Optimizing Per-Core Priorities to Minimize End-To-End Latencies

Francesco Paladino
Scuola Superiore Sant'Anna, Pisa, Italy

Alessandro Biondi
Scuola Superiore Sant'Anna, Pisa, Italy

Enrico Bini
University of Turin, Italy

Paolo Pazzaglia
Robert Bosch GmbH, Corporate Research, Renningen, Germany

Abstract

Logical Execution Time (LET) allows decoupling the schedule of real-time periodic tasks from their communication, with the advantage of isolating the communication pattern from the variability of the schedule. However, when such tasks are organized in chains, the usage of LET at the task level does not necessarily transfer the same LET properties to the chain level.

In this paper, we extend a LET-like model from tasks to chains spanning over multiple cores. We leverage the designed constant latency chains to optimize per-core priority assignment. Finally, we also provide a set of heuristic algorithms, that are compared in a large-scale experimental evaluation.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Computer systems organization → Embedded and cyber-physical systems; Software and its engineering → Multiprocessing / multiprogramming / multitasking; Software and its engineering → Real-time schedulability

Keywords and phrases Cause-Effect Chains, Logical Execution Time, End-to-End Latency, Design Optimization, Task Priorities, Data Age, Reaction Time

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2024.6

Funding *Enrico Bini*: Partially supported by the spoke "FutureHPC and BigData" of the ICSC – Centro Nazionale di Ricerca in High-Performance Computing, Big Data and Quantum Computing funded by European Union – NextGenerationEU

1 Introduction

Multicore real-time systems are commonly developed by distributing tasks among the available cores. It is quite rare that real-world applications include independent tasks only: rather, as for instance witnessed by automotive benchmarks [38], they tend to be involved in a series of producer-consumer relationships that define *cause-effect task chains* [22]. A chain of tasks generally starts by sampling and processing an input of the system (e.g., produced by a sensor or another system connected by a network), proceeds by traversing a series of tasks that process the data produced by the previous task in the chain, and terminates with the production of a system output (e.g., in the form of a control output for an actuator or a network message for another system).

The deployment of real-time applications requires to bound the *end-to-end latency* for task chains [18], which represent the most stringent constraint in some systems [21, 26]. Studying the dependency of the end-to-end latency on the parameters of the tasks that compose the chain is not trivial. Indeed, even if adopting the Logical Execution Time (LET) [32] paradigm

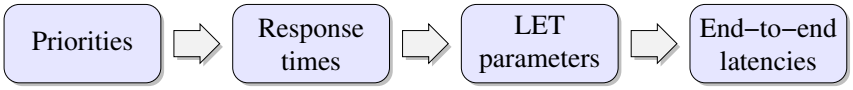

Figure 1 The approach followed in this work.

to allow tasks producing outputs in constant, logical times, the end-to-end latency of a task chain may vary significantly at run time [12]. Computing tight bounds or exact values for end-to-end latency measures is also complex from a computational perspective [28].

As such, if one intends to optimize the deployment of a real-time application, it becomes particularly challenging to explore the design space of the task set parameters to find a solution that, for instance, minimizes end-to-end latency for a set of chains. Nevertheless, the optimization of relevant parameters such as the tasks' priorities can significantly influence end-to-end latency measures [16, 53] and is hence worth to be investigated in detail.

Contribution. This paper tackles the problem of computing optimized priority assignments for real-time tasks, running onto a multicore platform under partitioned fixed-priority scheduling, which minimize a latency-dependent objective function. A generalized LET model [45, 42] is used so that tasks are allowed to publish outputs at or after their response time. In this context, the selection of priorities determines the LET parameters of tasks, which in turn affect end-to-end latency metrics (see Figure 1). The paper shows how to build *constant-latency chains*, which allow computing exact end-to-end latency measures in polynomial time (Section 4). This efficient construction is the enabler of a feasible exploration of the design space of priorities. Experiments also show that constant-latency chains increase the latency measures by a small amount. Remarkably, the regularity of the pattern of constant-latency chains may also reduce some metrics. An optimal priority assignment algorithm for tasks involved in constant-latency chains is then presented (Section 5). Finally, heuristic algorithms for priority assignment are presented (Section 6). Both the optimal and heuristic algorithms are validated by experiments.

2 Model

2.1 Tasks and communication

The application is modeled by a set of *periodic tasks*, denoted by $\mathcal{T}$. We denote a task with τ_i and the number of tasks by $N = |\mathcal{T}|$. Starting from a start-up instant, each task τ_i releases a job every *period* T_i. We assume that the indices of jobs belong to $\mathbb{N}$, the set of natural numbers including 0.

Tasks communicate by means of shared memory. Each job $j \in \mathbb{N}$ of τ_i performs the following operations, in this order:

1. it reads the input data from shared memory at the *read instant* $\mathsf{rd}_i(j)$;
2. it performs its computations; and
3. it writes the output data in shared memory at the *write instant* $\mathsf{wr}_i(j)$, making it available to other tasks.

This work focuses on a generalization of the Logical Execution Time (LET) paradigm to implement inter-task communication, inspired by previous works like [24, 47, 45], where data reads and data writes are performed at fixed instants, but not necessarily at the start and end of the task periods. We allow any protocol for managing the shared memory, as long as **(i)** read/write operations are non-blocking, **(ii)** the data is available to the consumer(s) only

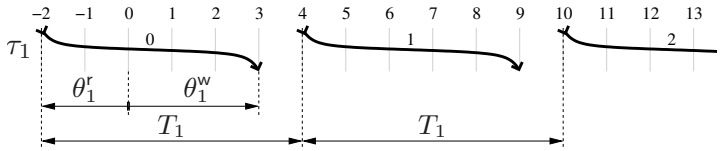

Figure 2 Representing the communication instants of the LET task $\tau_1 = \langle 6, -2, 3\rangle$. According to Eq. (1): $\mathsf{rd}_1(0) = -2$, $\mathsf{rd}_1(1) = 4$, ... and $\mathsf{wr}_1(0) = 3$, $\mathsf{wr}_1(1) = 9$, and so on.

Table 1 Summary of notations.

	Symbol	Interpretation
Tasks	τ_i	Task
	$\mathcal{T}$	Set of tasks
	$N = \lvert\mathcal{T}\rvert$	Number of tasks
	T_i	Period of task τ_i
	C_i	WCET of task τ_i
	R_i	Response time of task τ_i
	$\mathsf{rd}_i(j)$	Read instant of job j of τ_i
	$\mathsf{wr}_i(j)$	Write instant of job j of τ_i
	θ_i^{r}	Read phasing of τ_i
	θ_i^{w}	Write phasing of τ_i

	Symbol	Interpretation
Cores	m	Number of cores
	$\mathcal{T}_k$	Set of tasks mapped to core k
	$\mathsf{pri}(i)$	Priority of task τ_i
Chains	γ	Chain (function or enumeration)
	$\gamma(z)$	Index of task at position z in chain γ
	κ_i	Occurrences of τ_i in any chain
	$\mathbb{J}_\gamma$	Set of jobs of γ
	$\mathbb{J}_\gamma(\ell)$	ℓ-th job of the chain γ

when the producer task has completed the write operation, and **(iii)** each write operation by the same task overwrites the previously written data, thus read operations will only access the last written data. Note that these requirements hold in AUTOSAR implementations of the LET paradigm [13].

Under the considered generalized LET paradigm, the read/write instants of the jobs of τ_i (enumerated with index j) are *fixed* and can be written as

$$\mathsf{rd}_i(j) = j\,T_i + \theta_i^{\mathsf{r}}, \qquad \mathsf{wr}_i(j) = j\,T_i + \theta_i^{\mathsf{w}}, \tag{1}$$

with θ_i^{r} representing the *read phasing* of τ_i, θ_i^{w} representing the *write phasing* of τ_i, and job j of task τ_i executing (and completing its execution) between $\mathsf{rd}_i(j)$ and $\mathsf{wr}_i(j)$.

The phasings θ_i^{r} and θ_i^{w}, also called *offsets* and *task deadlines* in other works, may in principle take any value in $\mathbb{R}$, provided that causality holds, i.e., $\theta_i^{\mathsf{w}} \geq \theta_i^{\mathsf{r}}$. Our definition allows negative values for the phasings. We remark that the "time 0" has no special interpretation here (such as the typical "critical instant"), i.e. having the read instant at negative time does not invalidate our contribution. Also, one may add an arbitrarily large number to all phasings to make them all positive, without affecting the correctness of the paper. Since we are only interested in how task τ_i communicates with the others, we represent it by the tuple $\tau_i = \langle T_i, \theta_i^{\mathsf{r}}, \theta_i^{\mathsf{w}}\rangle$. Under the original definition of LET [32, 35], all tasks are $\tau_i = \langle T_i, 0, T_i\rangle$, with reads and writes occurring at the beginning and end of the tasks' periods, respectively. This work hence considers a more general interpretation of LET where the read and write phases are arbitrary, but anyway constant. The read and write phases are the "LET parameters" cited in Figure 1 which, as detailed in the next sections, impact end-to-end latency metrics. Figure 2 shows an example of the communication instants of a LET task.

