Bounding Deadline Misses in Weakly-Hard Real-Time Systems with Task Dependencies

Zain A. H. Hammadeh, Rolf Ernst
TU Braunschweig, Germany
{hammadeh, ernst}@ida.ing.tu-bs.de

Sophie Quinton
Inria Grenoble, France
sophie.quinton@inria.fr

Rafik Henia, Laurent Rioux
Thales Research & Technology, France
{rafik.henia, laurent.rioux}@thalesgroup.com

***Abstract*—Real-time systems with functional dependencies between tasks often require end-to-end (as opposed to task-level) guarantees. For many of these systems, it is even possible to accept the possibility of longer end-to-end delays if one can bound their frequency. Such systems are called weakly-hard.**

In this paper we provide end-to-end deadline miss models for systems with task chains using Typical Worst-Case Analysis (TWCA). This bounds the number of potential deadline misses in a given sequence of activations of a task chain. To achieve this we exploit task chain properties which arise from the priority assignment of tasks in static-priority preemptive systems. This work is motivated by and validated on a realistic case study inspired by industrial practice and derived synthetic test cases.

I. Introduction and Related Work

Timing performance analysis of real-time systems with concurrently executing task chains is notoriously difficult due to the complexity of timing interference between tasks. This is all the more true when task chains are derived from communicating threads [9]. In this paper, we are interested in the analysis of end-to-end guarantees for *weakly-hard* systems with task dependencies, i.e., systems for which it is possible to accept the possibility of end-to-end deadline misses if one can bound their frequency [1].

We present a method to compute *end-to-end deadline miss models* for static-priority preemptive systems with task chains. This bounds the number of potential deadline misses in a given sequence of executions of a task chain. Our approach is an extension of Typical Worst-Case Analysis (TWCA) [8], [10], for which we exploit task chain properties derived from the priority assignment of tasks in a way similar to [9].

To the best of our knowledge, there is no state-of-the-art method for the computation of weakly-hard guarantees in real-time systems with task dependencies.

Extensive research has focused on the schedulability analysis of *hard* real-time systems with task dependencies. This includes approaches focusing on offset analysis [2] but also more general precedence models [3]. In [9], an upper bound on the end-to-end latency of task chains in real-time systems is presented, on which we will base our work in this paper.

In contrast, there is little in the literature regarding the analysis of weakly-hard systems. Initial attempts [4], [1] can only handle periodic tasks (or sporadic tasks but using a coarse interarrival time model) and no task dependencies. Recent work has focused on providing guarantees for systems with more complex activation patterns [8], [5], [10] and [6], mostly relying on the so-called TWCA approach. None of these, however, can handle task dependencies.

The paper is organized as follows: Section II introduces our system model and formulates the problem that we address. Then, Section III explains the basic principles of TWCA. Section IV proposes an improved version of the worst-case latency analysis of [9] which we use in V for the core contribution of our paper. Finally, Section VI shows our experimental results while Section VII proposes some conclusions.

II. System Model

We consider uniprocessor systems consisting of a finite set of m disjoint *task chains* scheduled according to the Static Priority Preemptive (SPP) scheduling policy. A task chain is a sequence of distinct tasks which activate each other. Tasks in a system are required to belong to exactly one chain[1]. Formally, a task chain σ_a, $a \in [1, m]$, is defined by a finite sequence $(\tau_a^1, \tau_a^2, \ldots, \tau_a^n)$ of distinct tasks for some $n \in \mathbb{N}^+$, meaning that the output of τ_a^i is connected to the input of τ_a^{i+1} for $i \in [1, n-1]$. Every task chain σ_a is assigned an *activation model* (see definition below) defining the frequency of arrival at the input of τ_a^1; and a *relative deadline* D_a.

The tasks in σ_a are denoted τ_a^i, τ_a^j etc. Task τ_a^i denotes the i-th task in task chain σ_a. The number of tasks in σ_a is denoted n_a. The first task in σ_a is called the *header task* of σ_a and the last one is called its *tail task*.

Figure 1 shows an example system with two task chains: $\sigma_a = (\tau_a^1, \tau_a^2, \tau_a^3, \tau_a^4, \tau_a^5, \tau_a^6)$, $\sigma_b = (\tau_b^1, \tau_b^2, \tau_b^3)$.

We denote $\mathcal{C}$ the set of task chains. This set is partitioned into $\mathcal{SC}$ and $\mathcal{AC}$, which contain respectively the *synchronous* and *asynchronous* chains. Synchronous and asynchronous chains are specified in the same way but behave differently at execution: In a synchronous chain σ_a an incoming activation cannot be processed until the previous instances of σ_a have finished [9]. In an asynchronous chain σ_b an incoming activation is processed independently from previous instances.

