the sum of four terms: (1) the time $C_{\mathrm{a}-\omega_i}$ required to execute the code of τ_i *before* ω_i starts, (2) the time $C_{\mathrm{p}-\omega_i}$ required to execute the code of τ_i *after* ω_i has committed, (3) the time C_{ω_i} required for a successful attempt of ω_i, and (4) the time $W_{i,j}$ associated to the transaction overhead. Formally:

$$C_{i,j} \stackrel{\text{def}}{=} C_{\mathrm{a}-\omega_i} + C_{\omega_i} + C_{\mathrm{p}-\omega_i} + W_{i,j} \tag{2}$$

IV. Scheduling approaches

Traditional alternatives for the analysis of real-time tasks are based on (1) the system utilisation, (2) the system load or (3) the response time of each individual task. Out of these techniques, we adopted the response time analysis as it collates closely with the behaviour of the system, all the way down to the task level. This property is useful, especially when tasks may not be independent and/or may carry transactions. Before we provide the technical details of our WCRT analyses, this section recalls the main intuition and rules behind each adopted scheduling approach.

FIFO-CRT. This CM solves conflicts by ensuring that for transactions in the commit phase, each transaction update does not invalidate concurrent transactions with an earlier start time, otherwise the transaction is aborted. In order to avoid deadlocks: any preempted transaction can be aborted, irrespective of its start time stamp.

NPDA. This approach schedules jobs by following the classical P-EDF scheduler, but the transactions are non preemptable during their execution. When a transaction fails to commit, preemptions are temporarily enabled and the ready jobs with earlier deadlines are scheduled. As a consequence, NPDA allows for multiple transactions to be simultaneously in progress (although some may be preempted) on each core.

NPUC. This approach schedules jobs by following P-EDF, but transactions are executed without any preemption points until they commit. This scheduling technique ensures that at most one transaction may be in progress on each core at runtime.

SRPTM. This approach schedules jobs by following P-EDF, but a few extra rules are added when a transaction is in progress. First, based on SRP [21], tasks τ_i and τ_j are assigned preemption levels, say λ_i and λ_j, such that $\lambda_i > \lambda_j$ iff $D_i < D_j$. Then, each transaction ω_i is assigned a preemption level λ_{ω_i} that is the highest preemption level among all tasks that belong to the same contention group Ω_g (see Definition 1) as ω_i. Formally $\lambda_{\omega_i} = \max(\lambda_j \mid \{\omega_i, \omega_j\} \subset \Omega_g)$. On each core π_k, the local scheduler maintains a *core ceiling* parameter Λ_k. This parameter represents a non-negative variable that holds the highest preemption level of any task whose progress depends on the transaction currently in progress on that core. When no transaction is in progress, the core ceiling is null

[4]Figures 1 and 2 are borrowed from [15].

and all jobs are scheduled by following the EDF scheduler. Otherwise, SRPTM does not schedule the job with the earliest deadline only in the following two situations:

1) *The arriving job has a preemption level not greater than the current core ceiling*. This rule allows us to avoid situations where jobs, which are waiting on other cores for the transaction to commit, are improperly delayed.
2) *The arriving job has a preemption level greater than the current core ceiling, but carries a transaction*. In this case, the core ceiling is raised to the preemption level of this arriving job, and the job with the transaction already in progress is scheduled so that the transaction commits as soon as possible. Upon the commit, the core ceiling is reset to null and the scheduler behaves according to the EDF scheduler. This rule ensures that the system does not have more than m transactions in progress.

At this point, we have everything we need to compute a sound and convincing upper-bound on the WCRT of a set of hard real-time tasks, sharing STM data. Before going any further, we will discuss the intractability of a NPDA analysis.

V. On the intractability of a NPDA analysis

NPDA does not restrict the number of transactions that can simultaneously be in progress on each core, thus increasing the complexity of determining a tight upper-bound on the number of times each transaction may be aborted. The worst-case scenario in this context occurs when each transaction has to wait for all of its direct contenders before it can commit. To support this claim, Figure 3 illustrates a contention group that involves three cores $\{\pi_0, \pi_1, \pi_2\}$ and six tasks $\{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6\}$. Here, we assume that task τ_i carries transaction ω_i and that transactions $\omega_2, \omega_3, \omega_4$ are direct contenders of ω_1. Additionally, we assume that in the illustrated time segment[5], the chronological order in which transactions start is $\omega_2 \prec \omega_4 \prec \omega_6 \prec \omega_3 \prec \omega_1$, and that τ_2 and τ_4 are preempted when we start observing the system. In this figure, thin vertical lines indicate the time instants at which a transaction is fated to fail, either (1) by being invalidated when a contender commits, or (2) by the CM decision at the commit time. The transaction that aborts and the one that causes the abort are displayed on the extremes of each thin line. Transaction ω_1 executes on core π_0, and may abort in favor of transactions that started earlier on different cores. The sequence of events in this special case shows that ω_1 commits only when all its direct contenders (i.e. ω_2, ω_3 and ω_4) have committed. Note that this delay includes waiting for ω_6 (an indirect contender) to commit.

Although it is straightforward to identify the direct contenders of a transaction, the exercise becomes very challenging when it comes to compute the delay they impose on the commit of that transaction. As a matter of fact, when (1) the number of transactions (i.e., the vertices), (2) the number of dependencies (i.e., edges) and (3) the number of assigned cores in a contention group grow, the search-space where to find the sequence of transactions that leads to the longest commit delay also grows exponentially. Specifically, the computational complexity of completing this operation is in order of $\mathcal{O}(n^n \times m)$, where n is the total number of tasks and m is the total number of cores. Therefore, a tight feasibility analysis for NPDA is computationally intractable.

[5]Note that τ_5 is not released in the illustrated time segment.

