Characterizing the Effect of Deadline Misses on Time-Triggered Task Chains

Paolo Pazzaglia, *Member, IEEE*, and Martina Maggio, *Senior Member, IEEE*

***Abstract*—Modern embedded software includes complex functionalities and routines, often implemented by splitting the code across different tasks. Such tasks communicate their partial computations to their successors, forming a task chain. Traditionally, this architecture relies on the assumption of hard deadlines and timely communication. However, in actual implementations, tasks may miss their deadlines, thus affecting the propagation of their data. This article analyzes a task chain in which tasks can fail to complete their jobs according to the weakly-hard task model. We explore how missing deadlines affect chains in terms of classic latency metrics and valid data paths. Our analysis, based on mixed integer linear programming, extracts the worst-case deadline miss pattern for any given performance metric.**

***Index Terms*—Deadline misses, functional task chains, latency analysis, real-time systems, weakly-hard model.**

I. Introduction

COMMERCIAL real-time embedded applications have experienced a rapid increase of the number of components and functionality packed onto the same systems. The automotive sector is a clear example of this trend: new advanced driving assistance systems and better safety functions require the integration of a higher number of sensors and actuators, as well as increasingly complex software to be run. The implementation of these complex functions is commonly split among multiple tasks, creating *functional chains* [29]. A typical example of task chain comes from control systems. Modern controllers may require to collect inputs from sensors, filter such data, extract features, and finally produce control commands used by actuators and diagnostic monitors. All such steps are possibly managed by different tasks, often involving the connection of both new components and legacy software, the two having possibly mismatched rates [40]. As a consequence, most automotive systems are intrinsically *multirate* systems.

Tasks belonging to the same functional chain can be mapped to different computational units of a multicore processor for parallelism exploitation. Such platforms are powerful and efficient, but require a more complex design and analysis to provide correctness guarantees. In this context, analyzable communication mechanisms, such as the popular logical execution time (LET) paradigm [33], are introduced in order to preserve determinism and ensure a proper functioning.

Furthermore, such systems are often characterized by an environment where the demand for computational resources on the part of the application is not constant in time. When transient *overload conditions* occur, some task instances may not be able to complete their executions within the given deadline. These working conditions are quite common in real-world applications [3], and considering these systems as hard real-time ones would result in unnecessary pessimism. Indeed, automotive applications have been proven to be tolerant to overload conditions and missed deadlines, as long as the deadline misses are sporadic and limited in time [51], [60].

Despite this, little attention has been paid in the scientific literature to the analysis of the effect that a task's deadline misses may cause to other (functionally interconnected) tasks. A job missing its deadline is usually unable to produce its output on time, impacting the functional behavior of all the tasks that directly depend on it. This may inflate the reaction time of the task chain (e.g., from the sensing of an event to the reaction on the part of the actuators) possibly affecting the system performance [41]. In the worst case, a missed deadline for a task may even result in the entire chain skipping an output. The magnitude of such effects is, however, largely dependent on the taskset characteristics and its schedule. For example, a skipped output of a highly oversampled task may not have any significant effect on the data flow.

This article aims at assessing the impact of deadline misses in a functional chain, when at the deadline instant the instance of a task that has not completed is terminated (*killed*). Initially, we abstract from a model of how deadline misses occur in the system and analyze their effect in general on the data propagation. Then, we study effect chains that involve *weakly-hard* tasks mapped in a multicore platform, communicating with protocols based on the LET paradigm. We formulate the analysis as a mixed integer linear programming (MILP) problem and explore the effects of deadline misses in terms of common metrics for effect chains, such as input-output latency, and data age. Even when all tasks are designed to adhere with the hard real-time systems paradigm, this approach can be used as a sensitivity analysis to identify the tasks that may disrupt the system performance and to help designing better systems.

The approach is finally evaluated over the Perception RTSS 2021 Industry Challenge [44] that models an autonomous

Manuscript received 20 July 2022; accepted 26 July 2022. Date of publication 16 August 2022; date of current version 24 October 2022. This work was supported by the European Union's Horizon 2020 Research and Innovation Programme under Grant 871259 (ADMORPH – http://admorph.eu). This article was presented at the International Conference on Embedded Software (EMSOFT) 2022 and appeared as part of the ESWEEK-TCAD special issue. This article was recommended by Associate Editor A. K. Coskun. *(Corresponding author: Paolo Pazzaglia.)*

The authors are with the Department of Computer Science, Saarland University, 66123 Saarbrücken, Germany (e-mail: pazzaglia@cs.uni-saarland.de).

Digital Object Identifier 10.1109/TCAD.2022.3199146

driving application that includes a sensing pipeline and the calculation of a new control signal, which represents a good target example of emerging automotive applications.

II. Related Work

Task chains are historically classified in two distinct groups: 1) *time-triggered* (or periodic) and 2) *event-driven* [57]. In event-driven chains each task is activated by the termination of its predecessor. Event-driven chains are common in avionics [27] and robotics [12]. In time-triggered chains, tasks execute according to their own periodic behavior, and the communication occurs by shared memory value updates. Automotive systems usually employ time-triggered chains [4], following the AUTOSAR standard. We focus on time-triggered chains.

One of the first discussions on time-triggered task chains is found in [26], where the authors propose an algorithm to select the task periods while guaranteeing end-to-end latency requirements for a chain. A more rigorous timing analysis of time-triggered chains was proposed by Feiertag *et al.* [23]. This analysis introduced a detailed chain framework and its semantics and paved the way for multiple works on chain-related challenges in different configurations and with different levels of available information (e.g., [6], [21], [37], and [42]). Other works dealt with the problem of optimizing task parameters to satisfy end-to-end timing constraints for a task chain [17], [58], [65]. Recent papers targeted more complex scenarios: chains that may share one or more tasks with other chains [36], and globally asynchronous locally synchronous distributed chains [28]. Successful industrial tools, such as SymTA/S [31] now include the implementation of end-to-end delay analysis for time-triggered effect chains.

The LET paradigm was originally proposed in the Giotto framework [32], but gained traction recently in the industrial world [22], particularly in the automotive sector [29], [47]. With LET, tasks read and write data at prescribed time instants, regardless of the actual (and possibly varying) time required to complete a job's execution. LET provides time determinism and a predictable execution model, at the cost of a higher (yet fixed) input-output delay. Using LET provides system designers with a time contract, abstracting from the chosen platform and scheduler [19], and can be exploited to arbitrate memory accesses [9], [10]. LET is becoming a widely accepted candidate to enforce a causal execution order between tasks executing in different cores, applying only minimal changes to legacy code [22], [45], [49]. LET has been recently extended with the introduction of system-level LET (SL-LET) [25], where a fixed time interval is enforced also for intercore communications.

The LET semantics has also been naturally applied to task chains. A comparison of three communication protocols commonly used in automotive task chains, namely the implicit, explicit, and LET communication paradigm, was presented in [48]. Becker *et al.* [5] proposed a method to calculate the maximum data age with both implicit and LET communication models. Martinez *et al.* [47] presented an analytical characterization of the end-to-end latency of time-triggered chains, communicating through the LET model. Finally, a latency analysis that tackles the whole dependency graph for tasks under LET is presented in [39].

However, all the mentioned results consider hard-deadline task models, neglecting the case in which one or more tasks may experience deadline misses, which commonly occurs in practical applications [3]. When the number of deadline misses experienced by a task can be bounded, *weakly-hard* task models [8], [56] are often used. Extracting the weakly-hard bounds for a task requires an ad-hoc analysis. The analysis presented in [8] consists in checking the number of deadline misses of a task over time intervals that are long enough to be representative of all the possible execution conditions. The work of [59] provides an MILP-based weakly-hard analysis for real-time systems of periodic tasks with unknown offsets, and has been extended in [53] to include the case in which a job is terminated (or killed) when the deadline miss occurs. The related problem of verifying the safety of weakly-hard systems has been addressed in [24], and recently generalized in [34], [35]. A vast collection of works leverages the weakly-hard model to describe systems with overload activations, by using the typical worst-case analysis (TWCA) [55], [64] and its extension TypicalCPA [2], [38]. The combination of weakly-hard model and LET-based communication has been studied for control systems [13], [24], [43], [52], [61]. In particular, the knowledge of the maximum number of consecutive misses and hits proved to be critical to bound stability and performance of controllers [46], [63].

The papers cited above deal with either hard-real-time systems task chains, or weakly-hard independent tasks. The only notable exceptions that discuss nonhard real-time systems in combination with task chains are [30] and [16]. Hammadeh *et al.* [30] presented a TWCA approach to bound the number of deadline misses occurring to multiple *event-triggered* task chains that execute in parallel on a single processor. Conversely, we here target time-triggered chains in which each task is associated with weakly-hard constraints (for some tasks the constraints may possibly allow no misses). Choi *et al.* [16] proposed a scheduling algorithm that is based on a response-time analysis that takes into account (among other things) the possibility of missing a deadline. While [16] is the first work on time-triggered chains that analyzes the response time in the presence of deadline misses, the focus is on the scheduling side and tasks are characterized using worst-case execution times rather than weakly-hard models. To the best of our knowledge, our work is the first that specifically targets time-triggered chains composed of weakly-hard tasks, and provides a thorough characterization of task chain metrics.

III. System Model

We consider a task chain as an ordered sequence of n periodic tasks τ_i, and write it as $(\tau_i)_{i\in\mathcal{N}}$. $\mathcal{N} = \{1, \ldots, n\}$ is the set of indices of the tasks that form the chain, where the task order specifies data dependencies between the tasks, i.e., τ_i generates data used by τ_{i+1}.

We model an arbitrary task τ_i with the tuple $\tau_i = \{P_i, D_i, O_i, R_i^b, R_i^w, \ell_i^{\text{in}}, \ell_i^{\text{out}}, M_i\}$. In this tuple, P_i is the task period, D_i is the task deadline, and we assume that the task deadlines are constrained, i.e., $D_i \leq P_i$. The variable O_i

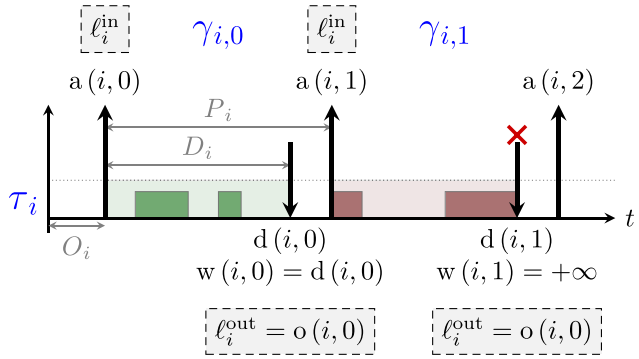

Fig. 1. Execution of a task τ_i, highlighting notation and basic definitions.

represents a bounded release offset such that $0 \leq O_i < P_i$. The values of R_i^b and R_i^w are, respectively, the best- and the worst-case response times of τ_i, and $R_i^b \leq D_i$.