Every job j of task τ_i has a *worst-case execution time (WCET)* C_i, which is the longest CPU time needed to complete any job. Tasks execute over a multicore architecture comprising m cores. Tasks are partitioned over the m cores and $\mathcal{T}_k \subseteq \mathcal{T}$ denotes the partition of tasks

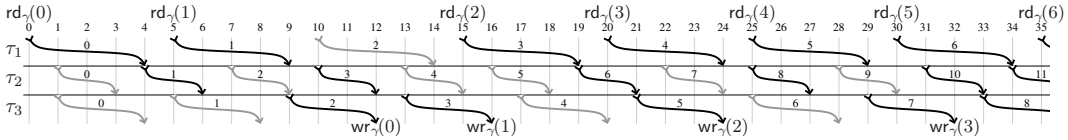

Figure 3 The jobs of chain $\gamma = [\tau_1, \tau_2, \tau_3]$ with $\tau_1 = \langle 5, 0, 4\rangle$, $\tau_2 = \langle 3, 1, 3\rangle$, and $\tau_3 = \langle 4, 1, 4\rangle$. The set of chain jobs is $\mathbb{J}\gamma = \{(0, 1, 2), (1, 3, 3), (3, 6, 5), (4, 8, 7), \ldots\}$ and the task jobs are drawn in black only if they belong to some chain job in $\mathbb{J}_\gamma$, otherwise they are drawn in gray. The depicted chain jobs are $\mathbb{J}_\gamma(0) = (0, 1, 2)$, $\mathbb{J}_\gamma(1) = (1, 3, 3)$, $\mathbb{J}_\gamma(2) = (3, 6, 5)$, and so on. Also, we use this example to illustrate the four latencies of Section 2.3. The maximum of D_γ^{LF} occurs at the job chain $\mathbb{J}_\gamma(6) = (7, 13, 11)$ with $\mathsf{rd}_\gamma(6) = \mathsf{rd}_1(7) = 35$ and $\mathsf{wr}_\gamma(6) = \mathsf{wr}_3(11) = 48$ (beyond the schedule represented above). This job chain yields the value of $D_\gamma^{\mathsf{LF}} = 48 - 35 = 13$. With the same logic, we find $D_\gamma^{\mathsf{FF}} = 19$ for job $\mathbb{J}_\gamma(2) = (3, 6, 5)$ and $D_\gamma^{\mathsf{LL}} = 19$ for job $\mathbb{J}_\gamma(1) = (1, 3, 3)$ of the chain. Finally, $D_\gamma^{\mathsf{FL}} = 27$ occurring for $\mathbb{J}_\gamma(2) = (3, 6, 5)$ and is found from $\mathsf{rd}_\gamma(1) = \mathsf{rd}_1(1) = 5$ to $\mathsf{wr}_\gamma(3) = \mathsf{wr}_3(7) = 32$.

assigned to core k. Within each core, tasks are scheduled by any fixed-priority algorithm. Each task τ_i is assigned a *priority* denoted by $\mathsf{pri}(i)$. A low value of $\mathsf{pri}(i)$ indicates a high priority.

We denote with R_i the *worst-case response time* of τ_i, computed with the standard iterative procedure [34, 2]. Then, since the output data is written at the completion of the jobs, the LET mechanism must be designed such that:

$$\theta_i^{\mathsf{w}} \geq R_i + \theta_i^{\mathsf{r}}. \tag{2}$$

2.2 Task chains

A *task chain* γ is an ordered sequence of $N_\gamma \geq 1$ tasks, and the set of all chains is denoted by Γ. We equivalently represent a chain γ in two ways:

- By enumerating its tasks as $\gamma = [\tau_a, \tau_b, \tau_c, \ldots]$ in the order they appear in the chain.
- By a function $\gamma : \{1, \ldots, N_\gamma\} \to \{1, \ldots, N\}$ such that $\gamma(z)$ is the index of the task in position z along the chain, i.e. $\tau_{\gamma(1)}$ is the first task of the chain.

In the chain γ, for any $z = 1, \ldots, N_\gamma - 1$, the task $\tau_{\gamma(z)}$ writes some data that is read by $\tau_{\gamma(z+1)}$. The last task $\tau_{\gamma(N_\gamma)}$ writes its output to the external environment. For instance, if task τ_i is the third task of γ and the first one in γ', then $\gamma(3) = \gamma'(1) = i$. Note that the same task $\tau_i \in \mathcal{T}$ may belong to more chains, while a task can appear at most once in a chain. For any pair of chains γ and γ', we use $[\gamma, \gamma']$ to denote the concatenation of the two chains, i.e. if $\gamma = [\tau_a, \tau_b, \tau_c]$ and $\gamma' = [\tau_x, \tau_y]$, then $[\gamma, \gamma'] = [\tau_a, \tau_b, \tau_c, \tau_x, \tau_y]$. Finally, we use

$$\kappa_i = \big|\{(\gamma, z) : \gamma(z) = i\}\big|, \tag{3}$$

to count the number of chains in which task τ_i appears.

Like tasks, chains have jobs too and $\mathbb{J}_\gamma \subset \mathbb{N}^{N_\gamma}$ denotes the set of all jobs of the chain γ. Informally speaking, given a chain γ, a job $(j_1, \ldots, j_{N_\gamma})$ belongs to $\mathbb{J}_\gamma$ as long as:

- for any $z = 1, \ldots, N_\gamma$, j_z is the index of a job of $\tau_{\gamma(z)}$,
- for any $z = 1, \ldots, N_\gamma - 1$, the job j_{z+1} is the earliest job of $\tau_{\gamma(z+1)}$ reading the data written by the job j_z of task $\tau_{\gamma(z)}$,
- for any pair of chain jobs $(j_1, \ldots, j_{N_\gamma}), (j'_1, \ldots, j'_{N_\gamma}) \in \mathbb{J}_\gamma$, $j_1 = j'_1$ if and only if $j_{N_\gamma} = j'_{N_\gamma}$.

Figure 3 shows an example of $\mathbb{J}_\gamma$ for a 3-task chain. The interested reader can find the formal definition in [46], which is also related to another common model, namely *immediate forward job chains* (see e.g., [20, 28, 27]).

The jobs in $\mathbb{J}_\gamma$ are *totally ordered* as they follow the total order of task jobs. Also, $\mathbb{J}_\gamma$ has a minimal element corresponding to the tuple with smallest job indices. We denote such a minimal element in $\mathbb{J}_\gamma$ as $\mathbb{J}_\gamma(0)$, and following the total order over the discrete set $\mathbb{J}_\gamma$, we denote by $\mathbb{J}_\gamma(1)$, $\mathbb{J}_\gamma(2)$, and so on all following chain jobs.

The definitions of read/write instants of Section 2.1 can now be naturally extended to any chain γ by considering the read instant of the first task $\tau_{\gamma(1)}$ in γ and the write instant of the last one $\tau_{\gamma(N_\gamma)}$. More formally, for any chain γ, we define the read and write instants of the ℓ-th job *of the chain* γ as

$$\mathbb{J}_\gamma(\ell) = (j_1, j_2, \ldots, j_{N_\gamma}) \quad \Rightarrow \quad \begin{cases} \mathsf{rd}_\gamma(\ell) = \mathsf{rd}_{\gamma(1)}(j_1) \\ \mathsf{wr}_\gamma(\ell) = \mathsf{wr}_{\gamma(N_\gamma)}\left(j_{N_\gamma}\right), \end{cases} \tag{4}$$

meaning that the read instant of the ℓ-th chain job, with $\mathbb{J}_\gamma(\ell) = (j_1, j_2, \ldots, j_{N_\gamma})$, is the read instant of the j_1-th job of the first task $\tau_{\gamma(1)}$ of γ, whereas its write instant is the write instant of the j_{N_γ}-th job of the last task of γ.

The read and write instants of chain jobs are instrumental for the definition of all the four end-to-end latencies of chains, as shown in next section.

2.3 End-to-end latency

This work allows four different definitions for the *maximum end-to-end latency* of task chains [22], which are listed below.

- The *Last-to-First* latency D_γ^{LF} represents the measure of the maximum *reaction* time of a task chain, from the last time an input of the chain is read until the first time the corresponding output of the chain is produced. Formally, it is defined as

$$D_\gamma^{\mathsf{LF}} = \max_{\ell \in \mathbb{N}} \left\{ \mathsf{wr}_\gamma(\ell) - \mathsf{rd}_\gamma(\ell) \right\}. \tag{5}$$

- The *First-to-First* latency D_γ^{FF} extends D_γ^{LF} by accounting for the input sampling delay, i.e., coping with the largest amount of time a chain is not detecting a new input at the beginning of the chain. Formally, it is defined as

$$D_\gamma^{\mathsf{FF}} = \max_{\ell \in \mathbb{N}} \left\{ \mathsf{wr}_\gamma(\ell) - \mathsf{rd}_\gamma(\ell - 1) \right\}, \tag{6}$$

  to account for the earliest time after which the input of the chain can change before being sampled by the ℓ-th job of the chain.
- The *Last-to-Last* latency D_γ^{LL} encodes the *maximum data age*, i.e., the longest time for which an input can affect the output of a chain, and is formally defined as

$$D_\gamma^{\mathsf{LL}} = \max_{\ell \in \mathbb{N}} \left\{ \mathsf{wr}_\gamma(\ell + 1) - \mathsf{rd}_\gamma(\ell) \right\}. \tag{7}$$

- The *First-to-Last* latency D_γ^{FL} extends D_γ^{LL} by accounting for the input sampling delay as D_γ^{FF} does, hence defined by

$$D_\gamma^{\mathsf{FL}} = \max_{\ell \in \mathbb{N}} \left\{ \mathsf{wr}_\gamma(\ell + 1) - \mathsf{rd}_\gamma(\ell - 1) \right\}. \tag{8}$$

Figure 3 reports the values of the four latencies for a task chain given as an example. The definitions of First-to-Last and Last-to-Last latency slightly differ from those proposed in [22] by accounting for the longest time the data produced by a chain remain eligible for being used (before they will be overwritten by the next chain job). This makes the above definitions more expressive and composable. Nevertheless, the definitions from [22] can be obtained by subtracting the period of the last task in the chain from the corresponding latency value.