The activation models of task chains are defined using *arrival curves* as in e.g. [7], i.e., functions $\eta_a^-, \eta_a^+ : \mathbb{N} \to \mathbb{N}$ such that for any time window ΔT, $\eta_a^+(\Delta T)$ defines the

This work has been partially funded by the German Research Foundation (DFG) as part of the project “TypicalCPA” under the contract number TWCA ER168/30-1.

[1]To analyze systems that are not only made of disjoint task chains but also contain forks and joins (but no cycle), one can additionally define *paths*, i.e. sequences of distinct task chains. This is out of the scope of this paper.

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

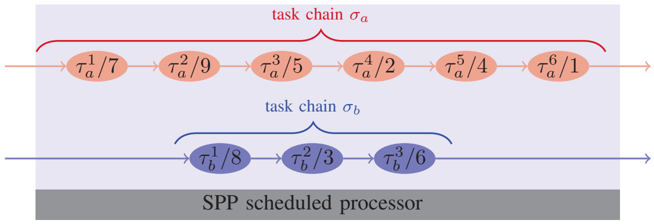

Figure 1. A system task structure with chains and task priorities

maximum number of activations of chain σ_a that might occur within ΔT, and $\eta_a^-(\Delta T)$ the minimum (in this paper we only use η_a^+). We will also need the pseudo-inverse representation of arrival curves, namely $\delta_b^-, \delta_b^+ : \mathbb{N} \to \mathbb{N}$, such that $\delta_b^-(k)$ (respectively $\delta_b^+(k)$) defines the minimum (respectively maximum) time that might pass between the first and the last activation in any sequence of k consecutive activations of σ_b.

A task τ_a^i is defined by: (1) an arbitrary priority π_a^i and (2) an upper bound on its execution time C_a^i (we take 0 as a lower bound). The notation $\pi_a^i > \pi_b^j$ is used to denote that τ_a^i has higher priority than τ_b^j. As a result, τ_a^i may preempt τ_b^j when it arrives. We also use the notation $\pi_a^i \geqslant \pi_b^j$.

The timing behavior of a task τ_a^i is an infinite sequence of instances defined by: an arrival time, possible preemption delays and a finish time. Preemption delays are due to the task being blocked by higher priority tasks from other task chains, but also by higher priority tasks from the same chain if it is asynchronous. In contrast, tasks in a synchronous chain cannot be preempted by other tasks of the same chain, even if they have higher priority. Task τ_a^i finishes latest after having been scheduled for C_a^i units of time.

The timing behavior of a task chain σ_a is an infinite sequence of task chain instances, where a task chain instance is made of one instance of each task in the chain such that the finish time of task τ_a^i corresponds to the arrival time of task τ_a^{i+1} (assuming τ_a^i is not the last task in the chain). The arrival of task τ_a^1 follows the activation model of σ_a.

The *latency* of an instance of a task chain σ_a is the time interval between the activation of the header task of σ_a and the finish time of the tail task of σ_a. The *worst-case latency* of σ_a is the maximum latency over all instances of σ_a. An instance of σ_b is said to *miss its deadline* if its latency exceeds the relative deadline of σ_b. This can happen in weakly-hard real-time systems. We consider a simple, deadline-agnostic scheduler that does not anticipate, monitor or react to deadline misses but instead runs every instance to completion, independent of whether a deadline miss has occurred or not.

As usual in TWCA, we suppose that deadline misses are caused by rarely activated sporadic chains, e.g., interrupt service routines or recovery chains. These chains cause transient overload, increasing chain latencies which may cause deadline misses, hence their name: *overload chains*. We assume that the set of overload chains is identified and denoted $\mathcal{C}_{over}$.

Definition 1. *A* deadline miss model *for a task chain σ_b is a function $dmm_b : \mathbb{N}^+ \to \mathbb{N}$ such that $dmm_b(k)$ bounds the maximum number of deadline misses in a window of k consecutive executions of σ_b.*

In this paper, we address the problem of computing DMMs of task chains in systems which contain overload task chains.

III. Principle of Typical Worst-Case Analysis

Typical Worst-Case Analysis (TWCA) is a technique to compute *deadline miss models* which bound the number of deadline misses in a sequence of activations of a given task. TWCA applies to systems of *independent tasks* which may occasionally miss deadlines due to *overload tasks*. We recall here the principle of TWCA and refer to [10] for more detail.