Tasks communicate with one another using memory locations, also called labels. At the beginning of its execution, the task τ_i reads data from all its input labels ℓ_i^{in}. The task writes its output data into local output labels ℓ_i^{out} at its deadline. A system-level mechanism synchronizes the output data of a producer task and the input data of a consumer tasks that reads the data when it is activated, with negligible overhead. Hence, we assume that at each activation instant of τ_{i+1} the data currently available in ℓ_i^{out} (written by τ_i) is copied into ℓ_{i+1}^{in}. This synchronization mechanism can be obtained, e.g., with the LET paradigm [32], which has been proposed in different implementations [7], [9], [25], [48], [50].

Finally, M_i is a deadline miss model, that specifies how deadline misses can occur during the execution of task τ_i. The formalization presented in the following sections is independent from the deadline miss model, that we will introduce only after the general formulation is in place.

An instance of execution of a task is called *job*. We denote with $\gamma_{i,j}$ the jth job of the ith task, with $j \in \mathbb{N}$. Furthermore, for every job $\gamma_{i,j}$, we define the following functions.

1) $a(i,j) = jP_i + O_i$ is the activation time of job $\gamma_{i,j}$.
2) $d(i,j) = \text{a}(i,j) + D_i$ is the absolute deadline of $\gamma_{i,j}$.
3) $w(i,j)$ is the absolute time in which the output of job $\gamma_{i,j}$ is available in ℓ_i^{out}. If the job completes its execution before its deadline, $w(i,j) = d(i,j)$. If not, then the job is killed and $w(i,j) = +\infty$.
4) $o(i,j)$ is the output data available in ℓ_i^{out} at $d(i,j)$.

We are interested in cases in which at least one of the tasks in the chain may miss a deadline, i.e., $\exists i \in \mathcal{N} \mid R_i^w > D_i$. If a deadline miss occurs, the corresponding job is *killed* (i.e., immediately terminated) at the deadline instant and does not produce any new output data.

Fig. 1 illustrates the execution of an arbitrary task τ_i under the proposed model. After an initial release offset O_i, job $\gamma_{i,0}$ starts its execution, which is correctly completed before the deadline $d(i,0) = w(i,0)$, when the output data $o(i,0)$ is written in the label ℓ_i^{out}. The second job $\gamma_{i,1}$ does not complete its execution within the deadline $d(i,1)$ and is thus killed. The output data written in ℓ_i^{out} remains unchanged, and the writing time of the output is set to infinity $w(i,1) = +\infty$.

Tasks τ_1 and τ_n are, respectively, called *head* and *tail* of the chain. The head task works with new data at each activation, the *chain input* (e.g., by sensing data from the environment). The tail task completes the function implemented in the chain by writing its final output, called the *chain output* (e.g., to the actuators that affect the environment). The successor of task τ_i is τ_{i+1} (respectively, the predecessor of τ_{i+1} is τ_i) when $1 \leq i < n$. By construction, the task chain is *time-triggered*, i.e., the release of each job is independent from other jobs, and only follows the pattern driven by task periods and initial offsets. This model is consistent with real-world applications, including automotive ones, that are intrinsically multirate.

IV. Data Propagation

In the following, we are interested in understanding the data propagation along the chain, i.e., the *data paths*. Intuitively, a data path is a sequence of jobs of the tasks in the chain that process the chain input into a corresponding chain output. We first explore the data paths in absence of deadline misses, but—contrary to previous research contribution—we provide a formulation that is suitable also to handle jobs that do not terminate. An example highlights the difficulty emerging from the introduction of deadline misses in the analysis of the chain. The section is concluded by the definition of relevant metrics, to understand the chain properties.

A. Preliminaries

We define here the sets $\mathcal{J}$, $\mathcal{F}$, and $\mathcal{M}$, respectively, the set of all the jobs, the set of jobs that terminate their execution within the respective deadlines, and the set of jobs that miss their deadlines.

Definition 1 (Jobs): $\mathcal{J} = \{\gamma_{i,j}\}_{i \in \mathcal{N}, j \in \mathbb{N}}$.

Definition 2 (Completed and Not Completed Jobs): $\mathcal{F} = \{\gamma_{i,j} \in \mathcal{J} | w(i,j) < +\infty\}$, $\mathcal{M} = \mathcal{J} \backslash \mathcal{F}$.

Clearly, deadline misses (that intrinsically depend on how tasks are scheduled) affect the composition of the sets $\mathcal{F}$ and $\mathcal{M}$. The formalization of the theoretical concepts around data paths exploits the definition of $\mathcal{F}$, and is, therefore, valid regardless of the presence of deadline misses.

B. Data Paths

Data propagation must follow the causality principle, meaning that information is carried in a path from predecessors to successors with a strict temporal succession. We can then define the set of data paths as the set of all the job sequences $(\gamma_{i,f_i})_{i \in \mathcal{N}}$, such that the activation of the successor jobs in the sequence occurs after (or at) the deadline of their predecessors.

Definition 3 (Set of Data Paths): $\mathcal{P} = \{(\gamma_{i,f_i})_{i \in \mathcal{N}} | d(i,f_i) \leq \text{a}(i+1, f_{i+1})\}$.

The hypothesis of the negligible overhead of the communication mechanism allows the usage of $\leq$ in the formulation of data paths. This definition only enforces the causality of the execution and may include both unfinished jobs and jobs whose output has been overwritten with new data before being consumed. We can then restrict the definition to *valid paths*.

Definition 4 (Set of Valid Data Paths): $\mathcal{V} = \{(\gamma_{i,f_i})_{i \in \mathcal{N}} \in \mathcal{P} | \forall i \in \mathcal{N}, \gamma_{i,f_i} \in \mathcal{F} \wedge \forall i \in \{1, \ldots, n-1\}, \nexists \gamma_{i,c_i} \in$

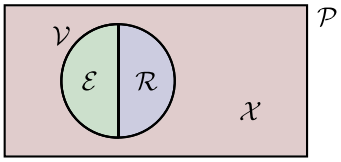

Fig. 2. Relations between the different sets of paths.

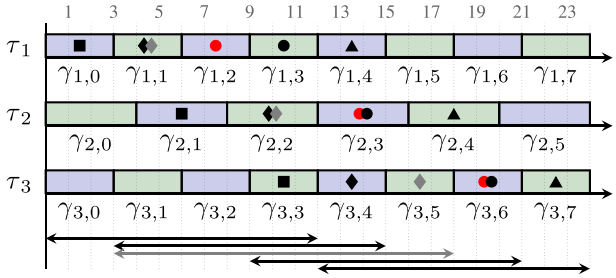

Fig. 3. Illustration of paths for Example 1.

$\mathcal{F}$ such that $a(i,f_i) < a(i,c_i) \wedge w(i,c_i) \leq a(i+1,f_{i+1}) \wedge o(i,f_i) \neq o(i,c_i)\}$.

In short, the set of valid paths includes only paths whose jobs terminated, and whose output data is not overwritten with the result of a different computation before the start of the successor job in the path. A path that does not belong to the set of valid data paths is denoted as *void*.

Definition 5 (Set of Void Data Paths): $\mathcal{X} = \mathcal{P} \backslash \mathcal{V}$.

A chain path is void if at least one of this conditions hold.

1) One or more of the jobs of the path miss a deadline.
2) The output of some of the jobs in the path is overwritten and never used, therefore, the propagation of the chain input is not completed across all tasks of the chain.

Valid paths represent meaningful data propagation and can be partitioned into two different subsets: 1) *effective* and 2) *redundant* paths. Redundant paths are paths in which the computation of some of the jobs is repeated (producing the same output that is pushed down the chain more than once).

Definition 6 (Set of Redundant Data Paths): $\mathcal{R} = \{(\gamma_{i,f_i})_{i\in\mathcal{N}} \in \mathcal{V} \,|\, \exists (\gamma_{i,c_i})_{i\in\mathcal{N}} \in \mathcal{V}$ such that $c_1 = f_1 \wedge \exists i | a(i,c_i) < a(i,f_i)\}$.

Valid nonredundant paths are effective: jobs in effective paths compute on fresh data and produce new and fresh output values, that are saved in the corresponding output labels.

Definition 7 (Set of Effective Data Paths): $\mathcal{E} = \mathcal{V} \backslash \mathcal{R}$.

Fig. 2 illustrates the relation between the different sets of paths. Deadline misses affect the membership of paths in $\mathcal{P}$ to the different sets $\mathcal{V}, \mathcal{X}, \mathcal{R}$, and $\mathcal{E}$.

Example 1: Consider a chain (τ_1, τ_2, τ_3), where $P_1 = P_3 = 3$, $P_2 = 4$ $\forall i\, P_i = D_i$ and $O_i = 0$, where $\mathcal{J} \equiv \mathcal{F}$, meaning that no task experiences deadline misses. Fig. 3 shows the first 24 time units of the execution of the tasks. In the displayed time window we highlight 6 data paths, identified with different markers. The paths marked with black square, black diamond, black circle, and black triangle are effective. The path marked with gray diamond markers is redundant, as $\gamma_{3,5}$ executes the same computation of $\gamma_{3,4}$. The path marked with red circles is void as the output data produced by $\gamma_{1,2}$ is overwritten by $\gamma_{1,3}$ before the start of $\gamma_{2,3}$. The bottom part of Fig. 3 shows the end-to-end latency of the valid paths, marking in gray the redundant path and in black the effective ones.

Imagine now that jobs can experience deadline misses, i.e., $\mathcal{M} \neq \varnothing$ and thus $\mathcal{F} \subset \mathcal{J}$.

1) If $\mathcal{M} = \{\gamma_{1,2}\}$, no changes would happen in the sets of paths, as $\gamma_{1,2}$ already belonged only to the red circle path (a void path).
2) If $\mathcal{M} = \{\gamma_{3,4}\}$, the black diamond path in Fig. 3 (that was effective) becomes void, while the gray diamond path (that was redundant) becomes effective.
3) If $\mathcal{M} = \{\gamma_{3,5}\}$, the gray diamond path (that was redundant) becomes void, but no changes occur in the chain output (which was already calculated by $\gamma_{3,4}$).
4) If $\mathcal{M} = \{\gamma_{1,3}\}$, the red circle path (that was void) becomes effective, while the black circle path is void.
5) If $\mathcal{M} = \{\gamma_{3,6}\}$, the red and black circle paths (respectively, void and effective) become void.

These are just some of the alternatives that can occur when a single job misses its deadline. Clearly, when multiple jobs (possibly of more than one task) can experience deadline misses, determining their effect is much more complex.