In the trivial case of a chain $\gamma = [\tau_i]$ composed of one task τ_i only, by exploiting:

- the above definitions of Equations (5)–(8)
- the fact that $\mathsf{rd}_\gamma(j) = \mathsf{rd}_i(j)$ and $\mathsf{wr}_\gamma(j) = \mathsf{wr}_i(j)$, and
- the expressions of $\mathsf{rd}_i(j)$ and $\mathsf{wr}_i(j)$ of (1),

we can extend the definition of the chain latencies to the case of a single task τ_i only. Hence, we define the per-task latencies D_i^{LF}, D_i^{FF}, D_i^{LL}, and D_i^{FL} as follows

$$\begin{cases} D_i^{\mathsf{LF}} = D_{[\tau_i]}^{\mathsf{LF}} = \max\limits_{j\in\mathbb{N}} \left\{\mathsf{wr}_i(j) - \mathsf{rd}_i(j)\right\} = \max\limits_{j\in\mathbb{N}} \left\{jT_i + \theta_i^{\mathsf{w}} - (jT_i + \theta_i^{\mathsf{r}})\right\} = \theta_i^{\mathsf{w}} - \theta_i^{\mathsf{r}} \\ D_i^{\mathsf{FF}} = D_{[\tau_i]}^{\mathsf{FF}} = \theta_i^{\mathsf{w}} - \theta_i^{\mathsf{r}} + T_i = D_i^{\mathsf{LF}} + T_i \\ D_i^{\mathsf{LL}} = D_{[\tau_i]}^{\mathsf{LL}} = \theta_i^{\mathsf{w}} - \theta_i^{\mathsf{r}} + T_i = D_i^{\mathsf{LF}} + T_i \\ D_i^{\mathsf{FL}} = D_{[\tau_i]}^{\mathsf{FL}} = \theta_i^{\mathsf{w}} - \theta_i^{\mathsf{r}} + 2T_i = D_i^{\mathsf{LF}} + 2T_i. \end{cases} \tag{9}$$

3 Problem statement and approach

As illustrated qualitatively in Figure 1 and described in greater detail later in Section 5, the assignment of priorities to the tasks of the chain affects the response times of such tasks, which in turn drive the designed values of the read and write phasings (i.e., the generalized LET parameters satisfying Eqs. (1) and (2)) and thus the latency of the chains.

Given a set Γ of task chains, this work is concerned with finding a priority assignment for all tasks in the system that **minimizes** the objective function

$$\sum_{\gamma\in\Gamma} f\left(D_\gamma^{\mathsf{LF}}, D_\gamma^{\mathsf{FF}}, D_\gamma^{\mathsf{LL}}, D_\gamma^{\mathsf{FL}}\right). \tag{10}$$

The above function is representative of a general latency-dependent metric that one may want to optimize. A concrete definition will be considered later in Section 5.

To reach this goal, the next section shows how to transform each chain $\gamma \in \Gamma$ into a chain γ^* with *constant latency* under all definitions provided in Section 2.3. Besides extending the LET paradigm to task chains, making it possible to leverage the benefits of LET (no jitter, composability, etc.) also at the chain level, this approach also has the advantage of enabling a very efficient computation of *exact* end-to-end latency measures with an algorithm that has $\mathcal{O}(N_\gamma \log(\max_{i:\tau_i\in\gamma}\{T_i\}))$ complexity. Furthermore, we will prove that constant-latency chains have latency bounded by a closed-form expression that allows constructing efficient heuristics. These properties make the exploration of the priority space very efficient whenever a design is concerned with the optimization of latency-dependent objective functions.

4 Building constant latency chains

LET tasks have constant latency. It is disappointing that chains of LET tasks do not, whichever definition among the ones of Section 2.3 is used. A recent work [12] showed that the introduction of a properly designed "copier" task (hereafter called *publisher* to recall its basic work of publishing data to the readers) in a chain of two tasks does eliminate the jitter, hence making the corresponding end-to-end latency *constant*. Ensuring such property is particularly unique in the spectrum of other prominent state-of-the-art works such as [16, 17, 55], where the target is instead optimizing the average end-to-end latencies.

Publisher tasks are logical activities (i.e., they do not necessarily have to be implemented as application tasks) that are in charge of *periodically* fetching or publishing data along the chain, with their own timing. Their computation time is therefore negligible. For the purposes

of this work, it is convenient to first provide a generalization of the jitter removal approach from chains of two tasks, to chains with an arbitrary number of tasks. This generalization is a new contribution made by this work as it was only hinted as possible in [12], without providing a formal method, algorithms, and proofs. Further details on how this work differs from [12] are reported in Section 7.

They **key benefits** of this approach are twofold:

- It makes the computation of end-to-end latency metrics particularly efficient, hence enabling the design of latency-dependent optimization.
- It allows the extension of the LET paradigm to any chain while not increasing, on average, the end-to-end latency metrics.

Next, we first recall the previous result [12] targeting a chain of two tasks, and then we demonstrate how to extend it to a chain with a generic number of tasks.

4.1 Chain of two tasks

First, we introduce some needed basic operators: $\gcd(x, y)$ is the greatest common divisor of any two integers x and y and $\lfloor x \rfloor_m$ is the remainder of the integer division of x by m, so that $0 \leq \lfloor x \rfloor_m \leq m - 1$ and $x = q\, m + \lfloor x \rfloor_m$ for some $q \in \mathbb{Z}$.

Following [12], the jitter of any chain of two LET tasks $\gamma = [\tau_a, \tau_b]$ can be eliminated by adding a third *publisher task* $\tau_{\mathsf{pub}} = \langle T_{\mathsf{pub}}, \theta^{\mathsf{r}}_{\mathsf{pub}}, \theta^{\mathsf{w}}_{\mathsf{pub}} \rangle$, attached to γ to form the extended chain γ^*. As formally stated next, the publisher task τ_{pub} for a 2-task chain $\gamma = [\tau_a, \tau_b]$ has always the largest period $T_{\mathsf{pub}} = \max\{T_a, T_b\}$ among the two tasks and:

- when $T_a < T_b$, it is attached before τ_a to "slow down" the data fetched by τ_a,
- when $T_a > T_b$, it is appended after τ_b to prevent publishing redundant data produced by τ_b,
- when $T_a = T_b$, it does not change the chain which already has constant latency.

▶ **Definition 1** (Theorems 4 and 5 from [12]). *Given $\gamma = [\tau_a, \tau_b]$, let $G = \gcd(T_a, T_b)$. Then, the publisher task $\tau_{\mathsf{pub}} = \langle T_{\mathsf{pub}}, \theta^{\mathsf{r}}_{\mathsf{pub}}, \theta^{\mathsf{w}}_{\mathsf{pub}} \rangle$ and the corresponding chain γ^* extended with τ_{pub} are defined as follows:*

- *the period of τ_{pub} is*

$$T_{\mathsf{pub}} = \max\{T_a, T_b\} \tag{11}$$

- *if $T_a \geq T_b$, then $\gamma^* = [\gamma, \tau_{\mathsf{pub}}]$, that is τ_{pub} is appended after γ, where τ_{pub} has phases*

$$\theta^{\mathsf{r}}_{\mathsf{pub}} = \theta^{\mathsf{w}}_{\mathsf{pub}} = -\theta^{\mathsf{r}}_b + \theta^{\mathsf{w}}_a + \lfloor \theta^{\mathsf{r}}_b - \theta^{\mathsf{w}}_a \rfloor_G - G + \theta^{\mathsf{w}}_b + T_b \tag{12}$$

- *otherwise, $\gamma^* = [\tau_{\mathsf{pub}}, \gamma]$, that is τ_{pub} is attached before γ, where τ_{pub} has phases*

$$\theta^{\mathsf{r}}_{\mathsf{pub}} = \theta^{\mathsf{w}}_{\mathsf{pub}} = \theta^{\mathsf{r}}_b - \theta^{\mathsf{w}}_a - \lfloor \theta^{\mathsf{r}}_b - \theta^{\mathsf{w}}_a \rfloor_G + G + \theta^{\mathsf{r}}_a - T_a. \tag{13}$$

The name "publisher" follows from the coincidence of the read and write phasings of τ_{pub}: such a task makes a simple data copy and it could also be implemented by some low level mechanism (without requiring a real OS task).

As shown in [12], the advantage of the chain γ^* with the publisher task τ_{pub} over the original chain γ is that, by construction, its first and last (third) tasks $\tau_{\gamma^*(1)}$ and $\tau_{\gamma^*(3)}$, respectively, have the same period T_{pub} of (11). This, combined with the particular choice of the phases of τ_{pub} set by Equations (12) and (13) guarantees that (i) the output of each job of $\tau_{\gamma^*(1)}$ is always read by some job of $\tau_{\gamma^*(2)}$ and that (ii) all the values of $\tau_{\gamma^*(1)}$ are further

Algorithm 1 The publisher of a chain of two tasks.

function PUBLISHER(τ_a, τ_b)
  $T_{\mathsf{pub}} \leftarrow \max\{T_a, T_b\}$ ▷ *from Eq. (11)*
  $G \leftarrow \gcd(T_a, T_b)$
  $\theta' \leftarrow \lfloor \theta^{\mathsf{r}}_b - \theta^{\mathsf{w}}_a \rfloor_G - \theta^{\mathsf{r}}_b + \theta^{\mathsf{w}}_a - G$ ▷ *Eqs. (12), (13)*
  if $T_a \geq T_b$ **then**
    $\theta^{\mathsf{r}}_{\mathsf{pub}} \leftarrow \theta^{\mathsf{w}}_{\mathsf{pub}} \leftarrow \theta' + \theta^{\mathsf{w}}_b + T_b$ ▷ *Eq. (12)*
    $\tau_{\mathsf{pub}} \leftarrow \langle T_{\mathsf{pub}}, \theta^{\mathsf{r}}_{\mathsf{pub}}, \theta^{\mathsf{w}}_{\mathsf{pub}} \rangle$
    $\mathsf{pos} \leftarrow$ "tail" ▷ *publisher after τ_b*
    $\tau_{\mathsf{eq}} \leftarrow \langle T_{\mathsf{pub}}, \theta^{\mathsf{r}}_a, \theta^{\mathsf{w}}_{\mathsf{pub}} \rangle$
  else
    $\theta^{\mathsf{r}}_{\mathsf{pub}} \leftarrow \theta^{\mathsf{w}}_{\mathsf{pub}} \leftarrow -\theta' + \theta^{\mathsf{r}}_a - T_a$ ▷ *Eq. (13)*
    $\tau_{\mathsf{pub}} \leftarrow \langle T_{\mathsf{pub}}, \theta^{\mathsf{r}}_{\mathsf{pub}}, \theta^{\mathsf{w}}_{\mathsf{pub}} \rangle$
    $\mathsf{pos} \leftarrow$ "head" ▷ *publisher before τ_a*
    $\tau_{\mathsf{eq}} \leftarrow \langle T_{\mathsf{pub}}, \theta^{\mathsf{r}}_{\mathsf{pub}}, \theta^{\mathsf{w}}_b \rangle$
  return $\left(\tau_{\mathsf{pub}}, \mathsf{pos}, \tau_{\mathsf{eq}}\right)$

propagated to $\tau_{\gamma^*(3)}$, such that two consecutive jobs of $\tau_{\gamma^*(3)}$ never read the same input value propagated from $\tau_{\gamma^*(1)}$, formally $\forall j_1 \in \mathbb{N},\ \exists (j_1, j_2, j_3) \in \mathbb{J}_{\gamma^*}$, for some unique j_2, j_3. This means that the resulting chain γ^* has the same read/write instants of a periodic LET task.