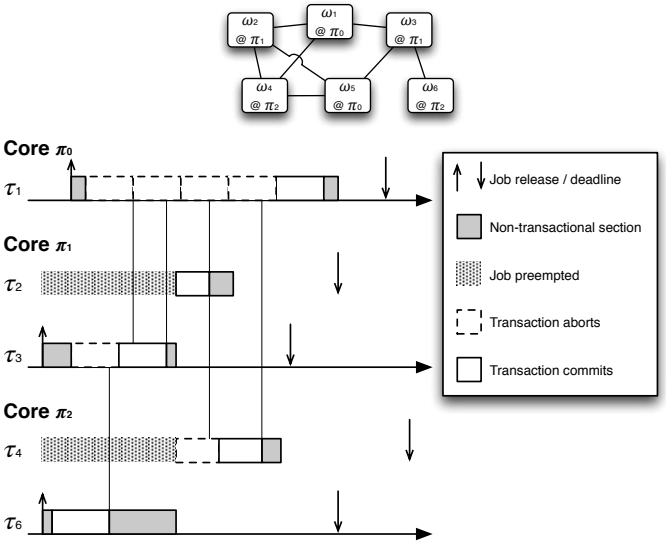


Fig. 3. Preempted jobs executing ω_2 and ω_4 are rescheduled in times to abort transaction ω_1.

VI. WCRT analysis for NPUC

We recall that NPUC schedules transactions in a non-preemptive manner until they commit (see Section IV). This property ensures two essential predicates: (i) at most one transaction can be in progress on each core at any time instant, and (ii) the delay experienced by a transaction prior to its commit depends exclusively on its set of direct contenders, which in turn, depend on their own set of direct contenders. We can compute an upper-bound on the WCRT of a transaction by determining the sequence of transactions that will produce the longest delay, which occurs when the transaction under analysis arrives at a time instant when the pending workload associated to its contenders is maximum. For the sake of clarity, we assume that the transaction under analysis arrives upon the start time of all its contenders.

Main idea. We consider that task τ_i carries a transaction ω_i and is the task under analysis. To compute a sound and tight upper-bound on the WCRT of τ_i, we should compute: (1) the WCRT of the transaction ω_i; (2) the WCRT of the section a-ω_i before the transaction; (3) the WCRT of the section p-ω_i after the transaction; and finally (4) we sum up all these values[6].

A. WCRT of transaction ω_i

This section presents two methods for the computation of an upper bound on the commit delay of ω_i. The first method provides a tight bound, but requires that all possible sequences of transactions that produce a delay on ω_i are known, whereas

[6]As the WCRT of τ_i is upper-bounded by the sum of the response times of the non-transactional sections and the transactional sections, we consider only one transaction per task.

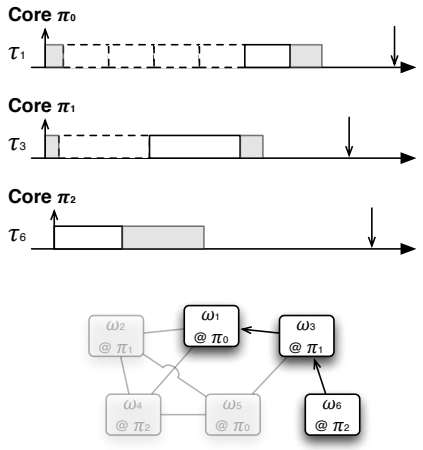


Fig. 4. Sequence of transactions until ω_1 commits.

the second method provides a pessimistic upper bound, but with a linear-time complexity. The selection of one method over the other will depend on the user needs and the available computational resource and time.

▷ *Method 1: Tight WCRT of transaction* ω_i. This method is based on the contention graph (see Definition 1) to determine all the possible transaction sequences that may delay ω_i. From this process, the sequence that produces the longest delay on ω_i is found by considering the *simple paths*[7] converging to the vertex ω_i with no repetition of cores. This is possible as the contention graph is finite. For each transaction sequence, an upper-bound on the WCRT of ω_i is computed by upper-bounding the times to commit of all the transactions prior to ω_i. Figure 4 depicts an example in which three transactions — ω_6, ω_3 and ω_1 — start almost simultaneously on cores π_2, π_1 and π_0, respectively. Assuming that the transactions arrive in this order, the WCRT of ω_1 depends on that of ω_3, which in turn depends on the WCRT of ω_6. Note that for each sequence of transactions: (1) *each transaction performs at most two attempts before success once it becomes eligible to commit*; (2) *the very first transaction may abort after the start of* ω_i. Figure 5 illustrates a case in which such a phenomenon occurs. Upon the commit of the transaction out of the sequence, ω_1 is fated to abort; however ω_3 has started prior to this attempt of ω_1 to commit. Point (1) and Point (2) allow us to formulate an upper bound on the WCRT of the transaction in a given sequence. To this end, we consider a sequence of $k > 0$ transactions and υ as the function that returns the transaction at each position in this sequence. We have $\upsilon(k) = \omega_i$.

Lemma 1. *The WCRT of* ω_i *in a given sequence, denoted as* $R^{(k)}$*, is computed recursively by Equation 3.*

$$\begin{cases} R^{(1)} = 2 \cdot C_{\upsilon(1)} \\ R^{(q)} = \left(\left\lceil \frac{R^{(q-1)}}{C_{\upsilon(q)}} \right\rceil + 1 \right) \cdot C_{\upsilon(q)} \quad \text{if } 1 < q \leq k. \end{cases} \tag{3}$$

Proof: The proof follows directly from Point (1) and Point (2). In the worst-case scenario, the first transaction in the sequence takes at most two attempts to commit (see Point (2)). Then, transaction at position $q \geq 2$ has to wait for the transaction at position $q-1$ to commit plus at most one additional attempt to successfully commit (see Point (1)). ■

[7]A simple path is a sequence of connected vertices with no repetition.

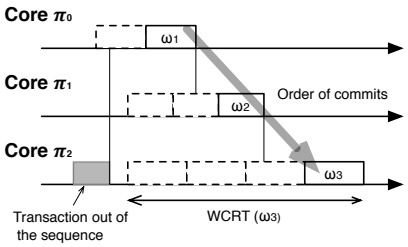


Fig. 5. ω_1, the first transaction in the sequence aborts once before commits.

Now, let $\mathcal{S}_i$ denote the set of all simple paths that converge towards ω_i and let $\mathcal{S}_{i,k}$ denote the subset of $\mathcal{S}_i$ which consists only of simple paths of length k. We assume that ω_i belongs to contention group Ω_g.