In order to understand the input-output relations for the chain, we can safely restrict our analysis to effective paths only, as they are responsible of carrying new information from the head to the tail task. Two paths are *distinct* when they have at least one noncommon job, i.e., $(\gamma_{i,f_i})_{i\in\mathcal{N}} \not\equiv (\gamma_{i,c_i})_{i\in\mathcal{N}} \iff \exists i \in \mathcal{N} \,|\, f_i \neq c_i$. It is simple to show that a job $\gamma_{i,j}$ cannot belong to two distinct effective paths.

Lemma 1: Given $(\gamma_{i,f_i})_{i\in\mathcal{N}} \in \mathcal{E}$ and $(\gamma_{i,c_i})_{i\in\mathcal{N}} \in \mathcal{E}$, then $(\gamma_{i,f_i})_{i\in\mathcal{N}} \not\equiv (\gamma_{i,c_i})_{i\in\mathcal{N}} \implies \forall i \in \mathcal{N}, f_i \neq c_i$.

Proof: We provide a proof of the lemma by contradiction. Suppose that $(\gamma_{i,f_i})_{i\in\mathcal{N}} \in \mathcal{E}$, $(\gamma_{i,c_i})_{i\in\mathcal{N}} \in \mathcal{E}$, $(\gamma_{i,f_i})_{i\in\mathcal{N}} \not\equiv (\gamma_{i,c_i})_{i\in\mathcal{N}}$, but also that $\exists i_a, f_{i_a} = c_{i_a}$. Since $(\gamma_{i,f_i})_{i\in\mathcal{N}} \not\equiv (\gamma_{i,c_i})_{i\in\mathcal{N}}$, it must hold that $\exists i_b \neq i_a$ such that $f_{i_b} \neq c_{i_b}$.

If $i_a = 1$, then $a\big(i_b, f_{i_b}\big) \lneqq a\big(i_b, c_{i_b}\big)$ and, therefore, it follows from Definition 6 that one of the two paths is redundant. On the contrary, if $i_a \neq 1$, then $f_1 \neq c_1$, and hence $o(1,f_1) \neq o(1,c_1)$. This means that the two effective paths propagate data from different chain inputs, thus the labels read by the shared job $\gamma_{i_a,f_{i_a}}$ must contain at the same time two different values in the corresponding effective paths. However, this cannot happen due to the communication mechanism, so one of the two paths must be void. The lemma follows. ■

From Lemma 1, we can immediately conclude that the activations of jobs of the same task belonging to two distinct effective paths follow the same temporal order.

Corollary 1: Given $(\gamma_{i,f_i})_{i\in\mathcal{N}} \in \mathcal{E}$ and $(\gamma_{i,c_i})_{i\in\mathcal{N}} \in \mathcal{E}$, then $a(1,f_1) > a(1,c_1) \implies a(i,f_i) > a(i,c_i)$ $\forall i > 1$.

This allows us to define a total order of the elements in $\mathcal{E}$. We then specify the zth effective path as π_z, and $\mathcal{E} = \{\pi_z\}_{z\in\mathbb{N}}$. For the zth effective path, we can define the function $p_z(i)$ that maps the ith task to the index of the job of the ith task that belongs to the zth effective path. We then write the generic zth effective path as $\pi_z = (\gamma_{i,p_z(i)})_{i\in\mathcal{N}}$.

Corollary 2: Given two consecutive effective paths, π_z and π_{z+1} $\forall i \in \mathcal{N}$, $p_z(i) < p_{z+1}(i)$.

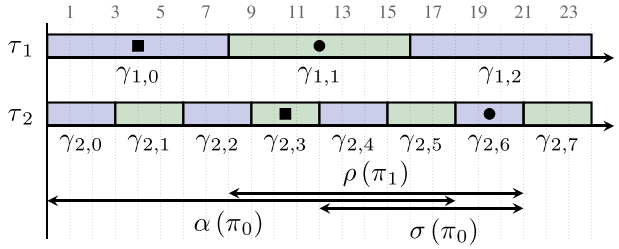

Fig. 4. Illustration of metrics for Example 3.

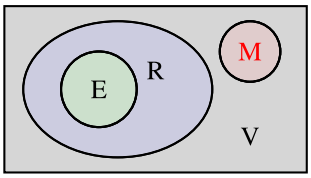

Fig. 5. Relations between the different job labels.

C. Time Window

We are interested in analyzing the chain in a generic time interval $[t_0, t_1]$. The zth effective path $\pi_z = (\gamma_{i,p_z(i)})_{i\in\mathcal{N}}$ is included in the window $[t_0, t_1]$ if $t_0 \leq a(1, p_z(1))$ and $w(n, p_z(n)) \leq t_1$. Furthermore, the inclusion is tight if $t_0 = a(1, p_z(1))$ and $w(n, p_z(n)) = t_1$. We can extend the definition to a window that contains w effective paths.

Definition 8 (Time Window With w Effective Paths): A time window $[t_0, t_1]$ contains w effective paths $\{\pi_z, \ldots, \pi_{z+w-1}\}$ if $t_0 \leq a(1, p_z(1))$ and $w(n, p_{z+w-1}(n)) \leq t_1$. The inclusion is tight if $t_0 = a(1, p_z(1))$ and $w(n, p_{z+w-1}(n)) = t_1$.

Example 2: For the chain introduced in Example 1, whose schedule is represented in Fig. 3, the time window $[t_0, t_1] = [0, 24]$ tightly contains the 4 effective paths $\{\pi_0, \pi_1, \pi_2, \pi_3\} = \{(\gamma_{1,0}, \gamma_{2,1}, \gamma_{3,3}), (\gamma_{1,1}, \gamma_{2,2}, \gamma_{3,4}), (\gamma_{1,3}, \gamma_{2,3}, \gamma_{3,6}), (\gamma_{1,4}, \gamma_{2,4}, \gamma_{3,7})\}$. The window $[t_0, t_1] = [0, 17]$ contains $\{\pi_0, \pi_1\}$.

D. Metrics

We introduce the following metrics to analyze the behavior of a single task chain.

1) Input-output latency.
2) Data age.
3) Update interval.

This selection is limited for the sake of brevity, but our approach can be easily extended also to other metrics of interest, if needed.

The *input-output latency* of an effective path π_z is the interval between the activation of the head job and the write instant of the tail job in the path. We also define the maximum input-output latency of the chain.

Definition 9 (Input-Output Latency of an Effective Path): $\rho(\pi_z) = \mathrm{w}(n, p_z(n)) - a(1, p_z(1))$.

Definition 10 (Maximum Input-Output Latency): $\overline{\rho} = \max_{z\in\mathbb{N}} \rho(\pi_z)$.

The data age of an effective path π_z is the time interval between the activation of the head job in the path and the activation of the tail job of the next effective path of the chain. We also define the maximum data age of the chain.

Definition 11 (Data Age of an Effective Path): $\alpha(\pi_z) = a(n, p_{z+1}(n)) - a(1, p_z(1))$.

Definition 12 (Maximum Data Age): $\overline{\alpha} = \max_{z\in\mathbb{N}} \alpha(\pi_z)$.

The update interval of an effective path π_z is the time interval between the output of the tail job of the path and the output of the tail job of the next effective path. We also define the maximum and minimum update intervals of the chain.

Definition 13 (Update Interval of an Effective Path): $\sigma(\pi_z) = w(n, p_{z+1}(n)) - w(n, p_z(n))$.

Definition 14 (Maximum and Minimum Update Interval): $\overline{\sigma} = \max_{z\in\mathbb{N}} \sigma(\pi_z)$ and $\underline{\sigma} = \min_{z\in\mathbb{N}} \sigma(\pi_z)$.

Example 3: Suppose we have the chain (τ_1, τ_2), where $P_1 = 8$, $P_2 = 3$, and $\forall i$, $P_i = D_i$ and $O_i = 0$. Fig. 4 shows the first 24 time units of the chain execution. We illustrate the first two effective paths π_0 (black square) and π_1 (black circle). The bottom part of the figure highlights the input-output latency of path π_1 and the data age and update interval of path π_0.

V. Impact of Deadline Misses

Under the hypothesis of hard deadlines and LET-based communication, one can automatically identify effective and redundant paths simply from the task characterization, i.e., periods, deadlines, and initial release offsets [47]. Example 1 listed a brief collection of potential consequences of killed jobs and shows that deadline misses influence the set of effective paths. In fact, when $\mathcal{J} \not\equiv \mathcal{F}$, the actual pattern of deadline hits and misses drives the pattern of task outputs, thus impacting the behavior of data paths.

Section V-A introduces the first part of our analysis formulation, which is independent of the deadline miss model used. This means that the concepts introduced are valid for the space of all possible deadline miss patterns (and also for unconstrained misses). In Section V-B, on the contrary, we constrain the deadline-miss model to being the weakly-hard model [8], and the concepts introduced here are valid for all the patterns that constrain the maximum number of misses experienced by a sequence of consecutive job activations.

A. Basic Characterization

For simplicity, we flag jobs based on their set memberships, i.e., on them being members of paths included in the sets $\mathcal{X}$, $\mathcal{E}$, and $\mathcal{R}$, or on them being members of $\mathcal{M}$. We use a simple algorithm described as follows. We start by flagging every job as void (V), as in principle we could create void paths including any job in the schedule. If a job belongs to the set $\mathcal{M}$, we change its flag to indicate it missed its deadline (M). If a job with flag (V) belongs to at least one path in $\mathcal{R}$, we upgrade its flag to redundant (R). Then, if the job belongs to an effective path, i.e., a path in $\mathcal{E}$, we upgrade its flag as effective (E). Fig. 5 summarizes the relations between the flag sets, using the flag to represent all jobs that receive it.

We now characterize the jobs in a given time window, from the perspective of an arbitrary task that can experience deadline misses. Between two effective jobs of $\tau_i - \gamma_{i,p_z(i)}$ and $\gamma_{i,p_{z+1}(i)}$ – a number (greater or equal zero) of jobs of the same task are activated. Any of these jobs may miss its deadline (M), or is necessarily flagged either as redundant (R) or

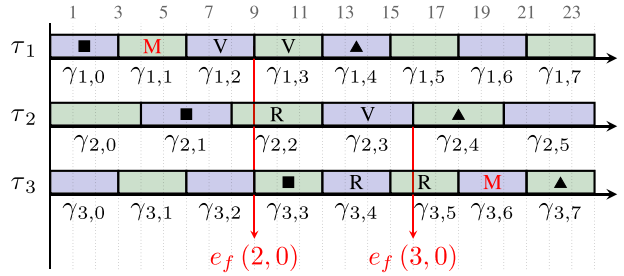

Fig. 6. Illustration of first input overwrite $e_f(2,0)$ for τ_2 and $e_f(3,0)$ for τ_3 between the effective paths π_0 (black squares) and π_1 (black triangle).

void (V). Considering the jobs in $\mathcal{F}$, whether a job is flagged redundant or void depends on the presence or absence of new input data at its activation instant. In particular, it is possible to define a time instant, which is here denoted with *first input overwrite* $e_f(i,z)$, where new data is available for the task τ_i to read, after completing its job $\gamma_{i,p_z(i)}$.