▶ **Lemma 2.** *Given a chain $\gamma = [\tau_a, \tau_b]$, the chain γ^* of Definition 1 has the same read and write instants of the LET task $\tau_{\mathsf{eq}} = \langle T_{\gamma^*}, \theta^{\mathsf{r}}_{\gamma^*}, \theta^{\mathsf{w}}_{\gamma^*} \rangle$, with*

$$T_{\gamma^*} = \max\{T_a, T_b\} \tag{14}$$

$$\theta^{\mathsf{r}}_{\gamma^*} = \begin{cases} \theta^{\mathsf{r}}_a & \text{if } T_a \geq T_b \\ \theta^{\mathsf{r}}_{\mathsf{pub}} \text{ of Eq. (13)} & \text{otherwise,} \end{cases} \tag{15}$$

$$\theta^{\mathsf{w}}_{\gamma^*} = \begin{cases} \theta^{\mathsf{w}}_{\mathsf{pub}} \text{ of Eq. (12)} & \text{if } T_a \geq T_b \\ \theta^{\mathsf{w}}_b & \text{otherwise.} \end{cases} \tag{16}$$

Proof. From the construction of γ^* of Definition 1, it is always the case that $T_{\gamma^*(1)} = T_{\gamma^*(3)} = \max\{T_a, T_b\}$ and $T_{\gamma^*(2)} = \min\{T_a, T_b\}$. Then, any pair of consecutive jobs $\mathbb{J}_{\gamma^*}(j)$ and $\mathbb{J}_{\gamma^*}(j+1)$ of the chain γ^* have read and write instants separated by exactly T_{γ^*}, which proves (14).

If $T_a \geq T_b$ then, from Definition 1, $\gamma^* = [\gamma, \tau_{\mathsf{pub}}]$. This implies that $\theta^{\mathsf{r}}_{\gamma^*}$ is the read phasing of the first task of γ^*, which is $\tau_{\gamma^*(1)} = \tau_a$. Hence $\theta^{\mathsf{r}}_{\gamma^*} = \theta^{\mathsf{r}}_a$. The last task of γ^* is the publisher τ_{pub} and then $\theta^{\mathsf{w}}_{\gamma^*} = \theta^{\mathsf{w}}_{\mathsf{pub}}$ of Equation (12).

On the other hand, if $T_a \leq T_b$, then $\gamma^* = [\tau_{\mathsf{pub}}, \gamma]$, which implies that $\theta^{\mathsf{w}}_{\gamma^*} = \theta^{\mathsf{w}}_b$ and $\theta^{\mathsf{r}}_{\gamma^*} = \theta^{\mathsf{r}}_{\mathsf{pub}}$ of Eq. (13). This concludes the proof. ◀

We provide here Algorithm 1, which translates Lemma 2 and Definition 1 into a constructive form, by returning

- the publisher task τ_{pub},
- its position "head" or "tail" along the chain, and
- the LET task τ_{eq} equivalent to the constant latency chain.

The time complexity is $\mathcal{O}(\log(\max\{T_a, T_b\}))$, which is polynomial, due to the complexity of computing the greatest common divisor G with Euclid's algorithm.

The next corollary, instead, determines the analytical expression of the Last-to-First latency of γ^* found from the two-task chain $[\tau_a, \tau_b]$.

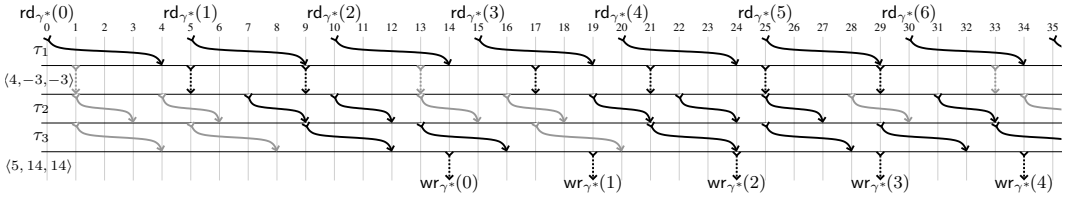

Figure 4 Constant latency chain γ^* constructed by CONSTLATCHAIN(γ) of Algorithm 2 from the same chain $\gamma = [\tau_1, \tau_2, \tau_3]$ of Figure 3. Upon the recursive invocation of CONSTLATCHAIN(γ_{tail}), with $\gamma_{\mathsf{tail}} = [\tau_2, \tau_3]$, the returned constant latency chain is $\gamma^*_{\mathsf{tail}} = [\langle 4, -3, -3\rangle, \tau_2, \tau_3]$, which is equivalent to $\tau_{\mathsf{tail}} = \langle 4, -3, 4\rangle$. After the invocation of PUBLISHER($\tau_1, \tau_{\mathsf{tail}}$) at line 6, the new publisher task $\langle 5, 14, 14\rangle$ needs to be appended to the tail of $[\tau_1, \gamma^*_{\mathsf{tail}}]$ so that the returned constant latency chain from γ is $\gamma^* = [\tau_1, \langle 4, -3, -3\rangle, \tau_2, \tau_3, \langle 5, 14, 14\rangle]$. The two publisher tasks $\langle 4, -3, -3\rangle$ and $\langle 5, 14, 14\rangle$ are represented by dotted down-arrows. Only the jobs belonging to $\mathbb{J}_{\gamma^*}$ are drawn in black, the others are drawn in grey. Unlike the case of Figure 3, all latencies are constant: $D^{\mathsf{LF}}_{\gamma^*} = 14$, $D^{\mathsf{FF}}_{\gamma^*} = D^{\mathsf{LL}}_{\gamma^*} = 19$, $D^{\mathsf{FL}}_{\gamma^*} = 24$. As also discussed in Section 4.3, publisher tasks may also allow reducing the First-to-Last latency, which here decreases from $D^{\mathsf{FL}}_{\gamma} = 27$, shown earlier in Figure 3, to $D^{\mathsf{FL}}_{\gamma^*} = 24$.

▶ **Corollary 3.** *If $\gamma = [\tau_a, \tau_b]$, the Last-to-First latency of the chain γ^* built as in Definition 1 is constant and equal to*

$$D^{\mathsf{LF}}_{\gamma^*} = D^{\mathsf{LF}}_a + D^{\mathsf{LF}}_b + \lfloor \theta^{\mathsf{r}}_b - \theta^{\mathsf{w}}_a \rfloor_G - G + \min\{T_a, T_b\}. \tag{17}$$

Proof. From Lemma 2, the chain γ^* has the same read and write instants as the LET task $\tau_{\mathsf{eq}} = \langle T_{\gamma^*}, \theta^{\mathsf{r}}_{\gamma^*}, \theta^{\mathsf{w}}_{\gamma^*}\rangle$. Hence, from (9), $D^{\mathsf{LF}}_{\gamma^*} = \theta^{\mathsf{w}}_{\gamma^*} - \theta^{\mathsf{r}}_{\gamma^*}$, with $\theta^{\mathsf{w}}_{\gamma^*}$ and $\theta^{\mathsf{r}}_{\gamma^*}$ as in (16) and (15), respectively.

If $T_a \geq T_b$, by taking the difference between $\theta^{\mathsf{w}}_{\gamma^*}$, which is Eq. (12), and $\theta^{\mathsf{r}}_{\gamma^*} = \theta^{\mathsf{r}}_a$, we find

$$D^{\mathsf{LF}}_{\gamma^*} = D^{\mathsf{LF}}_a + D^{\mathsf{LF}}_b + \lfloor \theta^{\mathsf{r}}_b - \theta^{\mathsf{w}}_a \rfloor_G + T_b - G.$$

Otherwise when $T_a < T_b$, $\gamma^* = [\tau_{\mathsf{pub}}, \gamma]$ and we take the difference between θ^{w}_b and $\theta^{\mathsf{r}}_{\mathsf{pub}}$ of Eq. (13) and we get

$$D^{\mathsf{LF}}_{\gamma^*} = D^{\mathsf{LF}}_a + D^{\mathsf{LF}}_b + \lfloor \theta^{\mathsf{r}}_b - \theta^{\mathsf{w}}_a \rfloor_G + T_a - G.$$

By merging the two cases, we find (17), as required. ◀

Corollary 3 is a fundamental step for the following parts of the paper. To further strengthen this concept, we will hereafter refer to the chain γ^* as a *constant-latency* chain.

4.2 Constant-latency chains with an arbitrary number of tasks

This section builds upon the results presented in Section 4.1 to realize constant-latency chains from chains with an arbitrary number of tasks.