Formally, a Deadline Miss Model (DMM) for a task τ_i is a function $dmm_i : \mathbb{N}^+ \to \mathbb{N}$ such that $dmm_i(k)$ bounds the maximum number of deadline misses that τ_i may experience out of a sequence of k consecutive executions. The DMM computation is based on the analysis of *unschedulable combinations*, i.e., sets of overload tasks which, when activated together, may lead to a deadline miss. More formally, a *combination*, denoted $\bar{c}$, is a set of overload tasks. $\bar{c}$ is *schedulable* (with respect to τ_i) if an instance of τ_i is guaranteed to meet its deadline as long as only tasks in $\bar{c}$ experience overload activations in its level-i busy window, where a *level-i busy window* is a maximal time interval during which the processor has activations of τ_i or higher priority tasks pending.

Let us consider a sequence of k activations of a given task τ_i and focus on the computation of $dmm_i(k)$. Note that the sequence may span multiple busy windows. The activation model of the overload tasks bounds the number of activations of these tasks (also called overload activations) which may arrive during the considered sequence. Assuming we have all unschedulable combinations at hand, the problem is then to find how to assign overload activations to busy windows so as to pack as many unschedulable combinations as possible into the level-i busy windows under consideration. Therefore the problem becomes a multi-dimensional knapsack problem.

Example. Figure 2 illustrates two possible packings of overload activations into 5 busy windows. Every row corresponds to one overload task while every column corresponds to one busy window of τ_i. The number of activations per line is constrained by the activation models of the overload tasks. The number of deadline misses associated to a given packing depends on how many columns are unschedulable combinations. Here, any combination containing more than one task is unschedulable.

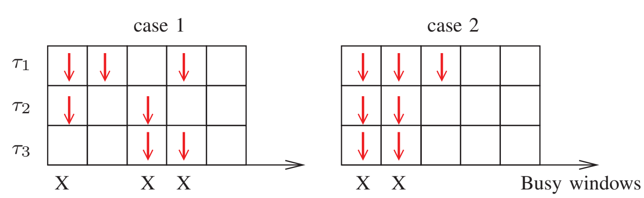

Figure 2. Packing combinations into busy windows (X = deadline miss).

So far, TWCA can only handle independent tasks. In the rest of this paper we show how the state-of-the-art approach can be generalized to systems with task chains.

IV. Latency Analysis Revisited

Let us first revisit the worst-case latency analysis of systems with task chains [9]. Consider two chains σ_a and σ_b. To quantify the interference of σ_a on σ_b we distinguish two cases:

1) *some* tasks in σ_a have lower priority than *all* tasks in σ_b; in that case, σ_a will be blocked by σ_b every time it reaches one of those tasks.
2) In any other case, σ_a is said to *arbitrarily interfere with* σ_b. This means that every time σ_a is triggered, we suppose that it may entirely execute before σ_b can be scheduled again. As we will see later, there is no guarantee however that this will happen.

Definition 2. *A chain σ_a is said to be* deferred by *chain σ_b if*

$$\exists i \in [1, n_a], \pi_a^i < \min\{\pi_b^j\}_{j=1}^{n_b}$$

Otherwise it is arbitrarily interfering with σ_b.

The set of chains deferred by σ_b is denoted $\mathcal{DC}(b)$ and the set of chains arbitrarily interfering with σ_b is denoted $\mathcal{IC}(b)$.

For a chain σ_a which is arbitrarily interfering with σ_b, interference on σ_b can be directly derived from the number of activations of σ_a. If σ_a is, however, deferred by σ_b, then interference is defined based on the concept of *segment* of σ_a w.r.t. σ_b. Intuitively, a segment of σ_a w.r.t. σ_b represents a subchain of σ_a that may interfere with σ_b.

Definition 3. *A* segment of σ_a w.r.t σ_b *is a maximal subchain $(\tau_a^i, \tau_a^{i+1}, \ldots, \tau_a^{i+k})$ of σ_a, $i \in [1, n_a]$ and $k \in [0, n_a-1]$, with the convention*[2] *that task identifiers should be read modulo n_a and such that*

$$\forall l \in [0, k], \pi_a^{i+l} \geqslant \min\{\pi_b^j\}_{j=1}^{n_b}$$

Note that we (conservatively) assume that a segment may span over two instances of σ_b. S_a^b denotes all such segments.

Example. Chain σ_a in Figure 1 has 2 segments w.r.t. chain σ_b: $(\tau_a^1, \tau_a^2, \tau_a^3)$ and (τ_a^5). Note that τ_a^4 and τ_a^6 have lower priority than τ_b^2 and are therefore not part of any segment.