Definition 15 (First Input Overwrite):

$$e_f(i,z) = \begin{cases} a(i, p_z(i)+1), & i = 1 \\ w\big(i-1, p_z(i-1)+x_{i,z}\big), & i \neq 1 \end{cases}$$

where $x_{i,z} \geq 1$ is the minimum integer value that satisfies the condition $o(i-1, p_z(i-1)) \neq o\big(i-1, p_z(i-1)+x_{i,z}\big)$.

From Definition 15, it immediately follows that $\forall i > 1$, $e_f(i,z) \leq w(i-1, p_{z+1}(i-1))$. The different definition given for the head task reflects the property (of the task chain model) that the head task can continuously read new data from the environment.

Any completing job of τ_i activated after job $\gamma_{i,p_z(i)}$ and before $e_f(i,z)$ is redundant. Any completing job of τ_i activated after $e_f(i,z)$ and before job $\gamma_{i,p_{z+1}(i)}$ is void. We can then immediately conclude that any job of the head task τ_1, between the ones in two consecutive effective paths, $\gamma_{1,p_z(1)}$ and $\gamma_{1,p_{z+1}(1)}$, is either void (V) or misses its deadline (M). The dual argument for the tail task τ_n is that if any job exists between the two consecutive effective jobs $\gamma_{n,p_z(n)}$ and $\gamma_{n,p_{z+1}(n)}$, then they are either redundant (R) or missed their deadline (M). Indeed, by construction, the tail task cannot have jobs that correctly complete their execution but are not part of either an effective or a redundant path.

Example 4: We illustrate the concept of first input overwrite introducing some deadline misses in the chain from Example 1. Fig. 6 shows the effective paths π_0 (black squares) and π_1 (black triangle). Since $\gamma_{1,1}$ misses its deadline (M), the first input overwrite for τ_2 occurs at the end of $\gamma_{1,2}$. Job $\gamma_{2,2}$ is redundant, $\gamma_{2,3}$ is void, and $\gamma_{2,4}$ is effective. Note also that, if $\gamma_{3,6}$ would have not missed its deadline, i.e., if $\gamma_{3,6} \in \mathcal{F}$, then $(\gamma_{1,3}, \gamma_{2,3}, \gamma_{3,6})$ would have been the effective path π_1. However, $e_f(3,0)$ would not have changed.

At this point, to proceed with the analysis, we need to introduce a model M_i for how deadline misses may occur during the execution of task τ_i. Commonly used models are either probabilistic or deterministic. Given a stochastic model for execution times [18] and a scheduling algorithm, it is possible to extract a probabilistic model of deadline misses [11], [14], [15], [20], [62]. Such a model would allow us to characterize a task chain in the probabilistic domain.

Deterministic models constrain the sequence of job outcomes (deadline hits and deadline misses). One such model is the weakly-hard (m,k) model [8], [56] that states that in every sliding window containing k task activations, the task may miss at most m deadlines. Deterministic models provide worst-case guarantees, even though these guarantees may be conservative with respect to their probabilistic counterparts.

B. Introducing Weakly-Hard Deadline Misses

In this article, we aim at deriving worst-case metrics for the task chain, and hence we build upon the weakly-hard (m,k) deadline miss model. Specifically, we use $\mathrm{M}_i = (m_i, k_i)$, introducing a weakly-hard constraint per task. Setting $m_i = 0$ implies that the task τ_i is a hard real-time task.

The miss model M_i introduces constraints in the job outcome pattern, and, therefore, limits the deadline misses that can be experienced by τ_i. The *configuration* of the worst-case pattern of deadline hits and misses for τ_i with respect to a given metric (see Section IV-D) clearly depends on such task constraints. However, the worst-case pattern from the data propagation point of view is not necessarily the one where each task misses as many deadlines as its constraint allows for. With respect to the data propagation, the position of deadline misses is in fact at least equally important (if not more) than the number of experienced misses. For example, missing the deadline of a redundant job does not disrupt the natural data flow in the chain as the computation would not change the output data written by the job. On the other hand, the data flow is affected by multiple consecutive deadline misses if the burst of misses starts with a job that would have carried new data. When the successive jobs (of the same task) miss their deadlines, they also prevent the propagation of this new data to the successor task. Additionally, even a small number of deadline misses experienced by tasks with large periods may have a greater impact compared to a larger number of deadline misses experienced by oversampled tasks.

In general, the worst case for data propagation occurs when τ_i experiences deadline misses (if any) for jobs activated after a new input is available for the task. This property is formalized in the following Lemma.

Lemma 2: Given an arbitrary chain task τ_i that satisfies a weakly-hard constraint (m_i, k_i), the distance between two consecutive effective jobs $\gamma_{i,p_z(i)}$ and $\gamma_{i,p_{z+1}(i)}$ is maximized when all possible deadline misses experienced by τ_i occur in a combination of jobs of τ_i activated right after $e_f(i,z)$ and right after $w(i-1, p_{z+1}(i-1))$.

The lemma above rests on the fact that if the jobs activated right after $e_f(i,z)$ miss their deadline (and are thus killed), the output of the task τ_i remains unchanged despite the presence of new data that τ_i should process. Similarly, missing deadlines right after $w(i-1, p_{z+1}(i-1))$ delays the successful completion of the effective job of the $(z+1)$th effective path, thus delaying the path output. Intuitively, this excludes jobs that would either be redundant (repeating the computation done with previously processed data) or void (whose output data is overwritten before being used in a future effective path). We can then restrict to finding patterns with the specific shape shown in Fig. 7, where a number of consecutive

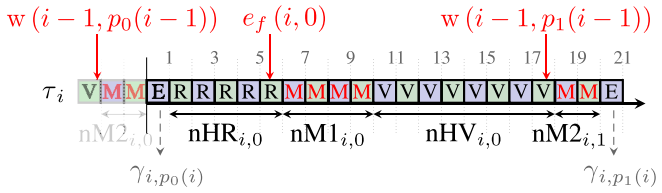

Fig. 7. Illustration of the shape of the worst-case pattern of deadline hits and misses with respect to data propagation.

deadline misses occurs in either or both these two disrupting positions (note that the two positions may coincide since $e_f(i,z) \leq w(i-1, p_{z+1}(i-1))$).

To find how many misses (introduced in the positions above) maximize or minimize a chosen metric, we specify an optimization problem, which must also take into account the (m,k) constraints. The solution of the optimization problem is the actual worst-case pattern of job outcomes for each task (with respect to the chosen metric). Depending on the metric, the pattern of missed deadlines may vary. The solver finds the worst-case by determining the value of some optimization variables, introduced in Section VI-A, and illustrated in Fig. 7. Any of these variables could in principle be zero.

Remark 1: If a finite sequence of job outcomes (hits and misses) satisfies a weakly-hard constraint (m,k), adding to the sequence a prefix and/or a suffix containing only deadline hits does not alter the satisfaction of the (m,k) constraint.

This means that, to check that the entire pattern satisfies the weakly-hard constraint M_i of task τ_i, we can restrict to verifying only those sequences of job outcomes that start with a deadline miss and end with a deadline miss. For the piece of schedule shown in Fig. 7, this means checking the following three sequences.

1) From the first miss after $w(i-1, p_0(i-1))$ to the last miss after $e_f(i,0)$.
2) From the first miss after $e_f(i,0)$ to the last miss after $w(i-1, p_1(i-1))$.
3) From the first miss after $w(i-1, p_0(i-1))$ to the last miss after $w(i-1, p_1(i-1))$.

These sequences can be identified programmatically given a time interval. We denote the set of such sequences for task τ_i with $\mathcal{S}(i)$. The cardinality of $\mathcal{S}(i)$ increase with the number of paths w in the time window, and is equal to $2w^2 - w$.

VI. Optimization Problem

This section presents an MILP formulation that extracts the worst-case metrics for a given task chain. We set up an optimization problem that takes as input the properties of the tasks in a chain and a value $w > 1$ representing the number of consecutive effective paths to be considered in the analysis. The problem targets a generic time window, $[t_0, t_1]$, which is set to (tightly) include w effective paths. From Definition 8, $t_0 = a(1, p_0(1))$ and $t_1 = w(n, p_{w-1}(n))$. Without loss of generality from the perspective of the solver, we assume that $t_0 = 0$, while t_1 is by construction an optimization variable, as it must coincide with the write instant of the tail task of the effective path with index $w-1$. The optimization problem returns the schedule that produces w consecutive effective paths, such that it optimizes (maximizing or minimizing) a given target metric, and where each task satisfies its deadline miss constraint $\mathrm{M}_i = (m_i, k_i)$.

A. Optimization Variables

The variables of the MILP problem are defined either natural (in $\mathbb{N}$) or real (in $\mathbb{R}$).

1) *Relative Offset:* For each task τ_i, $\mathrm{OF}_i \in \mathbb{R} \wedge \mathrm{OF}_i \geq 0$ is the time interval between t_0 and the release of the earliest job of τ_i after t_0.[1]
2) *Index of Effective Job:* For each effective path π_z and task τ_i, $\mathrm{iE}_{i,z} \in \mathbb{N}$ denotes the index $p_z(i)$. Without loss of generality, the job of τ_i that starts at $t_0 + \mathrm{OF}_i$ is assigned the index 0.

We introduce also variables to count jobs between effective paths, i.e., for each task τ_i and for each effective path π_z:

1) *Number of R Jobs:* $\mathrm{nHR}_{i,z} \in \mathbb{N}$ is the number of redundant jobs of task τ_i between the output produced by $\gamma_{i,p_z(i)}$ and the instant of its first input overwrite, i.e., in the time interval $[w(i,p_z(i)), e_f(i,z))$. These jobs read the same input of $\gamma_{i,p_z(i)}$ and repeat the computation performed by the previous effective job.
2) *Number of M Jobs After the First Input Overwrite:* $\mathrm{nM1}_{i,z} \in \mathbb{N}$ is the number of jobs of τ_i released in the interval $[e_f(i,z), \mathrm{w}(i-1, p_{z+1}(i-1)))$. These jobs have a new input to read, but *miss* their deadlines.
3) *Number of V Jobs:* $\mathrm{nHV}_{i,z} \in \mathbb{N}$ is the number of jobs of τ_i released in the interval $[e_f(i,z) + \mathrm{nM1}_{i,z} \cdot P_i, w(i-1, p_{z+1}(i-1)))$. These jobs read an input value that has not yet been propagated to the successor task, and complete their execution within their deadlines. However, they do not belong to any valid path.
4) *Number of M Jobs After the Next Effective Path Input Propagation:* $\mathrm{nM2}_{i,z} \in \mathbb{N}$ is the number of jobs of τ_i released in the interval $[w(i-1, p_z(i-1)), a(i, p_z(i)))$. These jobs could process the next input that belongs to an effective path, but miss their deadlines.