Starting from the original chain γ, Algorithm 2 returns the pair $(\gamma^*, \tau_{\mathsf{eq}})$ of the constant-latency chain γ^* and its equivalent LET task τ_{eq}. If the chain is made of one task only, then it already has constant latency (line 3) and it outputs the chain itself and its only task. Otherwise, the algorithm invokes recursively CONSTLATCHAIN(γ_{tail}) (at line 5) and obtains as output, the constant-latency sub-chain γ^*_{tail} and the equivalent single task τ_{tail} (their equivalence is proved later in Theorem 4). By exploiting the equivalence between the chain γ_{tail} and the LET task τ_{tail}, at line 6 it is invoked PUBLISHER($\tau_{\gamma(1)}, \tau_{\mathsf{tail}}$) to obtain the publisher task τ_{pub}, its position pos in the constructed constant latency γ^* chain, and

Algorithm 2 Building constant-latency chains.

```
1:  function constLatChain(γ)
2:      if γ has one task only then
3:          return (γ, τ_γ(1))                                  ▷ the task itself
4:      γ_tail ← [τ_γ(2), ..., τ_γ(N_γ)]                        ▷ all tasks but τ_γ(1)
5:      (γ*_tail, τ_tail) ← constLatChain(γ_tail)
6:      (τ_pub, pos, τ_eq) ← publisher(τ_γ(1), τ_tail)
7:      if pos = "tail" then                                    ▷ publisher is last task
8:          γ* ← [τ_γ(1), γ*_tail, τ_pub]
9:      else                                                    ▷ publisher is first task
10:         γ* ← [τ_pub, τ_γ(1), γ*_tail]
11:     T ← T ∪ {τ_pub}                                         ▷ add the publisher to T
12:     return (γ*, τ_eq)
```

the LET task τ_{eq} equivalent γ^*. Finally, the last part (lines 7–10) builds the final chain γ^* by properly concatenating $\tau_{\gamma(1)}$, all tasks in γ^*_{tail}, and the publisher task τ_{pub} in the proper position indicated by $\mathsf{pos} \in \{\text{“}\mathsf{head}\text{”}, \text{“}\mathsf{tail}\text{”}\}$.

Algorithm 2 applies the recursion over the chain γ by splitting between the head $\tau_{\gamma(1)}$ and all other tasks in γ_{tail}. Any other recursive invocation (for example, to recursively split the chain in the middle) is also feasible. Such investigation, however, is narrow in scope and is not further elaborated in this paper.

Example. We illustrate here an example of invocation of constLatChain(γ) for $\gamma = [\tau_1, \tau_2, \tau_3]$ with $\tau_1 = \langle 5, 0, 4\rangle$, $\tau_2 = \langle 3, 1, 3\rangle$, and $\tau_3 = \langle 4, 1, 4\rangle$, the same chain of Figure 3.

- $\gamma_{\mathsf{tail}} \leftarrow [\tau_2, \tau_3]$, i.e., the last two tasks of γ
- constLatChain($[\tau_2, \tau_3]$) is recursively invoked on γ_{tail}
- publisher(τ_2, τ_3) builds the publisher (from Alg. 1) to make $[\tau_2, \tau_3]$ a constant latency chain
  - $\tau_{\mathsf{pub}} = \langle 4, -3, -3\rangle$ the publisher task
  - $\mathsf{pos} =$ “head” because $T_2 < T_3$ implies that τ_{pub} is added in such a position, and
  - $\tau_{\mathsf{eq}} = \langle 4, -3, 4\rangle$ is the LET task equivalent to $[\tau_{\mathsf{pub}}, \tau_2, \tau_3]$, from Line 14 of Alg. 1
- the values returned by constLatChain($[\tau_2, \tau_3]$) (line 5) are then assigned to $\gamma^*_{\mathsf{tail}} = [\langle 4, -3, -3\rangle, \tau_2, \tau_3]$ and $\tau_{\mathsf{tail}} = \langle 4, -3, 4\rangle$
- publisher($\tau_1, \tau_{\mathsf{tail}}$) is then invoked to make the final constant-latency chain and returns
  - the publisher $\tau_{\mathsf{pub}} = \langle 5, 14, 14\rangle$,
  - its position $\mathsf{pos} =$ “tail”, because $T_1 = 5 > T_{\mathsf{tail}} = 4$, and
  - the equivalent task $\tau_{\mathsf{eq}} = \langle 5, 0, 14\rangle$.
- The final chain $\gamma^* = [\tau_1, \langle 4, -3, -3\rangle, \tau_2, \tau_3, \langle 5, 14, 14\rangle]$ is then built (at line 8) by concatenating τ_1 and γ^*_{tail} with the last created publisher τ_{pub}.

Figure 4 shows the constructed constant-latency chain γ^*.

The next theorem establishes the equivalence between the constant-latency chain γ^* and a LET task, and then gives the value T_{γ^*} of the period of such a task, which is hence the periodicity of the whole chain.

▶ **Theorem 4.** *Given the pair $(\gamma^*, \tau_{\mathsf{eq}})$ returned by the invocation of* constLatChain(γ) *of Algorithm 2 for $\gamma = [\tau_{\gamma(1)}, \ldots, \tau_{\gamma(N_\gamma)}]$, the chain γ^*:*

- *has the same read and write instants as the task $\tau_{\mathsf{eq}} = \langle T_{\gamma^*}, \theta^{\mathsf{r}}_{\gamma^*}, \theta^{\mathsf{w}}_{\gamma^*}\rangle$, and*
- *the period is $T_{\gamma^*} = \max_{i=1,\ldots,N_\gamma}\{T_{\gamma(i)}\}$.*

Proof. The proof is by induction over the length of the chain.

Base case (γ has one task). The algorithm exits at line 3 and returns the constant-latency chain $\gamma^* = \gamma$. The read/write instants of γ^* coincide with the ones of its only task $\tau_{\gamma(1)}$.

Inductive step. From the inductive hypothesis on the chain of length $N_\gamma - 1$, given $(\gamma^*_{\mathsf{tail}}, \tau_{\mathsf{tail}})$ returned by constLatChain(γ_{tail}) at line 5, we know that

- γ^*_{tail} has the same read and write instants as the task $\tau_{\mathsf{tail}} = \langle T_{\mathsf{tail}}, \theta^{\mathsf{r}}_{\mathsf{tail}}, \theta^{\mathsf{w}}_{\mathsf{tail}} \rangle$, and
- $T_{\mathsf{tail}} = \max_{i=2,\dots,N_\gamma}\{T_{\gamma(i)}\}$.

If $\mathsf{pos} =$ "tail", then $\gamma^* = [\tau_{\gamma(1)}, \gamma^*_{\mathsf{tail}}, \tau_{\mathsf{pub}}]$, from line 8. From the inductive hypothesis of equivalence between γ^*_{tail} and τ_{tail}, γ^* is equivalent to $[\tau_{\gamma(1)}, \tau_{\mathsf{tail}}, \tau_{\mathsf{pub}}]$. Finally, the equivalence between $[\tau_{\gamma(1)}, \tau_{\mathsf{tail}}, \tau_{\mathsf{pub}}]$ and τ_{eq} follows from Lemma 2 allowing us to conclude that γ^* has the same read/write instants of τ_{eq}.

If instead $\mathsf{pos} =$ "head", then $\gamma^* = [\tau_{\mathsf{pub}}, \tau_{\gamma(1)}, \gamma^*_{\mathsf{tail}}]$, from line 10. From the inductive hypothesis, γ^* is equivalent to $[\tau_{\mathsf{pub}}, \tau_{\gamma(1)}, \tau_{\mathsf{tail}}]$, which is equivalent to τ_{eq} from Lemma 2.

The second statement follows from

$$T_{\gamma^*} = \overbrace{\max\{T_{\gamma(1)}, T_{\mathsf{tail}}\}}^{\text{from line 2 of Alg. 1}} = \overbrace{\max\{T_{\gamma(1)}, \max_{i=2,\dots,N_\gamma}\{T_{\gamma(i)}\}\}}^{\text{from inductive hyp.}} = \overbrace{\max_{i=1,\dots,N_\gamma}\{T_{\gamma(i)}\}}^{\text{as required}}$$

This concludes the proof. ◀

The algorithm returns the constant-latency chain γ^* equivalent to γ, with the equivalent task $\tau_{\mathsf{eq}} = \langle T_{\gamma^*}, \theta^{\mathsf{r}}_{\gamma^*}, \theta^{\mathsf{w}}_{\gamma^*} \rangle$. The time complexity of Algorithm 2 is $\mathcal{O}(N_\gamma \log(\max_{i:\tau_i \in \gamma}\{T_i\}))$ due to the $N_\gamma - 1$ invocations of Algorithm 1.

The resulting chain γ^*, built through Algorithm 2 from an arbitrary-length chain γ, has the enormous advantage over γ of having *all* latency metrics constant, as presented in the following Corollary.

▶ **Corollary 5.** *Let $(\gamma^*, \tau_{\mathsf{eq}})$, with $\tau_{\mathsf{eq}} = \langle T_{\gamma^*}, \theta^{\mathsf{r}}_{\gamma^*}, \theta^{\mathsf{w}}_{\gamma^*} \rangle$, be the pair returned by the function constLatChain(γ). Then, γ^* has all constant latencies $D^{\mathsf{LF}}_{\gamma^*}$, $D^{\mathsf{FF}}_{\gamma^*}$, $D^{\mathsf{LL}}_{\gamma^*}$, and $D^{\mathsf{FL}}_{\gamma^*}$ equal to*

$$\begin{cases} D^{\mathsf{LF}}_{\gamma^*} = \theta^{\mathsf{w}}_{\gamma^*} - \theta^{\mathsf{r}}_{\gamma^*}, & D^{\mathsf{FF}}_{\gamma^*} = D^{\mathsf{LF}}_{\gamma^*} + T_{\gamma^*}, \\ D^{\mathsf{LL}}_{\gamma^*} = D^{\mathsf{LF}}_{\gamma^*} + T_{\gamma^*}, & D^{\mathsf{FL}}_{\gamma^*} = D^{\mathsf{LF}}_{\gamma^*} + 2T_{\gamma^*}. \end{cases} \tag{18}$$

Proof. From Theorem 4, the constant latency chain γ^* has the same read and write instants as the LET task $\tau_{\mathsf{eq}} = \langle T_{\gamma^*}, \theta^{\mathsf{r}}_{\gamma^*}, \theta^{\mathsf{w}}_{\gamma^*} \rangle$. From the expressions of all four latencies of a single task of (9), the statement immediately follows. ◀

The $D^{\mathsf{LF}}_{\gamma^*}$ latency can also be upper bounded by a closed-form linear-time expression provided by the next theorem.