Lemma 2. *The WCRT of* ω_i *is given by:*

$$R_{\omega_i} = \max_{k=1}^{m_g} \left\{ \max_{\mathcal{S}_{i,k}} \left(R^{(k)} \right) \right\} \tag{4}$$

Proof: The length of the simple path to transaction ω_i is at most m_g as ω_i belongs to Ω_g. For any sequence of transactions in Ω_g, an upper-bound on the WCRT of the transaction at position q is given by Equation 3. From these two predicates, it follows that the WCRT of ω_i cannot exceed the RHS of Equation 4. Now, as Ω_g is finite by assumption, this RHS is also an upper-bound on the WCRT of ω_i. ■

Although this method is tight, it suffers from the fact that all the possible simple paths must be considered. This might become impractical with an increase of the number of transactions and cores in the system.

▷ *Method 2: Linear-time WCRT of transaction* ω_i. The main idea behind this method is to avoid the usage of any specific sequence of transactions. To this end, we consider the transaction with the longest execution time on each core, which belongs to the same contention group as ω_i. Formally, first we determine the longest execution time C_{Ω_g,π_ℓ} of all the transactions in Ω_g that are assigned to each core $\pi_\ell \in \Pi_g$ by using Equation 5.

$$C_{\Omega_g,\pi_\ell} = \max \left\{ C_{\omega_j} \mid \omega_j \in \Omega_g \ \wedge \ \sigma(\tau_j) = \pi_\ell \right\} \tag{5}$$

Then, from Point (1) and Point (2) in Method 1, we compute an upper-bound I_{Ω_g,π_k} on the delay that any transaction in Ω_g assigned to core π_k may suffer by using Equation 6.

$$I_{\Omega_g,\pi_k} \stackrel{\text{def}}{=} \sum_{\pi_\ell \in \Pi_g \setminus \pi_k} 2 \cdot C_{\Omega_g,\pi_\ell} \tag{6}$$

Note that I_{Ω_g,π_k} is common to all transactions in Ω_g assigned to π_k, so it is computed only once for each pair of core and contention group.

Lemma 3. *An upper-bound on the WCRT of transaction* ω_i *is given by Equation 7.*

$$R_{\omega_i} = I_{\Omega_g,\pi_k} + 2 \cdot C_{\omega_i} \tag{7}$$

Proof: The WCRT of ω_i is upper-bounded by the delay that ω_i suffers from all concurrent transactions with an earlier

start time (see Equation 6), augmented by the time that ω_i takes to commit once it is possible to do so (see Point (1)). ■

This method is pessimistic as Equations 6 and 7 consider a sequence of transactions that may never occur in practice.

B. WCRT of task τ_i

We recall that the non-transactional sections of task τ_i ($\mathrm{a}{-}\omega_i$ and $\mathrm{p}{-}\omega_i$) are scheduled by following the fully preemptive P-EDF scheduler, whereas the transaction (ω_i) is scheduled with disabled preemption.

1) WCRT of $\mathrm{a}{-}\omega_i$: We compute an upper-bound on the WCRT of $\mathrm{a}{-}\omega_i$ by tuning the technique described by Spuri [22] for the WCRT of a task. In [22], a single core platform is assumed and tasks are scheduled by following the fully-preemptive EDF scheduler. However, the model of computation assumed in this work requires three adaptations.

1st adaptation: WCET of a task with transaction. The Spuri et al. [22], [23] technique is based on the concept of "deadline-d busy period length", which requires the WCET of all tasks executing during that period. As such, the deadline-d busy period exploits an approximation on the WCET values of the tasks carrying a transaction rather than their actual values. Since the execution time of such a task must include the overhead induced by the aborted attempts of the transaction, the WCET of task τ_i is upper-bounded by Equation 8.

$$C_i \stackrel{\text{def}}{=} C_{\mathrm{a}-\omega_i} + R_{\omega_i} + C_{\mathrm{p}-\omega_i} \tag{8}$$

2nd adaptation: Extension of the deadline-d busy period. In order to determine an upper bound on the WCRT of $\mathrm{a}{-}\omega_i$, the deadline-d busy period is relevant until the completion time of this section. To this end, the length of the deadline-d busy period is adapted from [22] as in Equation 9.

$$\begin{cases} L^{(0)}_{\mathrm{a}-\omega_i}(a) & = 0 \\ L^{(q+1)}_{\mathrm{a}-\omega_i}(a) & = \sum\limits_{\tau_j \in \mathcal{D}_i} \min\left\{ \left\lceil \frac{L^{(q)}_i}{T_j} \right\rceil, 1 + \left\lfloor \frac{a+D_i-D_j}{T_j} \right\rfloor \right\} \cdot C_j \\ & \quad + \left\lfloor \frac{a}{T_i} \right\rfloor \cdot C_i + C_{\mathrm{a}-\omega_i} \end{cases} \tag{9}$$

In Equation 9, a denotes the release time of the job under analysis, $\mathcal{D}_i \stackrel{\text{def}}{=} \{\tau_j \mid D_j < a + D_i; j \neq i; \sigma(\tau_i) = \sigma(\tau_j)\}$ and the deadline-d busy period, denoted as $L_{\mathrm{a}-\omega_i}$, is computed by using a recursive algorithm, which stops as soon as $L^{(q+1)}_{\mathrm{a}-\omega_i}(a) = L^{(q)}_{\mathrm{a}-\omega_i}(a)$ for some integer $q \geq 0$ or when $L^{(q+1)}_{\mathrm{a}-\omega_i}(a)$ exceeds $a + D_i$. In this latter case, the system is not schedulable. Note that the longest busy period occurs when the last job of τ_i is released at instant a_m such that $a_m = \mathrm{argmax}(L_{\mathrm{a}-\omega_i}(a))$, see [22] for details.

3rd adaptation: lower priority blocking. If a job is released when a lower priority job is executing a transaction, then the newly released job will be blocked until the transaction commits. Upon the commit, preemption is enabled again. In this case, the WCRT of $\mathrm{a}{-}\omega_i$ must consider a possible blocking occurring at the time instant at which the job is released. In the worst-case, this blocking time corresponds to the longest response time of a transaction carried by a job with a lower priority and assigned to the same core as τ_i, as formalised in Equation 10.

$$B_i \stackrel{\text{def}}{=} \max\left\{ R_{\omega_j} \mid D_i < D_j \ \wedge \ \sigma(\tau_i) = \sigma(\tau_j) \right\} \tag{10}$$

It follows that an upper-bound on the WCRT of $\mathrm{a}{-}\omega_i$, denoted as $R_{\mathrm{a}-\omega_i}$, is given by Equation 11.

$$R_{\mathrm{a}-\omega_i} \stackrel{\text{def}}{=} B_i + \max\left\{ C_{\mathrm{a}-\omega_i}, L_{\mathrm{a}-\omega_i} - a_m \right\} \tag{11}$$

Intuition behind Equation 11: An upper-bound on the WCRT of $\mathrm{a}{-}\omega_i$ is defined by the longest delay on the last job that completes in the busy-period.