Finally, the optimization problem relies also on the following auxiliary boolean variables (in $\mathbb{B}$).

1) *Existence of V Jobs:* For each effective path π_z and each task τ_i, $\mathrm{bHV}_{i,z} \in \mathbb{B}$ is equal to 1 if $\mathrm{nHV}_{i,z} > 0$, and 0 otherwise.
2) *Length of sequence exceeds k_i:* For each task τ_i and each sequence $s \in \mathcal{S}(i)$, $\mathrm{bSK}_{i,s} \in \mathbb{B}$ is equal to 1 if the number of jobs in s is greater than k_i; and 0, otherwise.

B. Constraints

This section explores the definition of constraints in the MILP formulation, which are used to either exclude unfeasible situations (e.g., tasks not respecting their weakly-hard constraints) or speed up the solution finding (e.g., excluding

[1]The time interval $[t_0, t_1]$ is arbitrary, i.e., t_0 does not necessarily coincide with the initial release instant of the whole task schedule. The choice of OF_i being a real variable is done for the sake of runtime efficiency, and under the hypothesis that the starting release offset O_i is bounded, but possibly variable. If, on the contrary, the initial offset O_i of all tasks τ_i is fixed, then OF_i can be restricted to assume a limited set of values; e.g., [54, Lemma 3].

patterns that are not worst-case patterns). Formal proofs are presented only for the less obvious constraints.

Basic Constraints: By definition, the window of interest $[t_0, t_1]$ starts with the release of $\gamma_{1,p_0(1)}$, i.e., the job of τ_1 that belongs to π_0. Thus, the relative offset of τ_1, with respect to t_0, is necessarily equal to zero. This is enforced as follows.

Constraint 1: For task τ_1, $\text{iE}_{1,0} = 0$ and $\text{OF}_1 = 0$.

For $i > 1$, τ_i may start with an offset (with respect to t_0) in $[0, P_i)$.

Constraint 2: For each task τ_i, with $i > 1$, $0 \leq \text{OF}_i < P_i$.

As discussed in Section V-A, the head task has no redundant jobs and, dually, all hit jobs of the tail task are either effective or redundant.

Constraint 3: For each effective path π_z, $\text{nHR}_{1,z} = 0$, $\text{nM2}_{1,z} = 0$ and $\text{nHV}_{n,z} = 0$.

Then, we enforce the definition of $\text{bHV}_{i,z}$, which checks if there exist void (completed) jobs of τ_i between $\gamma_{i,p_z(i)}$ and $\gamma_{i,p_{z+1}(i)}$, with a *big-M* formulation as follows.

Constraint 4: For each π_z, for each τ_i, $\text{nHV}_{i,z} \leq \text{bHV}_{i,z} \cdot \text{M}$ and $\text{nHV}_{i,z} \geq \text{bHV}_{i,z}$, where M is a sufficiently-large positive constant value to represent infinity.

Proof: If $\text{nHV}_{i,z} = 0$, the first inequality is *inactive* (it holds for both $\text{bHV}_{i,z} = 0$ and $\text{bHV}_{i,z} = 1$), while the second one is satisfied only with $\text{bHV}_{i,z} = 0$. Conversely, if $\text{nHV}_{i,z} > 0$, then the first inequality is satisfied only when $\text{bHV}_{i,z} = 1$, while the second one is inactive. The constraint follows. ■

Scheduling Paths Rules: Following the definition of effective paths, $\text{a}(i, p_z(i))$ occurs after (or at) $w(i-1, p_z(i-1))$, but before $\text{w}(i-1, p_{z+1}(i-1))$.

Constraint 5: For each π_z, for each τ_i, with $i > 1$,

$$\begin{aligned}&\text{OF}_i + \text{iE}_{i,z} \cdot P_i \geq \text{OF}_{i-1} + \text{iE}_{i-1,z} \cdot P_{i-1} + D_{i-1}\\&\text{OF}_i + \text{iE}_{i,z} \cdot P_i < \text{OF}_{i-1} + \text{iE}_{i-1,z+1} \cdot P_{i-1} + D_{i-1}.\end{aligned}$$

Additionally, $\text{a}(i, p_z(i))$ must occur before $e_f(i, z)$. Notice here that in the absence of void jobs of the predecessor task, i.e., if $\text{bHV}_{i-1,z} = 0$, then $e_f(i, z) = \text{w}(i-1, p_{z+1}(i-1))$ (a case already covered in Constraint 5), therefore, we can write a new constraint only to handle the case of $\text{bHV}_{i-1,z} = 1$.

Constraint 6: For each π_z, for each τ_i, with $i > 1$

$$\begin{aligned}\text{OF}_i + \text{iE}_{i,z} \cdot P_i < \big(1 - \text{bHV}_{i-1,z}\big) \cdot \text{M} + \text{OF}_{i-1}\\ + \big(\text{iE}_{i-1,z} + \text{nHR}_{i-1,z} + \text{nM1}_{i-1,z} + 1\big) \cdot P_{i-1} + D_{i-1}.\end{aligned}$$

Then, the activation of the last redundant job of τ_i after $\gamma_{i,p_z(i)}$ must occur before the completion of the first job of τ_{i-1} that successfully completes and produces a different output. This property is encoded as follows.

Constraint 7: For each π_z, for each τ_i, with $i > 1$

$$\begin{aligned}&\text{OF}_i + \big(\text{iE}_{i,z} + \text{nHR}_{i,z}\big) \cdot P_i < \text{OF}_{i-1} + \big(\text{iE}_{i-1,z} + \text{nHR}_{i-1,z}\\&\quad + \text{nM1}_{i-1,z} + 1\big) \cdot P_{i-1} + D_{i-1} + \big(1 - \text{bHV}_{i-1,z}\big) \cdot \text{M}\\&\text{OF}_i + \big(\text{iE}_{i,z} + \text{nHR}_{i,z}\big) \cdot P_i\\&\quad < \text{OF}_{i-1} + \text{iE}_{i-1,z+1} \cdot P_{i-1} + D_{i-1} + \text{bHV}_{i-1,z} \cdot \text{M}.\end{aligned}$$

Proof: If there exists any void job of τ_{i-1} between $w(i-1, p_z(i-1))$ and $a(i-1, p_{z+1}(i-1))$, then $\text{bHV}_{i-1,z} = 1$ and only the first inequality is active, where the right hand side represents the output instant of the first void job after the (possible) redundant jobs and the deadline misses following $e_f(i, z)$. Conversely, if no void jobs exist in that interval, then the first job of τ_{i-1} that completes its execution after the redundant ones is exactly the effective job of π_{z+1}. Thus, $\text{bHV}_{i-1,z} = 0$ and only the second inequality is active. The constraint follows. ■

Then, we enforce that any job of τ_i that occurs between $\text{w}(i-1, p_z(i-1))$ and $\text{a}(i, p_z(i))$ must miss their deadlines.

Constraint 8: For each π_z, for each τ_i, with $i > 1$

$$\begin{aligned}&\text{nM2}_{i,z} > \big(\text{OF}_i + \text{iE}_{i,z} \cdot P_i - OE_{z,i-1}\big)/P_i - 1\\&\text{nM2}_{i,z} \leq \big(\text{OF}_i + \text{iE}_{i,z} \cdot P_i - OE_{z,i-1}\big)/P_i\end{aligned}$$

where $OE_{z,i-1} = (\text{OF}_{i-1} + \text{iE}_{i-1,z} \cdot P_{i-1} + D_{i-1})$.

Finally, we calculate the index $p_{z+1}(i)$ of the job of task τ_i that belongs to the effective path π_{z+1} by means of the index of $p_z(i)$, plus the variables counting the number of jobs in between, plus 1.

Constraint 9: For each π_z, with $z < w$, for each τ_i

$$\text{iE}_{i,z+1} = \text{iE}_{i,z} + \text{nHR}_{i,z} + \text{nM1}_{i,z} + \text{nHV}_{i,z} + \text{nM2}_{i,z+1} + 1.$$

Weakly Hard Constraints: After enforcing scheduling properties, we deal with the weakly-hard constraints. First, from the definition of the weakly-hard constraint $\text{M}_i = (m_i, k_i)$, a task τ_i cannot miss more than m_i deadlines every k_i activations, which also means that it cannot miss more than m_i *consecutive* deadlines. This is enforced with the following constraint.

Constraint 10: For each π_z, for each τ_i, $\text{nM1}_{i,z} \leq m_i$, and $\text{nM2}_{i,z} \leq m_i$.

If there are no void jobs between $\gamma_{i,p_z(i)}$ and $\gamma_{i,p_{z+1}(i)}$, the variables $\text{nM1}_{i,z}$ and $\text{nM2}_{i,z}$ correspond to two subsequences of missed jobs occurring alongside. Thus, the constraint that the task cannot miss more than m_i consecutive deadlines must be enforced for the whole sequence, as follows.

Constraint 11: For each π_z, for each τ_i, $\text{nM1}_{i,z} + \text{nM2}_{i,z+1} \leq m_i + \text{bHV}_{i,z} \cdot \text{M}$.

Then, we must also enforce the (m_i, k_i) constraint for all the sequences in $\mathcal{S}(i)$. First, for any sequence $s \in \mathcal{S}(i)$, we enforce the definition of the boolean variable $\text{bSK}_{i,s}$, which determines if the sequence is longer than k_i elements.

Constraint 12: For each task τ_i, for each sequence $s \in \mathcal{S}(i)$

$$\begin{aligned}&\text{nJS}_{i,s} > k_i - \big(1 - \text{bSK}_{i,s}\big) \cdot \text{M}\\&\text{nJS}_{i,s} \leq k_i + \text{bSK}_{i,s} \cdot \text{M}\end{aligned}$$

where $\text{nJS}_{i,s} \in \mathbb{N}$ is an auxiliary variable properly defined to count the number of jobs included in s.

The auxiliary variable $\text{nJS}_{i,s}$ is computed as the sum of all variables of the MILP formulation associated to the jobs in s. In the example of Fig. 7, the sequence between time instants 6 and 20 will have $\text{nJS}_{i,s} = \text{nM1}_{i,z} + \text{nHV}_{i,z} + \text{nM2}_{i,z+1}$. The general formulation of $\text{nJS}_{i,s}$ is omitted here for brevity.