▶ **Theorem 6.** *The end-to-end Last-to-First latency $D^{\mathsf{LF}}_{\gamma^*}$ of the constant-latency chain γ^*, built by Alg. 2 from a chain γ of size N_γ, is bounded by*

$$D^{\mathsf{LF}}_{\gamma^*} \leq \sum_{i=1}^{N_\gamma} (D^{\mathsf{LF}}_{\gamma(i)} + T_{\gamma(i)}) - \max_{i=1,\dots,N_\gamma}\{T_{\gamma(i)}\} - N_\gamma + 1. \tag{19}$$

Proof. Let $(\gamma^*[x], \tau_{\mathsf{eq}}[x])$ be the pair returned by CONSTLATCHAIN($[\tau_x, \ldots, \tau_{N_\gamma}]$), i.e. when invoked over the last $N_\gamma - x + 1$ tasks of γ. The parameters of $\tau_{\mathsf{eq}}[x]$ are denoted in accordance to $\tau_{\mathsf{eq}}[x] = \langle T_{\gamma^*}[x], \theta^{\mathsf{r}}_{\gamma^*}[x], \theta^{\mathsf{w}}_{\gamma^*}[x] \rangle$. We also set $D^{\mathsf{LF}}_{\gamma^*}[x] = \theta^{\mathsf{w}}_{\mathsf{eq}}[x] - \theta^{\mathsf{r}}_{\mathsf{eq}}[x]$, as in Eq. (18). We remark that Theorem 4 implies that $T_{\gamma^*}[x] = \max_{i=x,\ldots,N}\{T_{\gamma(i)}\}$.

We aim at proving

$$D^{\mathsf{LF}}_{\gamma^*}[x] \leq \sum_{i=x}^{N_\gamma} (D^{\mathsf{LF}}_{\gamma(i)} + T_{\gamma(i)}) - T_{\gamma^*}[x] - N_\gamma + x \tag{20}$$

by backward induction on x, as Eq. (20) for $x = 1$ is our required theorem statement of Eq. (19).

Base case. If $x = N_\gamma$, the chain $\gamma^*[N_\gamma]$ is just the last task $[\tau_{\gamma(N_\gamma)}]$ and the RHS of (20) becomes

$$D^{\mathsf{LF}}_{\gamma(N_\gamma)} + T_{\gamma(N_\gamma)} - T_{\gamma(N_\gamma)} - 1 + 1 = D^{\mathsf{LF}}_{\gamma(N_\gamma)},$$

which makes Eq. (20) true because $D^{\mathsf{LF}}_{\gamma^*}[N_\gamma] = D^{\mathsf{LF}}_{\gamma(N_\gamma)}$.

Inductive step. We now assume Eq. (20) be true for x and we need to prove it for $x - 1$. From Cor. 3 and from the basic property of the modulo operator $\lfloor \cdot \rfloor_m \leq m - 1$, we find that the $D^{\mathsf{LF}}_{\gamma^*_\circ}$ of a two-task chain $\gamma_\circ = [\tau_a, \tau_b]$ is bounded by

$$D^{\mathsf{LF}}_{\gamma^*_\circ} \leq D^{\mathsf{LF}}_a + D^{\mathsf{LF}}_b + G - 1 + \min\{T_a, T_b\} - G = D^{\mathsf{LF}}_a + D^{\mathsf{LF}}_b + \min\{T_a, T_b\} - 1.$$

Algorithm 2 (at line 5) determines the chain $\gamma^*[x-1]$ by concatenating $\tau_{\gamma(x-1)}$ and $\gamma^*[x]$. The chain $\gamma^*[x]$ is equivalent to the task $\tau_{\mathsf{eq}}[x]$ with latency $D^{\mathsf{LF}}_{\gamma^*}[x]$ and period $T_{\gamma^*}[x]$, so we can leverage the above bound as follows:

$$D^{\mathsf{LF}}_{\gamma^*}[x-1] \leq D^{\mathsf{LF}}_{\gamma(x-1)} + D^{\mathsf{LF}}_{\gamma^*}[x] + \min\{T_{\gamma(x-1)}, T_{\gamma^*}[x]\} - 1.$$

Let us now work on the RHS above. We have

$$\begin{aligned}
&D^{\mathsf{LF}}_{\gamma^*}[x] + D^{\mathsf{LF}}_{\gamma(x-1)} + \min\{T_{\gamma^*}[x], T_{\gamma(x-1)}\} - 1 \\
&\overbrace{\qquad\qquad\qquad\qquad}^{D^{\mathsf{LF}}_{\gamma^*}[x]} \\
&\leq \sum_{i=x}^{N_\gamma} (D^{\mathsf{LF}}_{\gamma(i)} + T_{\gamma(i)}) - T_{\gamma^*}[x] - N_\gamma + x + D^{\mathsf{LF}}_{\gamma(x-1)} + \min\{T_{\gamma^*}[x], T_{\gamma(x-1)}\} - 1 \\
&= \sum_{i=x-1}^{N_\gamma} (D^{\mathsf{LF}}_{\gamma(i)} + T_{\gamma(i)}) - T_{\gamma^*}[x] - T_{\gamma(x-1)} + \min\{T_{\gamma^*}[x], T_{\gamma(x-1)}\} - N_\gamma + x - 1 \;\leq \ldots
\end{aligned}$$

Now, from the property that $a + b - \min\{a, b\} = \max\{a, b\}$, the inequality above can be further simplified as

$$\begin{aligned}
\ldots &\leq \sum_{i=x-1}^{N_\gamma} (D^{\mathsf{LF}}_{\gamma(i)} + T_{\gamma(i)}) - \max\{T_{\gamma^*}[x], T_{\gamma(x-1)}\} - N_\gamma + x - 1 \\
&= \sum_{i=x-1}^{N_\gamma} (D^{\mathsf{LF}}_{\gamma(i)} + T_{\gamma(i)}) - T_{\gamma^*}[x-1] - N_\gamma + x - 1,
\end{aligned}$$

which is exactly the statement of Eq. (20) for $x - 1$. Such a statement is then proven for all x, specifically for $x = 1$, which concludes the proof. ◀

Table 2 Relative gap percentages of the latencies of γ^* w.r.t. γ.

	Benchmark periods			Log-uniform periods		
	Avg	Min	Max	Avg	Min	Max
$D^{\mathsf{LF}}_{\gamma^*}$	3.16%	0%	98.04%	9.51%	0%	90.91%
$D^{\mathsf{LL}}_{\gamma^*}$	2.12%	0%	43.1%	5.3%	0%	39.7%
$D^{\mathsf{FF}}_{\gamma^*}$	1.3%	0%	48.31%	5.89%	0%	44.1%
$D^{\mathsf{FL}}_{\gamma^*}$	1.36%	-14.29%	19.92%	1.92%	-18.74%	24.06%

4.3 Evaluation of the insertion of publisher tasks

An experiment was set up to compare the constant-latency values produced by Alg. 2 with the exact latencies of (5)–(8). The latter was computed by exploring all chain jobs in $\mathbb{J}_\gamma$ up to the tasks' hyper-period. We generated random chains in two ways: (i) tasks with periods from the automotive benchmarks in [38], and (ii) tasks with log-uniform periods.

Benchmark periods [38]. The periods are drawn from $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$, with a probability set by the benchmark. The number of different periods per chain was uniformly chosen in $\{3, 4, 5\}$. For each period, we created from 1 to 3 tasks, hence with the number of tasks per chain ranging in $[3, 15]$. Finally, the tasks were randomly arranged in each chain. We generated 10^6 chains as described above.

Log-uniform periods. The periods were drawn from $[1, 1000]$ with a log-uniform distribution. The number of different task periods was 3 or 4 and for each period value we created from 1 to 3 tasks and placed them randomly along the chain. Because of the much larger hyper-period of this setup, we generated 10^4 random chains. In both cases, for each task τ_i, we set $\theta^{\mathsf{r}}_i = 0$ and the write phasing θ^{w}_i was drawn uniformly from the interval $[1, T_i]$, so abstracting from the actual WCETs of the tasks (varying also θ^{r}_i is not expected to provide different results). All four end-to-end latencies listed in Section 2.3 were evaluated.

Table 2 reports the average, minimum, and maximum of the relative gap $(D^{\mathsf{LF}}_{\gamma^*} - D^{\mathsf{LF}}_{\gamma})/D^{\mathsf{LF}}_{\gamma}$, in percentage, for all four semantics (D^{LF}_{γ}, D^{LL}_{γ}, D^{FF}_{γ}, and D^{FL}_{γ}) and for the two methods for random periods. From the results, it emerges that the latencies of the chain γ^* with the publisher tasks computed with Alg. 2 are, on average, significantly close to the values of the original chain γ. Also, it is worth observing that the regularity in the pattern of the constant-latency chain γ^* can even provide a *shorter First-to-Last latency* $D^{\mathsf{FL}}_{\gamma^*}$ compared to the original D^{FL}_{γ}. This apparently surprising property can be explained by noting that the read and write instants of two consecutive jobs of γ^* are always regularly separated by one period T_{γ^*}, while in the non-constant-latency chain γ this gap can be larger. An example of this phenomenon is shown by comparing D^{FL}_{γ} of the chain γ in Figure 3, which is 27 (between $\mathsf{rd}_\gamma(1)$ and $\mathsf{wr}_\gamma(3)$), with the equivalent chain γ^* shown in Figure 4, which has instead a constant $D^{\mathsf{FL}}_{\gamma^*} = 24$.

Noticeably, the runtime for computing the latencies of γ^* was five orders of magnitude smaller than the time needed for γ, since we are comparing a polynomial time algorithm vs. an algorithm that needs to construct all jobs over a hyper-period.

This preliminary evaluation shows that the benefits of the chain γ^* with publishers (constant per-job latencies, low time complexity) generally dominate its downsides (small latency gap in average). In the rest of the paper, we will use such a chain γ^* for the optimization of the priorities.