2) WCRT of $\mathrm{p}{-}\omega_i$: When a transaction is in progress, concurrent jobs with smaller deadlines that arrive are prevented from executing as preemption is disabled. Upon its completion, preemption is enabled again. As such, $\mathrm{p}{-}\omega_i$ may suffer interference from concurrent jobs with earlier deadlines: (i) that were released while ω_i was in progress; and (ii) that are released prior to the job completion. Hence, a conservative WCRT of $\mathrm{p}{-}\omega_i$ is obtained by maximizing the interference it may suffer. To this end, we assume that ω_i starts at the earliest possible time instant of its carrying job. This scenario allows us to accommodate the maximum number of concurrent jobs with an earlier deadline between the time at which the transaction starts and the deadline of the job. The deadline associated to transaction ω_i can thus be defined by Equation 12.

$$D_{\omega_i} \stackrel{\text{def}}{=} D_i - C_{\mathrm{a}-\omega_i} \tag{12}$$

An upper-bound on the WCRT of $\mathrm{p}{-}\omega_i$ can be computed in an iterative manner by using a fixed-point algorithm as presented in Equation 13 in which $\mathcal{D}_{\omega_i} \stackrel{\text{def}}{=} \{\tau_j \mid D_j < D_{\omega_i}; \sigma(\tau_i) = \sigma(\tau_j)\}$.

$$\begin{cases} R^{(0)}_{\mathrm{p}-\omega_i} & = C_{\mathrm{p}-\omega_i} \\ R^{(q+1)}_{\mathrm{p}-\omega_i} & = \sum\limits_{\tau_j \in \mathcal{D}_{\omega_i}} \min\left\{ \left\lceil \frac{R^{(q)}_{\mathrm{p}-\omega_i}}{T_j} \right\rceil, 1 + \left\lfloor \frac{D_{\omega_i}-D_j}{T_j} \right\rfloor \right\} \cdot C_j \\ & \quad + C_{\mathrm{p}-\omega_i} \end{cases} \tag{13}$$

Intuition behind Equation 13: As preemption is disabled when a transaction is in progress, all the released jobs with a higher priority will delay $\mathrm{p}{-}\omega_i$ and all jobs with a higher priority than τ_i will interfere with $\mathrm{p}{-}\omega_i$ upon the commit of ω_i.

3) WCRT of task τ_i: At this stage, we can derive the upper-bound R_i on the WCRT of task τ_i[8].

Theorem 1. *The WCRT of task τ_i is obtained by combining the WCRTs of the three sections that compose τ_i, as defined in Equation 14*

$$R_i = R_{\mathrm{a}-\omega_i} + R_{\omega_i} + R_{\mathrm{p}-\omega_i} \tag{14}$$

Proof: This theorem follows directly from Equation 11 (which bounds the execution of $\mathrm{a}{-}\omega_i$); Equation 4 or Equation 7 (which bounds the execution of ω_i); and finally Equation 13 (which bounds the execution of $\mathrm{p}{-}\omega_i$). ■

[8] An upper-bound on the WCRT of a task τ_i that does not carry a transaction is a special case of Equation 14 where $R_{\omega_i} = R_{\mathrm{p}-\omega_i} = 0$ and $C_{\mathrm{a}-\omega_i} = C_i$.

VII. WCRT ANALYSIS FOR SRPTM

In NPDA and NPUC, preemptions are disabled during the execution of the transaction in order to avoid a situation where a transaction can be aborted by a later released transaction. The approach presented in this section (SRPTM) achieves the same goal, but does not disable preemption. This feature allows us to improve the schedulability and the flexibility of the system. To this end, SRPTM applies the following rules:

1) Each transaction is assigned a preemption level that indicates its urgency (e.g., computed from the relative deadline of the carrying task).
2) Each transaction is protected from being preempted by concurrent jobs that are less urgent.
3) Any job is allowed to preempt the running job during the execution of its transaction only if (i) it does not carry a transaction and (ii) it has a higher preemption level than the transaction in progress.

These rules do not eliminate the interference but, they reduce it during the execution of a transaction. In a similar manner as for NPUC paradigm, deriving an upper-bound on the WCRT of a task under SRPTM consists of computing the WCRT of the individual parts of the task taken separately.

A. WCRT of transaction ω_i

SRPTM shares two fundamental features with NPUC: (1) a transaction suffers the delay associated to its direct contenders with an earlier start time before it can commit; and (2) no more than one transaction can be in progress on each core. From these two features, it remains true that every transaction requires at most two attempts to commit, once it is legally possible to do so. We denote an upper-bound on the WCRT of these last two attempts by $R^*_{\omega_i}$ for transaction ω_i.

Since SRPTM is based on the P-EDF scheduler, the computation of $R^*_{\omega_i}$ depends on the so-called *intra-core* interference only, i.e., the interference associated to the higher priority jobs without transactions and assigned to the same core as τ_i. The preemption level of such a task, say τ_j, is thus greater than the core ceiling (i.e., $\lambda_j > \Lambda_k \geq \lambda_{\omega_i}$).

Lemma 4. *An upper-bound $R^*_{\omega_i}$ on the WCRT of the last two attempts of ω_i is given by Equation 15 in which $\mathcal{D}^*_{\omega_i} \stackrel{\text{def}}{=} \{\tau_j \mid D_j < D_{\omega_i}; \sigma(\tau_i) = \sigma(\tau_j); \lambda_j > \lambda_{\omega_i}; \lambda_{\omega_j} = 0\}$.*

$$\begin{cases} R^{*(0)}_{\omega_i} & = 2 \cdot C_{\omega_i} \\ R^{*(q+1)}_{\omega_i} & = \sum\limits_{\tau_j \in \mathcal{D}^*_{\omega_i}} \min\left\{ \left\lceil \frac{R^{*(q)}_{\omega_i}}{T_j} \right\rceil, 1 + \left\lfloor \frac{D_{\omega_i} - D_j}{T_j} \right\rfloor \right\} \cdot C_j \\ & \quad +2 \cdot C_{\omega_i} \end{cases} \tag{15}$$

The calculation of $R^*_{\omega_i}$ stops as soon as the result converges, i.e., $R^{*(q+1)}_{\omega_i} = R^{*(q)}_{\omega_i}$ for some positive integer q or when $R^*_{\omega_i}$ exceeds D_{ω_i}.