Definition 4. *The* critical segment *of a chain σ_a deferred by σ_b, denoted $s_{a,b}^{crit}$, is the segment $(\tau_a^i, \tau_a^{i+1}, \ldots, \tau_a^{i+k})$ of σ_a w.r.t. σ_b that maximizes computation time, i.e., $\sum_{0 \leqslant l \leqslant k} C_a^{i+l}$.*

Definition 5. *Consider an asynchronous chain σ_a. We denote:*

- *s_a^{header} the subchain $(\tau_a^1, \tau_a^2, \ldots, \tau_a^i)$ where $i \in [1, n_a-1]$ is the smallest integer such that τ_a^{i+1} has the lowest priority in σ_a. If τ_a^1 has the lowest priority then s_a^{header} is empty.*
- *if σ_a is deferred by σ_b then we denote $s_{a,b}^{header}$ the* header segment of σ_a w.r.t. σ_b *defined as the subchain $(\tau_a^1, \tau_a^2, \ldots, \tau_a^i)$ where $i \in [1, n_a-1]$ is the smallest integer such that τ_a^{i+1} has lower priority than all tasks in σ_b.*

We now revisit the worst-case latency analysis introduced in [9] and propose a description that is similar to worst-case response-time analysis as explained in [8].

[2]That is, if $i + l > n_a$ then it should be read $(i + l) \bmod n_a$.

Definition 6. *A σ_b-*busy-window *is a maximal time interval during which (at least) one instance of σ_b is pending, i.e., it has been activated but has not finished yet.*

Definition 7. *The q-*event busy time *of a chain σ_b is the maximum time it may take to process q activations of σ_b within a σ_b-busy-window starting with the first of these q activations.*

Theorem 1. *The q-event busy time of σ_b is bounded by*

$$\begin{aligned} B_b(q) \;=\; & q \times C_b \\ & + \max(0, \eta_b^+(B_b(q)) - q) \times C_{s_b^{header}} \text{ if } \sigma_b \in \mathcal{AC} \\ & + \sum_{\sigma_a \in \mathcal{IC}(b)} \eta_a^+(B_b(q)) \times C_a \\ & + \sum_{\sigma_a \in \mathcal{AC} \cap \mathcal{DC}(b)} \eta_a^+(B_b(q)) \times C_{s_{a,b}^{header}} + \sum_{s \in S_a^b} C_s \\ & + \sum_{\sigma_a \in \mathcal{SC} \cap \mathcal{DC}(b)} C_{s_{a,b}^{crit}} \end{aligned} \tag{1}$$

where C_x denotes the sum of the execution time bounds of the tasks in segment or chain x.

Proof. The above equation is made of five components:

1) The first line corresponds to the time needed to actually perform the q computations;
2) The second component accounts for the interference of additional activations of σ_b which may arrive while the q activations under consideration are being processed. Note that these instances will at most interfere until they have to execute the lowest priority task in σ_b. This component only applies to asynchronous chains;
3) The third element represents the interference from arbitrarily interfering chains, synchronous or asynchronous;
4) The fourth line deals with interference from deferred, asynchronous chains. Instances can arbitrarily queue up which allows the header segment to interfere arbitrarily. For all other segments at most one instance can be backlogged because tasks between segments have lower priority than tasks within segments. Each such instance can interfere for at most one segment (see below).
5) The fifth component in the equation accounts for the interference from deferred, synchronous chains. Here only one instance per chain may interfere for at most one segment (see below). □

The correctness of the last two components in Equation (1) relies on the following property.

Lemma 1. *Tasks of a chain σ_a that are in different segments cannot execute instances corresponding to the same chain instance in the same σ_b-busy-window.*

Proof. Segments are maximal sequences of tasks with a priority higher than or equal to the lowest priority task, say τ_b^i, in σ_b. This means that between two segments of σ_a there is at least one task, say τ_a^j, that has lower priority than τ_b^i. In order to execute these two segments for the same instance of σ_a, one has to execute τ_a^j. Since τ_a^j has lower priority than all

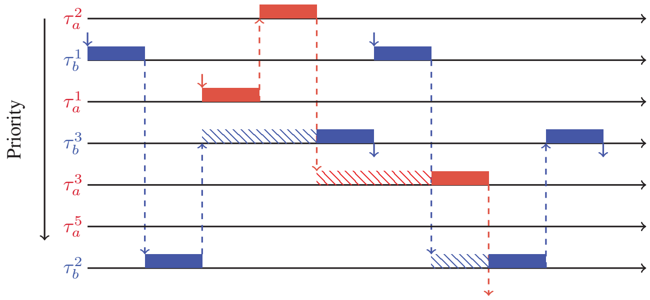

Figure 3. Number of impacted busy windows of chain b.