5 Optimal E2E latency aware priorities

In this section, we leverage the results presented above to explore the priority assignment of systems with constant-latency chains. The contribution provided in the previous section serves both as an enabler to efficiently evaluate different priority assignments as well as a set of mathematical foundations to devise analysis-driven optimization algorithms. More specifically, Algorithm 2 and Corollary 5, presented above, enable a feasible design space exploration when end-to-end latencies are among the constraints or participate in the cost to be optimized. We recall from Section 4.3 that the runtime for computing the latencies without publisher tasks is five orders of magnitude greater than that to compute the latencies with Algorithm 2. Without publisher tasks, the design space exploration is computationally more expensive.

A problem-specific optimization algorithm is presented next by building upon these observations. Note that the family of Optimal Priority Assignment (OPA) algorithms [1, 19] cannot be seamlessly applied to solve this problem, as they are concerned with ensuring schedulability constraints only. The approach presented next can handle any function of the latencies in the cost, in the constraints, or both. Nevertheless, for the purpose of the results discussed next, we choose the following representative form of the cost in (10):

$$\text{minimize } \mathcal{F}(\Gamma) = \sum_{\gamma \in \Gamma} D^{\mathsf{LF}}_{\gamma^*}, \tag{21}$$

with γ^* denoting a constant-latency chain, as constructed by Algorithm 2 from any $\gamma \in \Gamma$. This cost definition is relevant as it fairly provides benefits to all task chains.

Equation (19) of Theorem 6 links linearly the reduction of the task latencies D^{LF}_i to the reduction of $D^{\mathsf{LF}}_{\gamma^*}$, which are in the cost. Hence, it is convenient to set D^{LF}_i to its minimal value compatible with the schedulability constraints of (2), that is

$$D^{\mathsf{LF}}_i = \theta^{\mathsf{w}}_i - \theta^{\mathsf{r}}_i = R_i, \tag{22}$$

with the response time R_i computed through the standard iterative procedure [34, 2].

In this work, the design space of the priority assignments is explored and possibly pruned over a *decision tree*. At the root, no priority is assigned. At each level of the tree, a priority is assigned to some task. The assignment is performed on a per-core basis, as priorities are meaningful only among tasks belonging to the same core.

Algorithm 3 describes the recursive procedure, which takes the following input parameters: **(i)** $\mathcal{T}$: the set of all tasks; **(ii)** $\mathcal{U}$: the set of unassigned tasks, i.e., tasks that do not have a priority assigned yet; **(iii)** p: the current priority level in the decision tree; **(iv)** k: the core for which the priority assignment is currently in progress; **(v)** $currMin$: the current minimum objective function value.

The procedure starts with explorePrioTree$(\mathcal{T}, \mathcal{T}_1, |\mathcal{T}_1|, 1, +\infty)$. The exploration of the priorities proceeds level by level and is encoded by Lines 11–16. At any node at priority p, the algorithm creates a branch for each of the tasks currently in $\mathcal{U}$, which do not have a priority yet. All these branches are at priority level $p-1$. The recursive procedure terminates whenever a leaf of the tree is reached (Line 4). From Lines 2-9, whenever the priority assignment of the k-th core is complete, the algorithm moves to the next one. Once all tasks have a priority assigned, a leaf of the decision tree is reached (Line 4), enabling the explicit evaluation of the cost, possibly updating the minimum one ($currMin$) found so far.

In principle, this algorithm has a worst-case complexity of $O(N!)$. For this reason we used a branch-and-bound approach by means of two pruning rules to check if the current (partial) priority assignment can be pruned from the search (Line 14). The pruning rules can be independently enabled/disabled and we provide next the proof that they do not compromise optimality.

Algorithm 3 Exploring the priority space.

```
1: procedure EXPLOREPRIOTREE(𝒯, 𝒰, p, k, currMin)
2:     if 𝒰 = ∅ then
3:         k ← k + 1
4:         if k > m then                                   ▷ Reached a leaf of the tree
5:             if EVALOBJFUN(𝒯) < currMin then
6:                 currMin ← EVALOBJFUN(𝒯)
7:             return
8:         else                                            ▷ Moving to the next core
9:             𝒰 ← 𝒯_k; p ← |𝒯_k|
10:    p ← p − 1
11:    for τ_i ∈ 𝒰 do                                     ▷ Creating branches
12:        pri(i) ← p
13:        R_i ← COMPUTERESPONSETIME(τ_i, 𝒯, 𝒰)
14:        if R_i > T_i or GETLOWERBOUND(𝒯, 𝒰) > currMin then
15:            return                                      ▷ Prune the sub-tree
16:        EXPLOREPRIOTREE(𝒯, 𝒰 \ {τ_i}, p, k, currMin)
```

▶ **Rule 1** (schedulability-based pruning). *A partial priority assignment and its corresponding subtree can be pruned if the newly assigned task τ_i is not schedulable, i.e., $R_i > T_i$.*

▶ **Theorem 7.** *During the exploration of the priority decision tree with Algorithm 3, pruning the subtrees with Rule 1 does not prevent from reaching the optimal priority assignment according to Eq. (21).*

Proof. Suppose the algorithm reaches a partial priority assignment where task τ_i has just been assigned priority p, but Line 14 finds $R_i > T_i$. Similarly as in OPA [1, 19], priorities are assigned bottom-up, enabling the evaluation of the response time R_i even without the relative priority order of higher-priority tasks in $\mathcal{U}$. Therefore, any complete priority assignment in the subtree of this node will find τ_i non-schedulable. Being we interested in finding the optimal priority assignment where all the tasks are schedulable, it is guaranteed that the optimal solution cannot be in the current subtree; hence, it can be pruned without compromising the optimality of Algorithm 3. ◀

▶ **Rule 2** (cost-based pruning). *A partial priority assignment and its corresponding subtree can be pruned if a lower bound of the cost is greater than the minimum cost found by the algorithm so far, i.e., GETLOWERBOUND$(\mathcal{T},\mathcal{U}) > currMin$ (Line 14).*

The lower bound returned by GETLOWERBOUND$(\mathcal{T},\mathcal{U})$ can be computed by considering the best possible response times for the unassigned tasks (i.e., $R_i = C_i$). If even under this optimistic hypothesis the cost is larger than the current minimum, the whole subtree can be safely pruned.

▶ **Theorem 8.** *During the exploration of the priority decision tree with Algorithm 3, pruning the subtrees with Rule 2 does not prevent from reaching the optimal priority assignment according to Eq.* (21).

Proof. Suppose the algorithm reaches a partial priority assignment where task τ_i has just been assigned priority p, but Line 14 finds GETLOWERBOUND$(\mathcal{T},\mathcal{U}) > currMin$. For the lower bound computation, every task $\tau_k \in \mathcal{U}$ is optimistically assumed to have $R_k = C_k$, because R_k cannot be determined for tasks not yet having a priority assigned. Whatever priority assignment for τ_k in the subtree of the current node will necessarily find $R_k \geq C_k$,

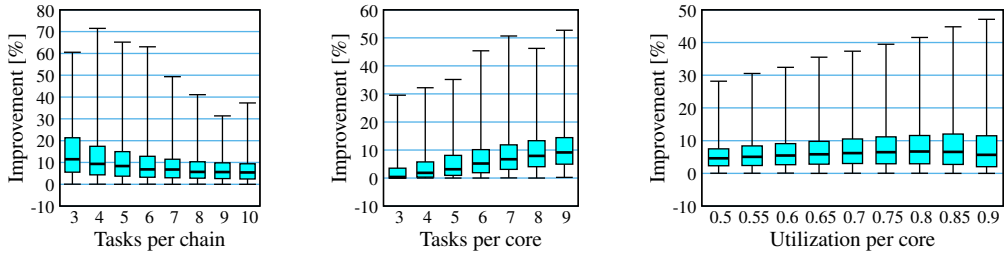

Figure 5 Improvement of the optimal priority assignment w.r.t. RM. The percentage of improvement is evaluated over the number of tasks per chain (left), the number of tasks per core (middle), and the per-core total utilization.

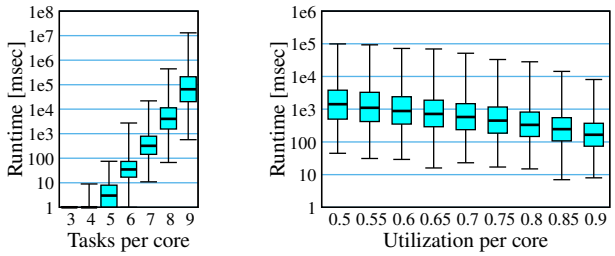

Figure 6 Runtime of the optimal priority assignment as function of the number of tasks per core (left) and of the utilization per core (right).

with C_k being a lower bound for R_k. Because of Theorem 6 and Eq. (22), an increase in the response time of tasks leads to an increase of chain latencies, which in turn inflates the cost function of Eq. (21). The subtree of the current node will contain complete priority assignments whose objective function value will be no lower than the lower bound just computed. Therefore, since the lower bound is already greater than the current minimum, the subtree of the current node cannot contain the optimal priority assignment and can be pruned without compromising the optimality of Algorithm 3. ◀

The exploration of the whole priority space with pruning guarantees that the returned solution is the optimal priority assignment minimizing $\mathcal{F}(\Gamma)$, as it is proved next.

▶ **Theorem 9.** *Given a set of tasks $\mathcal{T}$ and a set of chains Γ, Algorithm 3 returns the optimal priority assignment according to Eq.* (21)*.*

Proof. Suppose Algorithm 3 returns the priority assignment encoded by the function $\mathsf{pri}_a()$, and that there exists another priority assignment $\mathsf{pri}_b()$, obtained with some other procedure, so that $\textsc{evalObjFun}_b(\mathcal{T}) < \textsc{evalObjFun}_a(\mathcal{T})$. If $\mathsf{pri}_b()$, which has lower cost value than $\mathsf{pri}_a()$, was not returned by the algorithm, then it was not explored. The algorithm explores the whole priority space except the branches pruned by Rules 1 and 2. Therefore, $\mathsf{pri}_b()$ must belong to the pruned space. Theorems 7 and 8 show that the pruned space cannot contain the optimal solution for our optimization problem; it follows that $\mathsf{pri}_b()$ is not the optimal solution. ◀

5.1 Evaluation of optimal E2E priorities

The quality of the optimal priority assignments of Alg. 3 is evaluated w.r.t Rate Monotonic (RM) [40] as a baseline. The experiments consider synthetically generated task sets running on a 3-core platform generated as follows.