Proof: $R^*_{\omega_i}$ is determined by maximising the possible intra-core interference. As such, it is given by the execution time taken by the two attempts of the transaction, augmented by the execution time required by the concurrent jobs that are able to preempt τ_i during the time window corresponding to these two attempts. ■

Let us assume that τ_i is assigned to core π_k and ω_i belongs to contention group Ω_g. Then, we have everything necessary to compute a tight upper bound on the WCRT of ω_i. For each core π_ℓ in Π_g, but π_k, the transaction that presents the longest response time to execute two attempts is selected and sum-up to produce the worst-case delay on ω_i. As such, the *inter-core interference*, denoted as I_{Ω_g,π_k}, can be upper-bounded and formalized as in Equation 16.

$$I_{\Omega_g,\pi_k} = \sum_{\pi_\ell \in \Pi_g \setminus \pi_k} \max\left\{ R^*_{\omega_j} \mid \omega_j \in \Omega_g \ \wedge \ \sigma(\tau_j) = \pi_\ell \right\} \tag{16}$$

Once an upper-bound on the intra-core interference and an upper-bound on the inter-core interference are computed, an upper-bound on the WCRT of ω_i can be determined by combining these two expressions as follows.

$$R_{\omega_i} \stackrel{\text{def}}{=} I_{\Omega_g,\pi_k} + R^*_{\omega_i} \tag{17}$$

B. WCRT of task τ_i

SRPTM allows a newly released job to be blocked, irrespective of its priority, when a transaction is in progress. The blocking and interference factors differ depending on whether the tasks carry transactions or not. Thus, we compute an upper-bound on WCRT of τ_i in two distinct approaches.

1) WCRT of τ_i with a transaction: We consider the three sections of τ_i separately.

Blocking term. SRPTM does not allow more than one transaction in progress per core. When a transaction is in progress, the newly released job is *directly blocked* until the transaction commits. Upon the commit, preemption is enabled again and ready jobs are scheduled by following a classical EDF scheduler. This implies that no job can incur an indirect blocking. Any job with a transaction can be directly blocked at most once. Hence, the maximum blocking time, denoted as DB_i, is defined by longest response time of a transaction from all the transactions assigned to the same core. This is derived from the subset of tasks with a lower preemption level, as formalised in Equation 18.

$$\mathrm{DB}_i \stackrel{\text{def}}{=} \max\left\{ R_{\omega_j} \mid \lambda_j < \lambda_i \ \wedge \ \sigma(\tau_i) = \sigma(\tau_j) \right\} \tag{18}$$

WCRT of $\mathrm{a}{-}\omega_i$. Under P-EDF, any job can suffer interference from other jobs released on the same core. An upper-bound on the WCRT of $\mathrm{a}{-}\omega_i$ can once again be determined by adapting the technique presented by Spuri [22]. For the purpose of this analysis, the WCET of τ_i is approximated by Equation 19.

$$C_i = C_{\mathrm{a}-\omega_i} + R_{\omega_i} + C_{\mathrm{p}-\omega_i} \tag{19}$$

In the same manner as for NPUC (see Section VI-B), the extension of the deadline-d busy period is determined by Equation 20.

$$\begin{cases} L^{(0)}_{\mathrm{a}-\omega_i}(a) & = 0 \\ L^{(q+1)}_{\mathrm{a}-\omega_i}(a) & = \sum\limits_{\tau_j \in \mathcal{D}_i} \min\left\{ \left\lceil \frac{L^{(q)}_i}{T_j} \right\rceil, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor \right\} \cdot C_j \\ & \quad + \left\lfloor \frac{a}{T_i} \right\rfloor \cdot C_i + C_{\mathrm{a}-\omega_i} \end{cases} \tag{20}$$

The computation of $L_{\mathrm{a}-\omega_i}(a)$ is performed by using a fixed-point algorithm that converges if $L^{(q+1)}_{\mathrm{a}-\omega_i}(a) = L^{(q)}_{\mathrm{a}-\omega_i}(a)$ for some non-negative integer q. Otherwise, if $L^{(q+1)}_{\mathrm{a}-\omega_i}(a) > a + D_i$ for a given q, then the system is "not schedulable". In the former case, the longest deadline-d busy period $L_{\mathrm{a}-\omega_i}$ occurs when the last job of τ_i is released at the time instant a_m such that $a_m = \mathrm{argmax}(L_{\mathrm{a}-\omega_i}(a))$, see [22] for further details.

Finally, an upper-bound on the WCRT of $\mathrm{a}-\omega_i$ is given by an upper-bound on the WCRT of this non-transactional section, augmented by the delay produced by the busy period. The possible blocking time that a job can suffer when it is released is formalized as in Equation 21.

$$R_{\mathrm{a}-\omega_i} = \mathrm{DB}_i + \max\left\{C_{\mathrm{a}-\omega_i}, L_{\mathrm{a}-\omega_i} - a_m\right\} \tag{21}$$

WCRT of $\mathrm{p}-\omega_i$. This section can be preempted by: (1) the jobs with transactions that were released since the transaction is in progress, and (2) the jobs with a higher priority that are released in its window of execution. In order to maximise the interference, we consider the same time window as defined in Equation 12 for the jobs with a transaction. The *"critical time instant"* at which this section of the task can start executing occurs when the blocking time and the interference experienced are at their maximum. The length of the interval which start at a critical instant, denoted as $D_{\mathrm{p}-\omega_i}$, is formally given as in Equation 22.

$$D_{\mathrm{p}-\omega_i} \stackrel{\mathrm{def}}{=} D_i - (R_{\mathrm{a}-\omega_i} + R_{\omega_i}) \tag{22}$$