the tasks in σ_b, this can only happen after σ_b closes its current σ_b-busy-window. □

Theorem 2. *The maximum number of activations of σ_b in a σ_b-busy-window is*

$$K_b = \min\{q \geqslant 1 \mid B_b(q) \leqslant \delta_b^-(q+1)\}$$

The latency of a task chain σ_b is bounded by

$$WCL_b = \max_{q \in [1, K_b]} \{B_b(q) - \delta_b^-(q)\}$$

Proof. This proof proceeds exactly as the proofs in [8]. □

The main objective of TWCA is to bound the number of deadlines misses of a task chain σ_b which may be caused by an activation at the input of an overload task chain σ_a. For that, we need to know over how many σ_b-busy-windows a instance of σ_a may span.

We already know that, in a chain σ_a, the execution of tasks corresponding to the same instance of σ_a cannot take place in the same σ_b-busy-window if those tasks are in different segments. This implies that an instance of σ_a spans over at least as many σ_b-busy-windows as there are segments of σ_a w.r.t. σ_b.

Note that there is no guarantee that a segment of σ_a will be executed within one σ_b-busy-window. As an example, in Figure 3 the execution of segment $(\tau_a^1, \tau_a^2, \tau_a^3)$ spans over two σ_b-busy-windows. We therefore introduce the notion of *active segment*, which applies to subsegments which are guaranteed to be executed in the same σ_b-busy-window.

Definition 8. *An* active segment of σ_a w.r.t σ_b *is a subchain*[3] *of a segment $(\tau_a^i, \tau_a^{i+1}, \ldots, \tau_a^{i+k})$ of σ_a where $i \in [1, n_a]$ and $k \in [0, n_a - i]$ such that*

$$\forall l \in [1, k], \pi_a^{i+l} \geqslant \pi_b^{tail}$$

where τ_b^{tail} denotes the tail task of σ_b.

Example. In Figure 1, chain σ_a has three active segments: $(\tau_a^1, \tau_a^2), (\tau_a^3), (\tau_a^5)$.

Lemma 2. *The execution of an active segment of σ_a w.r.t. σ_b cannot span over more than one σ_b-busy-window.*

[3]Here, $i + l$ is always smaller than or equal to n_a.

Proof. Once the execution of an active segment of σ_a w.r.t. σ_b has started, τ_b^{tail} will not be able to execute because the active segment is blocking it or a task preceding it, and therefore the current σ_b-busy-window cannot be closed, until the whole segment has finished executing. □

This lemma is illustrated in Figure 3, where every active segment of chain σ_a executes within one σ_b-busy-window.

Note that an active segment is part of a segment in the sense of Definition 3. As a result, we easily conclude from Lemma 1 and 2 that two active segments of chain σ_a may be executed within one σ_b-busy-window if and only if they are part of the same segment of σ_a.

V. TWCA FOR TASK CHAINS

We now have all the ingredients needed to show how we extend TWCA to handle task chains. We follow here the same approach as the one for systems with independent tasks explained in Section III. For the rest of the section we suppose given a chain σ_b and $k \geqslant 1$ and focus on the computation of $dmm_b(k)$, that is, a bound on the number of deadlines that σ_b can miss out of a k-sequence, i.e., k consecutive activations. Similar to [10], we assume that there is at most one activation of an overload chain σ_a in a σ_b-busy-window. As a result, we can without loss of generality consider our overload task chains as synchronous.

A. Combinations for TWCA of task chains

For the case where tasks are independent, a *combination* is defined as a set of overload tasks. The DMM computation based on this definition heavily relies on the fact that one overload activation impacts exactly one busy window. In the context of task chains, we have seen in the previous sections that one instance of a task chain σ_a may span over several σ_b-busy-windows. As a result, the impact of one overload activation is not here limited to one σ_b-busy-window. We have however also shown that the execution an active segment of σ_a is restricted to a single σ_b-busy-window. Hence our choice to define combinations based on active segments rather than tasks or task chains.

Definition 9. *A* combination *$\bar{c}$ is a set of active segments w.r.t. σ_b such that if two active segments of the same chain σ_a are in $\bar{c}$ then they are part of the same segment of σ_a w.r.t. σ_b.*

Note that our definition excludes combinations which cannot execute within one σ_b-busy-window based on our definition of segment.

Example. There are four possible combinations of the active segments of chain σ_a in Figure 1: $\{(\tau_a^1, \tau_a^2)\}, \{(\tau_a^3)\}, \{(\tau_a^5)\}, \{(\tau_a^1, \tau_a^2), (\tau_a^3)\}$.