Then, we can verify the (m_i, k_i) constraint for all the sequences in $\mathcal{S}(i)$ as follows.

Constraint 13: For each task τ_i, for each sequence $s \in \mathcal{S}(i)$

$$\begin{aligned}&\text{nMS}_{i,s} \leq m_i + \text{bSK}_{i,s} \cdot \text{M}\\&\text{nJS}_{i,s} - \text{nMS}_{i,s} \geq k_i - m_i - \big(1 - \text{bSK}_{i,s}\big) \cdot \text{M}\end{aligned}$$

where $\text{nMS}_{i,s} \in \mathbb{N}$ is an auxiliary variable properly defined to count the number of deadline misses of τ_i in s.

As an example, for the sequence of Fig. 7 between time instants 6 and 20, $\text{nMS}_{i,s} = \text{nM1}_{i,z} + \text{nM2}_{i,z+1}$. As for $\text{nJS}_{i,s}$, the general formulation of $\text{nMS}_{i,s}$ is omitted for brevity.

The first inequality of Constraint 13 is active if the sequence is not longer than k_i. In such a case, we simply need to check that the number of misses in the sequence does not exceed the value m_i allowed by the weakly-hard constraint of τ_i. This check is both necessary and sufficient. The second inequality is active for sequences that are longer than k_i jobs. In this case, checking that the sequence has no more than m_i misses may be too conservative. Thus, we turn the problem into verifying that the number of hits in the sequence is at least $k_i - m_i$. On its own, this check is necessary but not sufficient. However, the combination of all checks for all the subsequences in $\mathcal{S}(i)$ is both necessary and sufficient to ensure the weakly-hard constraint satisfaction.

C. Objective Functions

To calculate the worst-case metrics of Section IV-D (Definitions 10, 12, and 14), it is sufficient to consider 2 effective paths in the MILP formulation, i.e., a window containing π_0 and π_1. The solver will then find the pattern of the jobs that maximize (or minimize) the chosen metric.

Maximise Input-Output Latency: The input-output latency $\rho(\pi_0)$ of the effective path π_0 is computed as the interval between $\text{a}(1, p_0(1))$ and $\text{w}(n, p_0(n))$ By recalling that $t_0 = \text{a}(1, p_0(1)) = 0$, the latency is maximized in the MILP formulation with the following function:

$$\overline{\rho} = \max\left\{\text{OF}_n + \text{iE}_{n,0} \cdot P_n + D_n\right\}.$$

Maximize Data-Age: The data-age $\alpha(\pi_0)$ of the effective path π_0 is computed as the interval between $\text{a}(1, p_0(1))$ and $\text{a}(n, p_1(n))$, and is maximized in the MILP formulation as

$$\overline{\alpha} = \max\left\{\text{OF}_n + \text{iE}_{n,1} \cdot P_n\right\}.$$

Maximize or Minimize Update Interval: The update interval $\sigma(\pi_0)$ of the effective path π_0 is computed as the interval between $\text{w}(n, p_0(n))$ and $\text{w}(n, p_1(n))$. Since both the jobs have the same deadline, it can then be simplified as the interval between their activation instants, i.e., in MILP notation

$$\begin{aligned}\overline{\sigma} &= \max\left\{\left(\text{iE}_{n,1} - \text{iE}_{n,0}\right) \cdot P_n\right\} \\ \underline{\sigma} &= \min\left\{\left(\text{iE}_{n,1} - \text{iE}_{n,0}\right) \cdot P_n\right\}.\end{aligned}$$

VII. Experimental Evaluation

This section describes our experimental evaluation, where we coded the optimization problem of Section VI in C++ using the CPLEX community edition [1].[2] The evaluation is based on the RTSS 2021 Industry Challenge, by Perceptin [44]. The challenge aim was to analyze an autonomous driving application, composed of 12 tasks, $\{\tau_1, \ldots, \tau_{12}\}$, communicating with one another. The architectural setup is precisely the one of this article, where tasks exchange information using labels in memory areas. We consider each task to be periodic.

[2]The code to reproduce the results presented in this article is available at https://github.com/PaoloPazzaglia/ChainMiss.

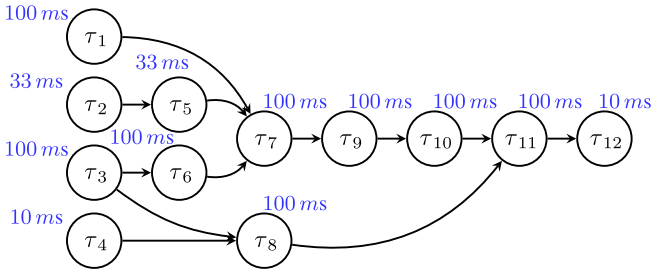

Fig. 8. Autonomous driving application taskset and chains from [44].

Fig. 8 is a schematic representation of the chain model of the Challenge. Nodes represent the application tasks while arrows indicate data exchange. The task periods are indicated near the task node, and $\forall i\, P_i = D_i$. Tasks τ_1, τ_2, τ_3, and τ_4 model sensors measurements (respectively, a radar, a camera, a lidar, and a navigation system). Task τ_5 and τ_6 handle 2-D and 3-D perception, while task τ_7 implements a sensor fusion algorithm and task τ_8 is dedicated to localization. Finally, tasks τ_9, τ_{10}, τ_{11}, and τ_{12} are, respectively, in charge of tracking, prediction, planning, and control. The application can be partitioned in 5 distinct task chains: $\text{chain}_1 = (\tau_1, \tau_7, \tau_9, \tau_{10}, \tau_{11}, \tau_{12})$, $\text{chain}_2 = (\tau_2, \tau_5, \tau_7, \tau_9, \tau_{10}, \tau_{11}, \tau_{12})$, $\text{chain}_3 = (\tau_3, \tau_6, \tau_7, \tau_9, \tau_{10}, \tau_{11}, \tau_{12})$, $\text{chain}_4 = (\tau_3, \tau_8, \tau_{11}, \tau_{12})$, and $\text{chain}_5 = (\tau_4, \tau_8, \tau_{11}, \tau_{12})$.

We now perform experiments with different tasks missing deadlines and show the impact of deadline misses on the metrics defined in Section IV-D. Initially, we perform experiments with only a single task missing deadlines. We selected τ_{11} (the planning task) and τ_{12} (the control task) to be subject to deadline misses. This is because:

1) they are common to all the chains, allowing us to evaluate the effect of introducing deadline misses on different data paths,
2) planning and control tasks may run complex algorithms, and hence may require a significant computational capacity and may be more susceptible to delays.

Finally, we perform tests where each task possibly miss deadlines. Computing one data point in each of the following tests took less than a second to run.

A. Planning Task (τ_{11}) Missing Deadlines

In the first experiment we set $m_i = 0$ for all tasks except τ_{11}. We select $k_{11} = 50$ and $m_{11} \in \{0, 1, \ldots, 30\}$. Similar results are obtained with different values of k_{11}, omitted for brevity. We first calculate the maximum input-output latency $\overline{\rho}_x$ for the five chains without any deadline miss, i.e., $m_{11} = 0$, obtaining (in *ms*) $\overline{\rho}_1 \approx 920$, $\overline{\rho}_2 \approx 852$, $\overline{\rho}_3 \approx 1120$, $\overline{\rho}_4 \approx 520$, $\overline{\rho}_5 \approx 340$.[3] These results, obtained with the MILP formulation of Section VI, are coherent with the ones obtainable with standard algorithms for hard-deadline task chains [17].

By varying m_{11} no change occurs in the maximum input-output latency $\overline{\rho}_x$ of the five chains. While counterintuitive,

[3]The numbers are approximately equal to the values indicated in the text. The approximation is due to the release offsets in the solution being real numbers. The obtained values are typically of the form $x - \varepsilon$ where x is the worst-case solution and ε is the resolution of the solver tolerance.

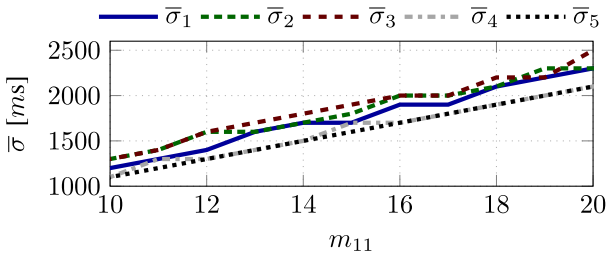

Fig. 9. Maximum update interval $\overline{\sigma}_x$ for the 5 chains of the application shown in Fig. 8, when $\mathrm{M}_{11} = (m_{11}, 50)$. Detail with $m_{11} \in [10, 20]$.

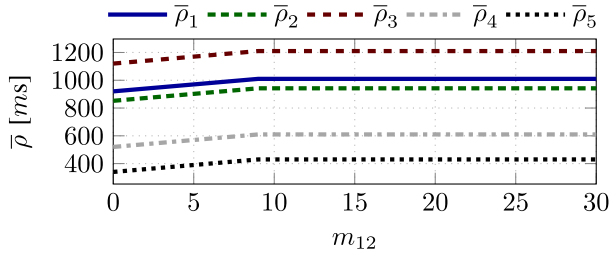

Fig. 10. Maximum input-output latency $\overline{\rho}_x$ for the 5 chains of the application shown in Fig. 8, when $\mathrm{M}_{12} = (m_{12}, 50)$.

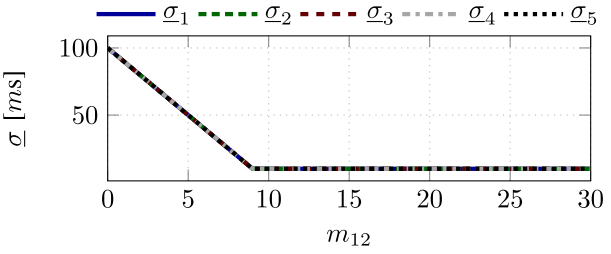

Fig. 11. Minimum update interval $\underline{\sigma}_x$ for the 5 chains of the application shown in Fig. 8, when $\mathrm{M}_{11} = (m_{12}, 50)$.