This interval defines the longest possible response time of $\mathrm{p}-\omega_i$ while assuming the lowest slack of τ_i upon the commit of ω_i. Thus, the WCRT of $\mathrm{p}-\omega_i$ can be defined by the execution demand of this section augmented by the execution demands of the concurrent jobs, with an earlier deadline, that are released inside both mentioned time windows as formalized in Equation 23, where:
$\mathcal{D}^1_{\mathrm{p}-\omega_i} \stackrel{\mathrm{def}}{=} \{\tau_j \mid D_j < D_{\omega_i}; \sigma(\tau_i) = \sigma(\tau_j); \lambda_{\omega_j} \neq 0\}$ and
$\mathcal{D}^2_{\mathrm{p}-\omega_i} \stackrel{\mathrm{def}}{=} \{\tau_j \mid D_j < D_{\mathrm{p}-\omega_i}; \sigma(\tau_i) = \sigma(\tau_j)\}$.

$$\begin{cases} R^{(0)}_{\mathrm{p}-\omega_i} = C_{\mathrm{p}-\omega_i} \\ \\ R^{(q+1)}_{\mathrm{p}-\omega_i} = \displaystyle\sum_{\tau_j \in \mathcal{D}^1_{\mathrm{p}-\omega_i}} \min\left\{\left\lceil \frac{R^{(q)}_{\mathrm{p}-\omega_i}}{T_j} \right\rceil, 1 + \left\lfloor \frac{D_{\omega_i} - D_j}{T_j} \right\rfloor\right\} \cdot C_j \\ \qquad + \displaystyle\sum_{\tau_j \in \mathcal{D}^2_{\mathrm{p}-\omega_i}} \min\left\{\left\lceil \frac{R^{(q)}_{\mathrm{p}-\omega_i}}{T_j} \right\rceil, 1 + \left\lfloor \frac{D_{\mathrm{p}-\omega_i} - D_j}{T_j} \right\rfloor\right\} \cdot C_j \\ \qquad + C_{\mathrm{p}-\omega_i} \end{cases} \tag{23}$$

The iterative computation stops when the result converges to a value such that $R^{(q+1)}_{\mathrm{p}-\omega_i} = R^{(q)}_{\mathrm{p}-\omega_i}$, otherwise if for a given q, we have $R^{(q+1)}_{\mathrm{p}-\omega_i} > D_i$, then the system is "not schedulable".

Theorem 2 (WCRT of a task τ_i with a transaction)**.** *An upper-bound on the WCRT of task τ_i is obtained by combining the direct blocking and the WCRTs of the three sections that compose τ_i, as defined in Equation 24.*

$$R_i = R_{\mathrm{a}-\omega_i} + R_{\omega_i} + R_{\mathrm{p}-\omega_i} \tag{24}$$

Proof: This theorem follows directly from Equation 21, Equation 17 and Equation 23. ■

2) WCRT of τ_i without a transaction: In addition to the classical interference that every task can suffer from the jobs executing on the same core, each task without transaction can experience either direct or indirect blocking.

Direct blocking. It occurs when the job with the earliest deadline is released, but has a preemption level which is not greater than the current core ceiling. We recall that once the transaction commits, preemption is enabled again and a classical EDF scheduler applies. At this moment, the job with the earliest absolute deadline is selected for execution.

Indirect blocking. It occurs when the job with the earliest deadline is released while a transaction is in progress and the job is able to execute because its preemption level is greater than the core ceiling. In the meantime, if another job carrying a transaction is released and fulfills the deadline and preemption level requirements to be scheduled, the core ceiling is raised to the transaction preemption level of this job and this operation forces the transaction already in progress to complete its execution. As such, this process helps the transaction in progress to commit as soon as it is legally possible to do so.

Blocking term. The longest direct blocking term that task τ_i can experience, denoted as DB_i, is given by an upper-bound on the WCRT of all the transactions which: (1) have a higher preemption level than λ_i, (2) belong to the subset of tasks with a lower preemption level than λ_i, and finally (3) are assigned to the same core as τ_i. This is formally expressed in Equation 25.

$$\mathrm{DB}_i = \max\left\{R_{\omega_j} \mid \lambda_{\omega_j} > \lambda_i \ \wedge\ \lambda_j < \lambda_i \ \wedge\ \sigma(\tau_j) = \sigma(\tau_i)\right\} \tag{25}$$

In contrast, an upper-bound on the longest indirect blocking term is given by an upper-bound on the WCRT of all the transactions with a lower preemption level than τ_i. Note that these transactions are assigned to the same core as τ_i. If a job of a task, say τ_g, carrying a transaction is able to preempt τ_i, then task τ_i will be indirectly blocked as formally expressed in Equation 26.

$$\begin{aligned} \mathrm{IB}_i = \max\{R_{\omega_j} \mid \lambda_{\omega_j} < \lambda_i \ \wedge\ \sigma(\tau_j) = \sigma(\tau_i) \\ \wedge\ \exists \tau_g \ :\ \lambda_g > \lambda_i \ \wedge\ \lambda_{\omega_g} > 0\} \end{aligned} \tag{26}$$

Intuition behind Equation 26: When a job of τ_i preempts the job with the current transaction in progress, τ_i may then be preempted by another job τ_g also carrying a transaction, but with an earlier deadline and a preemption level greater than the core ceiling.

Under SRPTM, if a job of τ_i (not carrying a transaction) is released while a transaction is in progress, then τ_i may be blocked at most once. Consequently, direct and indirect blocking terms are mutually exclusive for any job not carrying a transaction. In this case, the longest blocking time is given by the maximum value between direct and indirect blocking, as formally expressed in Equation 27.

$$B_i = \max\left\{\mathrm{DB}_i, \mathrm{IB}_i\right\} \tag{27}$$

WCRT of task τ_i without a transaction. Assuming a task that does not carry a transaction, its jobs are scheduled by

following the EDF scheduler. As such, until each released job completes its execution, it may suffer interference from any released job with an earlier absolute deadline. The computation of an upper-bound on the WCRT of task τ_i in this case can thus be achieved by the method described by Spuri [22], [23] without any adaptations. The iterative equation is replicated in Equation 28, where $\mathcal{D}_i$ is defined as in Equation 9.

$$\begin{cases} L_i^{(0)}(a) = 0 \\ L_i^{(q+1)}(a) = \sum\limits_{\tau_j \in \mathcal{D}_i} \min\left\{ \left\lceil \frac{L_i^{(q)}}{T_j} \right\rceil, 1 + \left\lfloor \frac{a+D_i-D_j}{T_j} \right\rfloor \right\} \cdot C_j \\ \qquad\qquad + \left\lfloor 1 + \frac{a}{T_i} \right\rfloor \cdot C_i \end{cases} \tag{28}$$

The iterative computation stops when the result converges to a value such that $L_i^{(q+1)}(a) = L_i^{(q)}(a)$, otherwise if the deadline is exceeded, the system is not schedulable. Assuming that a job of τ_i is released at time instant $a_m = \text{argmax}(L_i(a))$, the length of the deadline-d busy period that produces the longest delay is denoted as L_i, and the upper-bound on the WCRT of τ_i if no blocking could occur is given by $L_i - a_m$. This result can then be combined with the longest blocking time B_i (see Equation 27) to determine an upper-bound on the overall WCRT of a task without a transaction.