Definition 10. *A combination $\bar{c}$ is* schedulable *(w.r.t. σ_b) if σ_b is guaranteed not to miss any deadline in a σ_b-busy-window in which only the active segments in $\bar{c}$ execute (in addition to non-overload chains). Otherwise $\bar{c}$ is said to be* unschedulable.

B. An ILP formulation for the DMM

Having clarified the notion of combination that we use, we can now state our main theorem, similar to [10].

Theorem 3. *Let us define $dmm_b(k)$ as*

$$\max \left\{ N_b \sum_{\bar{c} \in \mathcal{U}} x_{\bar{c}} \,|\, \forall \sigma_a \in \mathcal{C}_{over}, \forall s \in \mathcal{S}_a, \sum_{\{\bar{c} \in \mathcal{U} \,|\, s \in \bar{c}\}} x_{\bar{c}} \leqslant \Omega_b^a \right\} \quad (2)$$

where

- *N_b is the maximum number of deadlines that σ_b can miss in one busy window;*
- *$\mathcal{U}$ is the set of unschedulable combinations;*
- *$x_{\bar{c}}$ is the variable constraining the number of busy windows that could contain one activation of the k-sequence and suffer from an overload corresponding to $\bar{c} \in \mathcal{U}$;*
- *$\mathcal{S}_a$ denotes the set of active segments of σ_a;*
- *Ω_b^a is the maximum number of activations of σ_a which could impact the considered k activations of σ_a.*

Then $dmm_b(k)$ is a DMM for σ_b.

The formal definition of N_b and Ω_b^a is given below. Because $\mathcal{U}$ can be too large to be statically constructed, Section V-C discusses an efficient criterion to determine whether a combination is in $\mathcal{U}$. The $x_{\bar{c}}$ are the variables of our ILP problem.

Proof. Assume that we have Ω_b^a for all chains σ_a, i.e. the maximum number of activations of σ_a which could impact the k-sequence. In the worst case, each active segment of σ_a also impacts σ_b Ω_b^a times. As in Section III, we here also face a multi-dimensional knapsack problem where items correspond to unschedulable combinations and capacities to Ω_b^a for every line s associated with an active segment of overload chain σ_a. So considering that $x_{\bar{c}}$ stands for the number of times that a combination $\bar{c}$ is used in the packing under consideration, we want to find the packing that maximizes the number of deadline misses of σ_b — which is equal to the number of unschedulable combinations used multiplied by the maximum number of deadline misses due to each combination. This packing is constrained by the fact that active segments cannot be used in more combinations than is allowed by their corresponding Ω_b^a. $\square$

Let us now formally define N_b and Ω_b^a.

Lemma 3.

$$N_b = \#\{q \in [1, K_b] \mid B_b(q) - \delta_b^-(q) > D_b\}$$

Proof. The proof proceeds exactly like that of Theorem 2. $\square$

Lemma 4. *The maximum number Ω_b^a of activations of σ_a which could impact the considered k activations of σ_a is*

$$\Omega_b^a = \eta_a^+(\delta_b^+(k) + WCL_b) + 1$$

Proof. Clearly, activations of chain σ_a which occur after the first instance of chain σ_b in the k-sequence is activated and before the last activation in the k-sequence finishes may have an impact on the latencies in the k-sequence. There are at most $\eta_a^+(\delta_b^+(k) + WCL_b)$ such activations. In contrast, an instance of σ_a which arrives after the last instance of chain σ_b in the k-sequence has finished does not impact the k-sequence. Finally, we have assumed that there is at most one activation of σ_a in a σ_b-busy-window so that at most one activation of σ_a before the k-sequence can impact it. $\square$

C. Criterion of schedulability

As already mentioned, $\mathcal{U}$ can be too large to be statically constructed. We present here an efficient criterion to determine whether a combination $\bar{c}$ is in $\mathcal{U}$ or not. Let us reorganize Equation 1 for the multiple busy-time computation to show explicitly the contribution of the overload chains of a combination in the multiple busy time (and the latency) of σ_b.