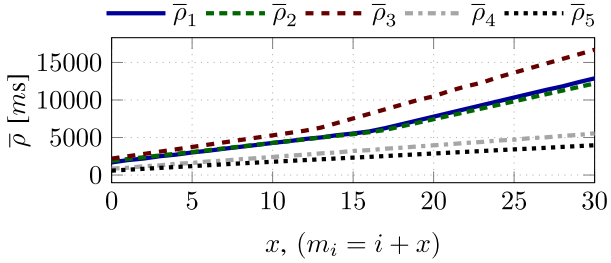

Fig. 12. Maximum input-output latency $\overline{\rho}_x$ for the 5 chains of the application shown in Fig. 8, when $\mathrm{M}_i = (i + x, 50)$.

this simply says that $\overline{\rho}_x$ is dominated in this case by other factors, such as the schedule interleaving. When task τ_{11} is finally ready to complete its execution after a (potentially long) string of misses, there is always fresher data coming from the predecessors, and hence the maximum input-output latency does not grow with m_{11}, i.e., with the misses of τ_{11}. However, this does not mean that the chains are unaffected by deadline misses of task τ_{11}. The maximum data age $\overline{\alpha}$ increases linearly with m_{11} for all five chains, while the maximum update interval $\overline{\sigma}$ (Fig. 9) shows a more irregular behavior, but still monotonic with m_{11}. The minimum update interval $\underline{\sigma}$ is 100 ms for all the chains with all possible configurations. This reflects the best possible alignment of tasks, which is dominated by the period of the slowest task in the chain.

From a chain perspective, it is interesting to compare these metrics. The maximum input-output latency, despite being unaffected, gives us an important information about the propagation of data in the chains, occurring with different rates in the application. The maximum data age tells us how long an output produced by the chain is going to affect the environment. The update interval tells us that a control signal may be applied to the controlled plant for a time that varies, e.g., for chain_1, between 100 ms and 3300 ms. Fig. 9 also shows that sometimes missing one more deadline does not necessarily mean an increase in maximum update interval.

B. Control Task (τ_{12}) Missing Deadlines

In the second experiment we set $m_i = 0$ for all the tasks except τ_{12}. We select $k_{12} = 50$ and vary $m_{12} \in \{0, 1, \ldots, 30\}$. Differently from the previous case, we found that also the maximum input-output latency $\overline{\rho}$ (Fig. 10) increases with the number of deadline misses, but only while $m_{12} < 10$. This can be explained by noting that, when $m_{12} \geq 10$, a new job of task τ_{11} completes before the activation of the next successful job of τ_{12}, providing a fresher output from the chain. Fig. 11 shows the minimum update interval $\underline{\sigma}$ for the five chains. Similarly to $\overline{\rho}$, $\underline{\sigma}$ varies only when $m_{12} < 10$ because those misses cause a job that would have been redundant for the tail task to become effective, and hence shorten the minimum update interval in some specific conditions.

C. All Tasks Missing Deadlines

In this last experiment with the Perception model we consider each task in the chain to be subject to a different weakly-hard constraint. In detail, for each test and for each task τ_i we set $m_i = i + x$, where i is equal to the task index and x is a variable in the interval $x \in \{0, 1, \ldots, 30\}$. Again, for simplicity we select the window size of all tasks as $k_i = 50$.

Fig. 12 shows the maximum input-output latency $\overline{\rho}_x$ for each chain_x. When comparing Figs. 10 and 12, it is evident that the combined effects of deadline misses in the tasks causes a higher level of disruption. The increase rate is, however, not linear with the number of worst-case misses, but irregular, albeit monotonic. In particular, the first three chains (which are the longer ones) show a sharp increase when x assumes values around 15. This is due to the possibility of more disruptive combinations of job outcomes when multiple deadline misses are allowed across multiple different tasks.

D. Scalability Evaluation

To respond to the question of whether the analysis scale when the number of tasks in the chain grows, we generated random task chains and calculated the time it takes for the solver to extract a chain worst-case metric. We vary the number of tasks in the chain in the set $n \in \{5, 10, 15, 20, 25, 30\}$; all the tasks have randomized deadline miss constraints with $k_i \in \{3, \ldots, 10\}$, and randomized periods and deadlines in a bucket from a realistic case study [40]. With 30 tasks, we hit

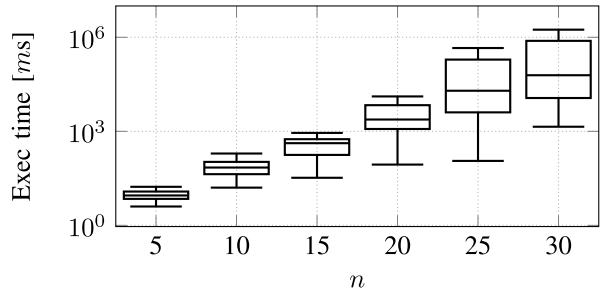

Fig. 13. Execution time box-plot for the analysis of a task chain with n tasks.

a 2-h-long timeout in 12 out of the 100 randomly generated chains. Furthermore, the CPLEX community edition limits the number of problem variables that can be specified, and it was impossible to analyze chains beyond 30 tasks without exceeding that limit. Fig. 13 shows a boxplot-representation of the execution time for the different chains, each box being the result of 100 experiments. The experiments have been performed on an 11th Generation Intel Core i7-1165G7 with 16 GB of RAM. As expected, the execution time increases exponentially with the size of the chain. However, it is possible to analyze the worst-case metrics for reasonably-sized chains in an acceptable amount of time (with 25 tasks, on average an answer is obtained in less than 10 s).

VIII. Conclusion

This article presents the first analysis of time-triggered task chains, in which tasks communicate according to the LET paradigm and where some of them can be subject to deadline misses according to the weakly-hard (m, k) model. We formulate the problem of analyzing chain metrics as an MILP problem and implement it. We then analyze a realistic application of task chains, demonstrating that our approach is able to determine the impact of deadline misses and aid design decisions. Future work will target more complex task chains, including graph structures and redundant tasks, as well as different deadline miss handling strategies.

Acknowledgment

The authors would like to thank Enrico Bini for his precious help with the problem formalization. This publication reflects only the authors' view and the European Commission is not responsible for any use that may be made of the information it contains.

References

[1] IBM ILOG CPLEX optimizer." Accessed: Apr. 7, 2022. [Online]. Available: https://www.ibm.com/analytics/cplex-optimizer

[2] L. Ahrendts, S. Quinton, T. Boroske, and R. Ernst, "Verifying weakly-hard real-time properties of traffic streams in switched networks," in *Proc. Euromicro Conf. Real-Time Syst. (ECRTS)*, 2018, pp. 1–22.

[3] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis, "An empirical survey-based study into industry practice in real-time systems," in *Proc. Real-Time Syst. Symp. (RTSS)*, 2020, pp. 3–11.

[4] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *Proc. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, 2016, pp. 159–169.

[5] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *J. Syst. Archit.*, vol. 80, pp. 104–113, Oct. 2017.

[6] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Analyzing end-to-end delays in automotive systems at various levels of timing information," *ACM SIGBED Rev.*, vol. 14, no. 4, pp. 8–13, 2018.

[7] M. Beckert, M. Möstl, and R. Ernst, "Zero-time communication for automotive multi-core systems under SPP scheduling," in *Proc. Conf. Emerg. Technol. Factory Autom. (ETFA)*, 2016, pp. 1–9.

[8] G. Bernat, A. Burns, and A. Liamosi, "Weakly hard real-time systems," *IEEE Trans. Comput.*, vol. 50, no. 4, pp. 308–321, Apr. 2001.

[9] A. Biondi and M. Di Natale, "Achieving predictable multicore execution of automotive applications using the LET paradigm," in *Proc. Real-Time Embedded Technol. Appl. Symp. (RTAS)*, 2018, pp. 240–250.

[10] A. Biondi, P. Pazzaglia, A. Balsini, and M. Di Natale, "Logical execution time implementation and memory optimization issues in autosar applications for multicores," in *Proc. Workshop Anal. Tools Methodol. Embedded Real-Time Syst. (WATERS)*, 2017, pp. 1–7.

[11] S. Bozhko, G. von der Brüggen, and B. Brandenburg, "Monte Carlo response-time analysis," in *Proc. Real-Time Syst. Symp. (RTSS)*, 2021, pp. 342–355.

[12] D. Casini, T. Blaß, I. Lütkebohle, and B. B. Brandenburg, "Response-time analysis of ROS 2 processing chains under reservation-based scheduling," in *Proc. Euromicro Conf. Real-Time Syst. (ECRTS)*, 2019, pp. 1–23.

[13] A. Cervin, P. Pazzaglia, M. Barzegaran, and R. Mahfouzi, "Using jittertime to analyze transient performance in adaptive and reconfigurable control systems," in *Proc. 24th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Zaragoza, Spain, Sep. 2019, pp. 1025–1032.

[14] K.-H. Chen and J.-J. Chen, "Probabilistic schedulability tests for uniprocessor fixed-priority scheduling under soft errors," in *Proc. Symp. Ind. Embedded Syst. (SIES)*, 2017, pp. 1–8.

[15] K.-H. Chen, N. Ueter, G. von der Brüggen, and J.-J. Chen, "Efficient computation of deadline-miss probability and potential pitfalls," in *Proc. Des. Autom. Test Europe (DATE)*, 2019, pp. 896–901.

[16] H. Choi, M. Karimi, and H. Kim, "Chain-based fixed-priority scheduling of loosely-dependent tasks," in *Proc. Conf. Comput. Des.*, 2020, pp. 631–639.

[17] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. L. Sangiovanni-Vincentelli, "Period optimization for hard real-time distributed automotive systems," in *Proc. Des. Autom. Conf.*, 2007, pp. 278–283.

[18] R. I. Davis and L. Cucu-Grosjean, "A survey of probabilistic timing analysis techniques for real-time systems," *Leibniz Trans. Embedded Syst.*, vol. 6, no. 1, pp. 1–60, 2019.

[19] P. Derler, E. A. Lee, S. Tripakis, and M. Törngren, "Cyber-physical system design contracts," in *Proc. Conf. Cyber-Phys. Syst. (ICCPS)*, 2013, pp. 109–118.

[20] J. L. Díaz *et al.*, "Stochastic analysis of periodic real-time systems," in *Proc. Real-Time Syst. Symp. (RTSS)*, 2002, pp. 289–300.

[21] M. Dürr, G. V. D. Brüggen, K.-H. Chen, and J.-J. Chen, "End-to-end timing analysis of sporadic cause-effect chains in distributed systems," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 5, pp. 1–24, 2019.

[22] R. Ernst, S. Kuntz, S. Quinton, and M. Simons, "The logical execution time paradigm: New perspectives for multicore systems," *Dagstuhl Rep.*, vol. 8, no. 2, pp. 122–149, 2018.

[23] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics," in *Proc. Real-Time Syst. Symp. (RTSS)*, 2009, pp. 1–8.

[24] G. Frehse, A. Hamann, S. Quinton, and M. Woehrle, "Formal analysis of timing effects on closed-loop properties of control software," in *Proc. Real-Time Syst. Symp. (RTSS)*, 2014, pp. 53–62.