Theorem 3. *An upper-bound on the overall WCRT of a task τ_i without transaction is formalized as in Equation 29.*

$$R_i = B_i + \max\{C_i, L_i - a_m\} \tag{29}$$

Proof: This theorem follows directly from the combination of Equation 27 and Equation 28. ■

VIII. Evaluation results

This section reports on the runtime performance of the three presented scheduling strategies. We used the same platform setup as the one described in [15] and run each simulation for two hyperperiod[9]. In summary, we showed that: (1) the number of task sets schedulable by SRPTM is higher than those schedulable by NPDA and NPUC (see Figure 6a); (2) SRPTM shows the lowest amount of transaction time overhead (see Figure 6b). In this paper, we go one step further in order to validate our proposed analyses. We determine the ratio between the analytic upper bounds as computed in this paper and the corresponding maximum observed transaction response times. Then, we compared the results for systems with 2, 4, 8 and 16 cores, respectively.

Task set generation. Groups of task sets in which 75% of the tasks execute one transaction (with 50% probability to be read-only) are generated. Each task set requires 75% of the platform capacity without overhead and the task-to-core mapping strategy is performed by following the worst-fit heuristic[10]. For a given number of cores, all groups share the same task generation parameters excepts for the C_{ω_i}/C_i ratio, which follows a normal distribution with a fixed (per group) $\mu \in \{20\%, \ldots, 80\%\}$ and $\sigma = 10\%$. We generated 7 groups of 500 task sets assuming 2, 4, 8 and 16 cores, respectively. All the parameters used for the task set generation are provided for a proof of concept of the proposed approach.

[9]The hyperperiod is the least-common-multiple of all the task periods.

[10]This choice is made for benchmark reasons, see [15].

Results for NPUC. The "tight method" appears to compute the maximum observed response times for a subset of transactions, thus showing the best ratio (i.e., 1) for the task sets in all platform settings. For the remaining transactions, we observed that the average (resp. the "worst") ratios range from 7.8 (resp. 245) up to 25.5 (resp. 879). In average, an increase of the number of cores adds more pessimism to the analysis. This can be explained by the long serialisation sequences that are considered. Interestingly, when the ratio C_{ω_i}/C_i increases, the pessimism decreases. This is because the probability to observe longer serialisation sequences grows as contention increases. Similar trends are also followed by the "linear time method" (see Figure 7a), although the derived upper bounds introduce more pessimism as expected. The average ratios range from 11.1 to 35.5 and the "worst" ratios from 246 up to 1281. Figure 7c compares the two methods and displays the extra overhead of the linear method. In average, the linear method adds about 100% more overheads (with $\sigma \simeq 5\%$) and this value tends to decrease as the number of cores grows.

Results for SRPTM. From an analysis viewpoint, we showed that NPUC is less pessimistic than SRPTM. This is due to the fact that the SRPTM involves extra preemption overhead as some transactions may be delayed by higher priority tasks that do not carry a transaction. However, the results for SRPTM have a similar trend as those for NPUC (see Figure 7b). The average (resp. maximum) ratios range from 11.1 (resp. 246) up to 36.6 (resp. 1281). Although the range limits are the same as those of the linear method for NPUC, the standard deviations in this case are higher.

IX. Conclusions

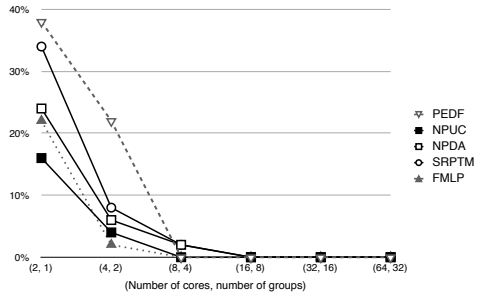
This paper addresses the response time analysis of hard real-time tasks, which share STM data under partitioned scheduling strategies. A framework wherein an upper-bound on the worst-case response time of each task has been discussed while assuming three scheduling approaches – namely NPDA, NPUC and SRPTM. We assume that transactions are serialised according to their arrival time and they are managed by following FIFO-CRT. Although NPDA was a promising approach for soft real-time tasks upon multi-core platforms, we showed that its associated analysis is intractable in the context of hard real-time tasks. Assuming NPUC, we provided a tight analysis and a linear-time analysis, which reduces the computational complexity of the proposed analysis of the former. Finally, we showed that SRPTM has good performances both from an analytic and experimental viewpoints. Interesting future directions would address the impact of the task-to-core mapping on the analysis.

Acknowledgment

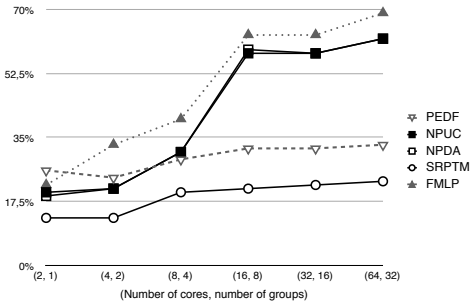
This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within the CISTER Research Unit (CEC/04234).

References

[1] A. Block, H. Leontyev, B. B. Brandenburg, and J. H. Anderson, "A Flexible Real-Time Locking Protocol for Multiprocessors," in *13th IEEE RTCSA*, Daegu, South Korea, 2007, pp. 47–56.