$$\begin{aligned} B_b^{\bar{c}}(q) \;=\; & q \times C_b \\ & + \max(0, \eta_b^+(B_b^{\bar{c}}(q)) - q) \times C_{s_b^{header}} \text{ if } \sigma_b \in \mathcal{AC} \\ & + \sum_{\sigma_a \in \mathcal{IC}(b) \setminus \mathcal{C}_{over}} \eta_a^+(B_b^{\bar{c}}(q)) \times C_a \\ & + \sum_{\sigma_a \in \mathcal{AC} \cap \mathcal{DC}(b)} \eta_a^+(B_b(q)) \times C_{s_{a,b}^{header}} + \sum_{s \in S_a^b} C_s \\ & + \sum_{\sigma_a \in \mathcal{SC} \cap \mathcal{DC}(b) \setminus \mathcal{C}_{over}} C_{s_b^a} \\ & + \sum_{\sigma_a \in \mathcal{C}_{over}} \sum_{s \in \mathcal{S}_a} C_s \times r_s^{\bar{c}} \end{aligned} \quad (3)$$

where $r_s^{\bar{c}}$ is a Boolean which holds exactly when $s \in \bar{c}$.

A combination $\bar{c}$ is schedulable if $B_b^{\bar{c}}(q) - \delta_b^-(q) \leqslant D_b$ for all $q \in [1, K_b]$. Now, let us define $L_b(q)$ as follows.

$$\begin{aligned} L_b(q) = \; & q \times C_b \\ & + \max(0, \eta_b^+(\delta_b^-(q) + D_b) - q) \times C_{s_b^{header}} \text{ if } \sigma_b \in \mathcal{AC} \\ & + \sum_{\sigma_a \in \mathcal{IC}(b) \setminus \mathcal{C}_{over}} \eta_a^+(\delta_b^-(q) + D_b) \times C_a \\ & + \sum_{\sigma_a \in \mathcal{AC} \cap \mathcal{DC}(b)} \eta_a^+(\delta_b^-(q) + D_b) \times C_{s_{a,b}^{header}} + \sum_{s \in S_a^b} C_s \\ & + \sum_{\sigma_a \in \mathcal{SC} \cap \mathcal{DC}(b) \setminus \mathcal{C}_{over}} C_{s_b^a} \end{aligned} \quad (4)$$

Then we now have a much simpler sufficient condition for schedulability: $\bar{c}$ is schedulable if

$$\forall q \in [1, K_b], \; L_b(q) + \sum_{\sigma_a \in \mathcal{C}_{over}} \sum_{s \in \mathcal{S}_a} C_s \times r_s^{\bar{c}} \leqslant \delta_b^-(q) + D_b \quad (5)$$

We have now shown how we can reuse the ILP solution of [10] for systems with task chains with limited changes.

VI. Experimental Results

We have experimented with a case study directly derived from industrial practice at Thales Research & Technology. The system is a single-core processor scheduled according to SPP. Figure 4 shows the specified task set and the real-time attributes of each task. In the following experiments we focus on providing DMMs for σ_c and σ_d.

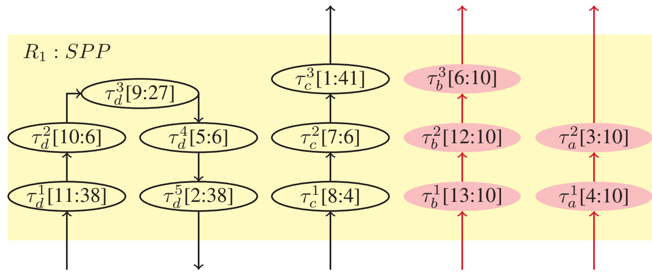

Figure 4. Model of our case study. We use the following notations: task chains are specified as $\sigma[\delta^-(2) : D]$ and tasks with $\tau[\pi : C]$. Chains σ_c and σ_d are periodically activated while σ_a and σ_b are sporadic, overload chains.

Experiment 1. We first compute the worst-case latency WCL of task chains σ_c and σ_d as described in Section IV. The analysis results show that the system is not schedulable as σ_c can in the worst-case miss its deadline, see Table I.

task chain	WCL	D
σ_c	331	200
σ_d	175	200

Table I
WCL OF TASK CHAINS σ_c AND σ_d

A second analysis, in which all overload chains are abstracted away, reveals that the system is schedulable and σ_c meets its deadline if neither σ_a nor σ_b are activated. We thus perform TWCA as presented in this paper. The computed DMM of σ_c is shown in Table II — σ_d is schedulable and therefore does not need a DMM.