[25] K.-B. Gemlau, L. Köhler, R. Ernst, and S. Quinton, "System-level logical execution time: Augmenting the logical execution time paradigm for distributed real-time automotive software," *Trans. Cyber-Phys. Syst.*, vol. 5, no. 2, pp. 1–27, 2021.

[26] R. Gerber, S. Hong, and M. Saksena, "Guaranteeing real-time requirements with resource-based calibration of periodic processes," *IEEE Trans. Softw. Eng.*, vol. 21, no. 7, pp. 579–592, Jul. 1995.

[27] A. Girault, C. Prévot, S. Quinton, R. Henia, and N. Sordon, "Improving and estimating the precision of bounds on the worst-case latency of task chains," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2578–2589, Nov. 2018.

[28] M. Günzel, K.-H. Chen, N. Ueter, G. von der Brüggen, M. Dürr, and J.-J. Chen, "Timing analysis of asynchronized distributed cause-effect chains," in *Proc. Real Time Embedded Technol. Appl. Symp. (RTAS)*, 2021, pp. 40–52.

[29] A. Hamann, D. Dasari, S. Kramer, M. Pressler, and F. Wurst, "Communication centric design in complex automotive embedded systems," in *Proc. Euromicro Conf. Real-Time Syst. (ECRTS)*, 2017, pp. 1–20.

[30] Z. A. Hammadeh, R. Ernst, S. Quinton, R. Henia, and L. Rioux, "Bounding deadline misses in weakly-hard real-time systems with task dependencies," in *Proc. Des. Autom. Test Europe (DATE)*, 2017, pp. 584–589.

[31] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis—The symTA/S approach," *IEE Proc. Comput. Digit. Techn.*, vol. 152, no. 2, pp. 148–166, 2005.

[32] T. A. Henzinger, C. M. Kirsch, M. A. A. Sanvido, and W. Pree, "From control models to real-time code using Giotto," *IEEE Control Syst. Mag.*, vol. 23, no. 1, pp. 50–64, Feb. 2003.

[33] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," *Proc. IEEE*, vol. 91, no. 1, pp. 84–99, Jan. 2003.

[34] C. Huang, K.-C. Chang, C.-W. Lin, and Q. Zhu, "SAW: A tool for safety analysis of weakly-hard systems," in *Proc. Conf. Comput.-Aided Verif. (CAV)*, 2020, pp. 543–555.

[35] C. Huang, W. Li, and Q. Zhu, "Formal verification of weakly-hard systems," in *Proc. Conf. Hybrid Syst. Comput. Control (HSCC)*, 2019, pp. 197–207.

[36] T. Klaus, M. Becker, W. Schröder-Preikschat, and P. Ulbrich, "Constrained data-age with job-level dependencies: How to reconcile tight bounds and overheads," in *Proc. Real-Time Embedded Technol. Appl. Symp. (RTAS)*, 2021, pp. 66–79.

[37] T. Kloda, A. Bertout, and Y. Sorel, "Latency analysis for data chains of real-time periodic tasks," in *Proc. Conf. Emerg. Technol. Factory Autom. (ETFA)*, 2018, pp. 360–367.

[38] L. Köhler and R. Ernst, "Improving a compositional timing analysis framework for weakly-hard real-time systems," in *Proc. Real-Time Embedded Technol. Appl. Symp. (RTAS)*, 2019, pp. 228–240.

[39] A. Kordon and N. Tang, "Evaluation of the age latency of a real-time communicating system using the LET paradigm," in *Proc. Euromicro Conf. Real-Time Syst. (ECRTS)*, 2020, pp. 1–20.

[40] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *Proc. Workshop Anal. Tools Methodol. Embedded Real-Time Syst. (WATERS)*, 2015, pp. 1–12.

[41] S. Lampke, S. Schliecker, D. Ziegenbein, and A. Hamann, "Resource-aware control—Model-based co-engineering of control algorithms and real-time systems," *J. Passenger Cars-Electron. Electr. Syst.*, vol. 8, no. 1, pp. 106–114, 2015.

[42] M. Lauer, F. Boniol, C. Pagetti, and J. Ermont, "End-to-end latency and temporal consistency analysis in networked real-time systems," *J. Crit. Comput. Based Syst.*, vol. 5, nos. 3-4, pp. 172–196, 2014.

[43] S. Linsenmayer and F. Allgower, "Stabilization of networked control systems with weakly hard real-time dropout description," in *Proc. Conf. Decis. Control (CDC)*, 2017, pp. 4765–4770.

[44] S. Liu, B. Yu, N. Guan, Z. Dong, and B. Åkesson, "Real-time scheduling and analysis of an autonomous driving system," in *Proc. RTSS Ind. Challenge Problem*, 2021, pp. 1–47. [Online]. Available: http://2021.rtss.org/industry-session/

[45] M. Lowinski, D. Ziegenbein, and S. Glesner, "Splitting tasks for migrating real-time automotive applications to multi-core ECUs," in *Proc. Symp. Ind. Embedded Syst. (SIES)*, 2016, pp. 1–8.

[46] M. Maggio, A. Hamann, E. Mayer-John, and D. Ziegenbein, "Control-system stability under consecutive deadline misses constraints," in *Proc. Euromicro Conf. Real-Time Syst. (ECRTS)*, 2020, pp. 1–4.

[47] J. Martinez, I. Sañudo, and M. Bertogna, "Analytical characterization of end-to-end communication delays with logical execution time," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2244–2254, Nov. 2018.

[48] J. Martinez, I. Sañudo, and M. Bertogna, "End-to-end latency characterization of task communication models for automotive systems," *Real-Time Syst.*, vol. 56, no. 3, pp. 315–347, 2020.

[49] P. Pazzaglia, A. Biondi, and M. Di Natale, "Optimizing the functional deployment on multicore platforms with logical execution time," in *Proc. Real-Time Syst. Symp. (RTSS)*, 2019, pp. 207–219.

[50] P. Pazzaglia, D. Casini, A. Biondi, and M. Di Natale, "Optimal memory allocation and scheduling for DMA data transfers under the LET paradigm," in *Proc. Des. Autom. Conf. (DAC)*, 2021, pp. 1171–1176.

[51] P. Pazzaglia, C. Mandrioli, M. Maggio, and A. Cervin, "DMAC: Deadline-miss-aware control," in *Proc. 31st Euromicro Conf. Real-Time Syst. (ECRTS)*, vol. 133, 2019, pp. 1–24.

[52] P. Pazzaglia, L. Pannocchi, A. Biondi, and M. Di Natale, "Beyond the weakly hard model: Measuring the performance cost of deadline misses," in *Proc. Euromicro Conf. Real-Time Syst. (ECRTS)*, 2018, pp. 1–22.

[53] P. Pazzaglia, Y. Sun, and M. Di Natale, "Generalized weakly hard schedulability analysis for real-time periodic tasks," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 1, pp. 1–26, 2020.

[54] R. Pellizzoni and G. Lipari, "Feasibility analysis of real-time periodic tasks with offsets," *Real-Time Syst.*, vol. 30, nos. 1–2, pp. 105–128, 2005.

[55] S. Quinton, M. Hanke, and R. Ernst, "Formal analysis of sporadic overload in real-time systems," in *Proc. Des. Autom. Test Europe (DATE)*, 2012, pp. 515–520.

[56] P. Ramanathan, "Overload management in real-time control applications using (m, k)-firm guarantee," *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 6, pp. 549–559, Jun. 1999.

[57] A. Sangiovanni-Vincentelli, P. Giusto, C. Pinello, W. Zheng, and M. Di Natale, "Optimizing end-to-end latencies by adaptation of the activation events in distributed automotive systems," in *Proc. Real-Time Embedded Technol. Appl. Symp. (RTAS)*, 2007, pp. 293–302.

[58] J. Schlatow, M. Mostl, S. Tobuschat, T. Ishigooka, and R. Ernst, "Data-age analysis and optimisation for cause-effect chains in automotive control systems," in *Proc. Symp. Ind. Embedded Syst.*, 2018, pp. 1–9.

[59] Y. Sun and M. D. Natale, "Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5, pp. 1–19, 2017.

[60] S. Tobuschat, R. Ernst, A. Hamann, and D. Ziegenbein, "System-level timing feasibility test for cyber-physical automotive systems," in *Proc. Symp. Ind. Embedded Syst. (SIES)*, 2016, pp. 1–10.

[61] E. P. van Horssen, A. B. Behrouzian, D. Goswami, D. Antunes, T. Basten, and W. Heemels, "Performance analysis and controller improvement for linear systems with (m, k)-firm data losses," in *Proc. Eur. Control Conf. (ECC)*, 2016.

[62] G. von der Brüggen, N. Piatkowski, K.-H. Chen, J.-J. Chen, K. Morik, and B. B. Brandenburg, "Efficiently approximating the worst-case deadline failure probability under EDF," in *Proc. Real-Time Syst. Symp. (RTSS)*, 2021, pp. 214–226.

[63] N. Vreman, A. Cervin, and M. Maggio, "Stability and performance analysis of control systems subject to bursts of deadline misses," in *Proc. Euromicro Conf. Real-Time Syst. (ECRTS)*, 2021, pp. 1–23.

[64] W. Xu, Z. Hammadeh, A. Kröller, R. Ernst, and S. Quinton, "Improved deadline miss models for real-time systems using typical worst-case analysis," in *Proc. Conf. Real-Time Syst. (ECRTS)*, 2015, pp. 247–256.

[65] Y. Zhao, V. Gala, and H. Zeng, "A unified framework for period and priority optimization in distributed hard real-time systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2188–2199, Nov. 2018.

Paolo Pazzaglia (Member, IEEE) received the Ph.D. degree in emerging digital technologies, curriculum embedded systems from ReTiS Lab, Scuola Superiore Sant'Anna, Pisa, Italy, in 2020.

He was a visiting Ph.D. student with the Department of Automatic Control, Lund University, Lund, Sweden, in 2018 and 2019. He is a Postdoctoral Researcher with the Computer Science Department, Saarland University, Saarbrücken, Germany. His research focuses on the interaction between control systems and real-time systems, with the goal of improving robustness and enforcing determinism in modern embedded control applications.

Martina Maggio (Senior Member, IEEE) received the Ph.D. degree from the Dipartimento di Elettronica, Informazione e Bioingegneria at Politecnico di Milano, Milan, Italy, in 2012.

She has been a Professor with the Computer Science Department, Saarland University, Saarbrücken, Germany, since 2020, and an Associate Professor with the Department of Automatic Control, Lund University, Lund, Sweden, since 2017. She was a visiting graduate student with the Computer Science and Artificial Intelligence Laboratory, Massachussetts Institute of Technology, Cambridge, MA, USA, in 2010 and 2011. In 2019, she spent a sabbatical year with Bosch Corporate Research, Renningen, Germany. Her research interest revolve around the interaction between control theory and computing systems.