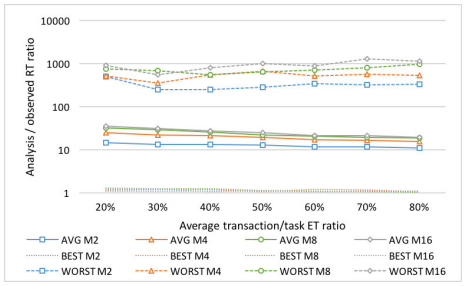


(a) Feasibility ratio for task sets with at least 75% of system capacity. 75% of tasks carry an atomic section.

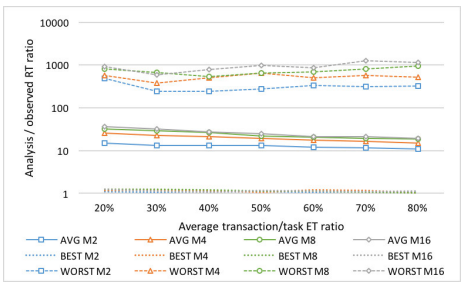


(b) Atomic section time overhead.

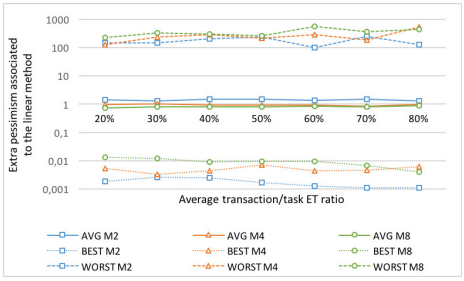
Fig. 6. Runtime performance of scheduling strategies.



(a) Linear (NPUC).



(b) Linear (SRPTM).



(c) Linear-time to tight upper bounds ratio.

Fig. 7. Ratio between the analytic upper-bounds and the maximum observed transaction response times.

[2] B. B. Brandenburg and J. H. Anderson, "Optimality results for multiprocessor real-time locking," in *IEEE 31st RTSS*, San Diego, USA, 2010, pp. 49–60.

[3] B. C. Ward and J. H. Anderson, "Supporting nested locking in multiprocessor real-time systems," in *24th ECRTS*, Pisa, Italy, 2012, pp. 223–232.

[4] ——, "Multi-resource real-time reader/writer locks for multiprocessors," in *IEEE 28th IPDPS*, Phoenix, USA, 2014, pp. 177–186.

[5] A. Dragojević, P. Felber, V. Gramoli, and R. Guerraoui, "Why STM can be more than a research toy," *Communications of the ACM*, vol. 54, no. 4, pp. 70–77, 2011.

[6] C. J. Rossbach, O. S. Hofmann, and E. Witchel, "Is transactional programming actually easier?" in *15th ACM SIGPLAN symposium on PPoPP*, Bangalore, India, 2010, pp. 47–56.

[7] A. Barros and L. M. Pinho, "Non-preemptive scheduling of real-time software transactional memory," in *27th Int. Conf. on ARCS*, Lübeck, Germany, 2014, pp. 25–36.

[8] J. H. Anderson, S. Ramamurthy, M. Moir, and K. Jeffay, "Lock-free transactions for real-time systems," in *Real-Time Database Systems: Issues and Applications*, USA, 1997, pp. 215–234.

[9] J. H. Anderson, R. Jain, and S. Ramamurthy, "Implementing hard real-time transactions on multiprocessors," in *Real-Time Database and Information Systems: Research Advances*, USA, 1997, pp. 247–260.

[10] J. Manson, J. Baker, A. Cunei, S. Jagannathan, M. Prochazka, B. Xin, and J. Vitek, "Preemptible Atomic Regions for Real-Time Java," in *26th IEEE RTSS*, Miami, USA, Dec. 2005, pp. 62–71.

[11] J. Ras and A. M. K. Cheng, "Response time analysis for the Abort-and-Restart event handlers of the Priority-Based Functional Reactive Programming (P-FRP) paradigm," in *15th IEEE Int. Conf. on Embedded and RTCSA*, 2009, pp. 305–314.

[12] S. F. Fahmy, B. Ravindran, and E. D. Jensen, "On bounding response times under software transactional memory in distributed multiprocessor real-time systems," in *DATE Conference & Exhibition*, Nice, France, 2009, pp. 688–693.

[13] J. H. Anderson and A. Srinivasan, "Mixed Pfair/ERfair scheduling of asynchronous periodic tasks," in *13th ECRTS*, Delft , Netherlands, jun 2001, pp. 76–85.

[14] T. Sarni, A. Queudet, and P. Valduriez, "Real-time support for software transactional memory," in *15th IEEE Int. Conf. on Embedded and RTCSA*, Beijing, China, 2009, pp. 477–485.

[15] A. Barros, L. M. Pinho, and P. M. Yomsi, "Non-preemptive and SRP-based fully-preemptive scheduling of real-time software transactional memory," *JSA*, vol. 61, no. 10, pp. 553–566, 2015.

[16] S. Cotard, "Contribuition á la robustesse des systémes temps réel embarqués multicoeur automobile," Ph.D. dissertation, Université de Nantes, 2013.

[17] M. El-Shambakey and B. Ravindran, "STM concurrency control for multicore embedded real-time software: time bounds and tradeoffs," in *27th Annual ACM SAC*. Riva del Garda, Italy: ACM, Mar. 2012, pp. 1602–1609.

[18] ——, "STM concurrency control for embedded real-time software with tighter time bounds," in *49th Annual DAC*. San Francisco, USA: ACM, Jun. 2012, pp. 437–446.

[19] ——, "On real-time STM concurrency control for embedded software with improved schedulability," in *18th ASP-DAC*, Yokohama, Japan, Jan. 2013, pp. 47–52.

[20] ——, "FBLT: A Real-Time Contention Manager with Improved Schedulability," in *Conference on DATE*. Grenoble, France: EDA Consortium, Mar. 2013, pp. 1325–1330.

[21] T. P. Baker, "Stack-based scheduling of realtime processes," *Real-Time Systems*, vol. 3, no. 1, pp. 67–99, 1991.

[22] M. Spuri, "Analysis of deadline scheduled real-time systems," INRIA, Tech. Rep., 1996.

[23] L. George, N. Rivierre, and M. Spuri, "Preemptive and non-preemptive real-time uniprocessor scheduling," INRIA, Tech. Rep., 1996.