task chain	DMM
σ_c	$dmm_c(3) = 3$, $dmm_c(76) = 4$, $dmm_c(250) = 5$

Table II
$dmm(k)$ FOR TASK CHAIN σ_c

Let us provide additional details resulting from this DMM computation. Both chains σ_a and σ_b arbitrarily interfere with σ_c because neither has a task with a priority lower than 1 which is the lowest priority in σ_c. As a result σ_a and σ_b have only one segment, respectively (τ_a^1, τ_a^2) and $(\tau_b^1, \tau_b^2, \tau_b^2)$. These two segments are also active segments because the priority of the tail task of chain σ_c is lower than all priorities in these segments (see figure 4). Therefore no constraints on combining active segments are needed. Our set of combinations thus has three elements: $\bar{c}_1 = \{(\tau_a^1, \tau_a^2)\}$, $\bar{c}_2 = \{(\tau_b^1, \tau_b^2, \tau_b^3)\}$, and $\bar{c}_3 = \{(\tau_a^1, \tau_a^2), (\tau_b^1, \tau_b^2, \tau_b^3)\}$. Based on the schedulability criterion we introduced in the previous section we conclude that $\bar{c}_3$ is the only unschedulable combination, so in this case the TWCA is fairly simple.

We now want to generalize the results obtained on our industrial case study, while preserving practical relevance. For that purpose, we arbitrarily modify the priority assignment so as to generate random systems with different scenarios.

Experiment 2. We arbitrarily assign priorities to show the impact of priority assignments on the schedulability and the deadline miss models. In this experiment we randomly choose 1000 assignments to test our analysis intensively. Figure 5 shows $dmm_c(10)$ and $dmm_d(10)$. Notice first that out of

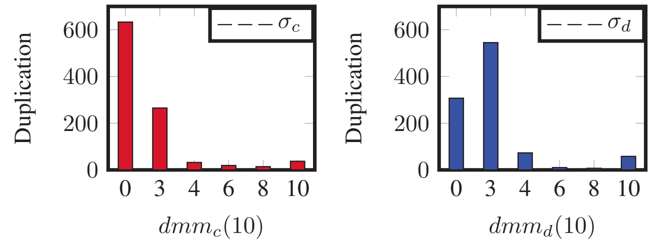

Figure 5. $dmm_c(10)$ and $dmm_d(10)$

the 1000 assignments generated, chain σ_c is schedulable (misses no deadline) 633 times. More interestingly, chain σ_d is schedulable only 307 times out of 1000. TWCA in that case is very useful as for more than 500 of the remaining systems it can guarantee that no more than 3 out 10 deadlines can be missed. Note that we have repeated our experiment 30 times and observed similar results.

VII. Conclusion

In this paper we present the first method for computing end-to-end deadline miss models for systems with task dependencies, using Typical Worst-Case Analysis (TWCA). This bounds the number of potential deadline misses in a given sequence of activations of a task chain. Our approach addresses uniprocessor systems with Static Priority Preemptive scheduling. We show how state-of-the-art TWCA can be extended using recent results in the analysis of hard real-time systems with task dependencies. Specifically, we show how we can formulate our problem as a knapsack problem. Our approach is validated on a realistic case study inspired by industrial practice and synthetic variants of it.

This paper is an important step towards using TWCA for the practical design of distributed embedded systems.

References

[1] G. Bernat, A. Burns, and A. Llamosí. Weakly hard real-time systems. *IEEE Trans. Computers*, 50(4):308–321, 2001.

[2] J. C. P. Gutiérrez and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of RTSS'19*, pages 26–37, 1998.

[3] J. C. P. Gutiérrez and M. G. Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *Proceedings of RTSS'99*, pages 328–339, 1999.

[4] M. Hamdaoui and P. Ramanathan. A dynamic priority assignement technique for streams with (m, k)-firm deadlines. *IEEE Trans. Computers*, 44(12):1443–1451, 1995.

[5] Z. A. H. Hammadeh, S. Quinton, and R. Ernst. Extending typical worst-case analysis using response-time dependencies to bound deadline misses. In *Proceedings of EMSOFT'14*, pages 10:1–10:10, 2014.

[6] P. Kumar and L. Thiele. Quantifying the effect of rare timing events with settling-time and overshoot. In *Proceedings of RTSS'33*, pages 149–160, 2012.

[7] M. Moy and K. Altisen. Arrival curves for real-time calculus: The causality problem and its solutions. In *Proceedings of TACAS'10*, pages 358–372, 2010.

[8] S. Quinton, M. Hanke, and R. Ernst. Formal analysis of sporadic overload in real-time systems. In *Proceedings of DATE'12*, pages 515–520. IEEE, 2012.

[9] J. Schlatow and R. Ernst. Response-time analysis for task chains in communicating threads. In *Proceedings of RTAS'16*, 2016.

[10] W. Xu, Z. A. H. Hammadeh, A. Kröller, S. Quinton, and R. Ernst. Improved deadline miss models for real-time systems using typical worst-case analysis. In *Proceedings of ECRTS'15*, pages 247–256, 2015.