The number of tasks per core is generated in the range $[3, 9]$, fixed for a given test and equal for all 3 cores. Task periods are randomly generated in the interval $[1, 1000]$ following a log-uniform distribution. The read phasing is set to $\theta_i^r = 0, \forall \tau_i$. The utilization of each task is generated by the UUniFast [9] algorithm, setting a fixed value of reference utilization $U_{\mathsf{ref}} \in [0.5, 0.9]$, equal for all 3 cores.

For each experiment, the number of task chains is randomly chosen in the range $[3, 10]$, following a uniform distribution. The number of tasks of each chain spans in the range $[3, 10]$. Chains are generated by randomly picking tasks from the set of tasks, allowing tasks mapped on different cores to be part of the same chain.

The metrics used for evaluating the optimizer are its running time and the percentage improvement of the optimal solution w.r.t. that obtained with RM, defined as $\frac{\mathcal{F}^{\mathrm{RM}} - \mathcal{F}^{\mathrm{opt}}}{\mathcal{F}^{\mathrm{RM}}} \times 100$.

Three types of experiments are set up to study the impact of the generation parameters on the above metrics.

Varying U_{ref}. We vary U_{ref} in $[0.5, 0.9]$ with step 0.05, generating 1000 task sets for each value. The number of tasks per core is fixed to 7, while the number of tasks per chain was randomly chosen in $[4, 10]$. The effects on the percentage improvement are shown in Figure 5 (right), from which it emerges that higher U_{ref} values permit larger improvements, up to more than 45% w.r.t. to RM. Running times are depicted in Figure 6 (right), where it is evident that higher U_{ref} values reduce the optimization time due to the increase in the number of configurations that can be pruned by leveraging schedulability constraints.

Varying the number of tasks per core. The number of tasks per core is varied in the range $[3, 9]$, generating 1000 task sets for each task number. The number of tasks per chain is randomly picked in $[4, 9]$. Figure 5 (center) shows representative results for $U_{\mathsf{ref}} = 0.8$ from which it emerges that the higher the number of tasks per core, the greater the improvement w.r.t. RM, with peaks above 50%. Figure 6 (left) reports the corresponding running time of the optimizer, which exponentially increases with the number of tasks. This is due to the factorial number of possible priority assignments.

Varying the number of tasks per chain. The number of tasks per chain is varied in the range $[3, 10]$. 1000 task sets are generated for each task number. Figure 5 (left) reports the results for the representative configuration with $U_{\mathsf{ref}} = 0.8$ and seven tasks per core. It can be noted that the improvement grows when the number of tasks per chain decreases, reaching very large values up to 70%.

6 Heuristic priority assignments

The minimization of the cost of Eq. (21) can be made only through the costly exploration described in Section 5 because:

- the space of priority assignment is discrete, and
- the response time R_i used for D_i^{LF} is a discontinuous function.

Hence, this section explores some heuristic algorithms.

For the purpose of highlighting the impact of priorities on the cost, we write

$$\sum_{\gamma \in \Gamma} D_{\gamma^*}^{\mathsf{LF}} \leq \sum_{\gamma \in \Gamma} \sum_{j=1}^{N_\gamma} D_{\gamma(j)}^{\mathsf{LF}} + \text{const.} = \sum_{i=1}^{N} \kappa_i D_i^{\mathsf{LF}} + \text{const.} = \sum_{i=1}^{N} \kappa_i R_i + \text{const.} \quad (23)$$

with:

- the inequality holding from Theorem 6
- the term "const." denoting the constant, priority-independent quantities of (19)
- κ_i equal to the number of times task τ_i appears in any chain in Γ, as defined in (3), and
- $D_i^{\mathsf{LF}} = R_i$, because of our choice of (22) that minimizes D_i^{LF}.

In short, we seek a priority assignment that minimizes a weighted sum of the response times of tasks.

6.1 Response-time upper-bound-driven (RUD) priorities

The first heuristic is developed under the following simplifying assumption: **(i)** we approximate the response time with a continuous upper bound [10]; **(ii)** we study the case of any task used by one chain; and **(iii)** we compare only the two highest priority tasks. Despite these restricting conditions, the derived $\mathcal{O}(N \log N)$ priority assignment performs very well, even in the general case without such restrictions.

We start by recalling the response time upper bound [10] $\bar{R}_i$ of task τ_i, that is

$$\bar{R}_i = \frac{C_i + \sum_{j:\mathsf{pri}(j)<\mathsf{pri}(i)} C_j(1-U_j)}{1 - \sum_{j:\mathsf{pri}(j)<\mathsf{pri}(i)} U_j}$$

with $U_j = C_j/T_j$. Even though tighter continuous upper bounds exist [11, 15], only this expression led us to the rule we will show next.

Let us now denote by $\Sigma\bar{R}_{j,i}$, the sum $\bar{R}_j + \bar{R}_i$, assuming that τ_j has a higher priority than τ_i (i.e. $\mathsf{pri}(j) < \mathsf{pri}(i)$) and that only $\{\tau_j, \tau_i\}$ are in the system. From the expression of the upper bound, we find

$$\Sigma\bar{R}_{j,i} = \overbrace{C_j}^{\bar{R}_j} + \overbrace{\frac{C_i + C_j(1-U_j)}{1-U_j}}^{\bar{R}_i} = 2C_j + \frac{C_i}{1-U_j}.$$

It now becomes relevant to capture the condition on the task parameters that makes $\Sigma\bar{R}_{j,i} \leq \Sigma\bar{R}_{i,j}$, corresponding to the two different priority orderings among τ_j and τ_i. For such a comparison, we find

$$\begin{aligned}
&\Sigma\bar{R}_{j,i} \leq \Sigma\bar{R}_{i,j} \quad \Leftrightarrow \quad 2C_j + \frac{C_i}{1-U_j} \leq 2C_i + \frac{C_j}{1-U_i} \quad \Leftrightarrow \\
&\frac{2}{C_i} + \frac{1}{C_j(1-U_j)} \leq \frac{2}{C_j} + \frac{1}{C_i(1-U_i)} \quad \Leftrightarrow \quad \frac{1}{C_j(1-U_j)} - \frac{2}{C_j} \leq \frac{1}{C_i(1-U_i)} - \frac{2}{C_i} \quad \Leftrightarrow \\
&\frac{1}{T_j}\frac{2U_j-1}{U_j(1-U_j)} \leq \frac{1}{T_i}\frac{2U_i-1}{U_i(1-U_i)}
\end{aligned}$$

which yields the following $\mathcal{O}(N \log N)$ *Response time Upper bound Driven (RUD)* priority assignment:

$$\boxed{\mathsf{pri}(i) \propto \frac{1}{T_i}\frac{2U_i-1}{U_i(1-U_i)}.} \tag{24}$$

Note that $\mathsf{pri}(i)$ is directly proportional to the ratio of Eq. (24) because the numerator is almost always negative. Interestingly, in the case of tasks with equal utilizations, such an ordering is exactly the same as RM. Also, note that the rule of Eq. (24) tends to give high priority to low utilization tasks.

6.2 $\hat{\kappa}$-monotonic priority assignment

The RUD priority assignment rule of Eq. (24) is derived by studying the case in which each task appears in one chain only, i.e., $\kappa_i = 1$ in Eq. (23). The heuristic presented next is instead derived by studying the general case of tasks that appear in more than one chain.

Since κ_i has a direct impact on the cost, as shown by (23), the principle explored here is to assign higher priorities to tasks having higher values of κ_i (hence, the name κ-*monotonic*), using the RUD heuristic of Eq. (24) to break ties when tasks have the same κ_i value.

Instead of directly applying this assignment, a more general approach is followed to balance between κ-monotonic and RUD. To this purpose, we re-scaled the κ_i values to obtain $\hat{\kappa_i} = \left\lfloor B \frac{\kappa}{\kappa_{\max}} \right\rfloor$, where $\kappa_{\max}$ is the maximum κ_i value of a given task set and B is a parameter representing the maximum value $\hat{\kappa_i}$ can assume. In this way, by setting $B = \kappa_{\max}$ we can get a pure κ-monotonic priority assignment, while $B = 0$ implies the RUD assignment (i.e., Eq. (24) only). The values of B in between produce a fluid combination of the two rules. Overall, after setting B, the resulting $\hat{\kappa}$-monotonic priority assignment consists in assigning the tasks priorities so that the higher $\hat{\kappa_i}$ the higher the task priority.

6.3 Bubble-sort-like meta-heuristic algorithm

We further improved the quality of the heuristic priority assignment with a bubble-sort-like meta-heuristic algorithm. Starting from any initial priority assignment above, the procedure iterates over the task set and swaps two tasks with adjacent priorities if the overall cost decreases. The choice of two tasks with adjacent priorities avoids the re-computation of the cost of all other tasks which is unaffected by the relative priority order of the two tasks.

Because the algorithm scans $N - 1$ tasks, and each one may be potentially swapped with all those having higher priority, the number of steps is $\mathcal{O}(N^2)$.

6.4 Evaluation of heuristics

The performance of the heuristic priority assignment algorithms was evaluated against the baseline of the optimal value found by the optimizer of Section 5 and then compared with RM. The quality was measured using the *cost* functions:

$$\text{cost}^{\text{heu}} = \frac{\mathcal{F}^{\text{heu}} - \mathcal{F}^{\text{opt}}}{\mathcal{F}^{\text{opt}}} * 100\,, \;\; \text{cost}^{\text{RM}} = \frac{\mathcal{F}^{\text{RM}} - \mathcal{F}^{\text{opt}}}{\mathcal{F}^{\text{opt}}} * 100. \tag{25}$$

The comparison is carried out with the RUD, κ-monotonic and $\hat{\kappa}$-monotonic algorithms. All their solutions are then used as the starting point for the bubble sort. In our testing scenario, it is not required that each task completes within its period, thus Rule 1 was not enforced in Algorithm 3. Still, the constraint $U_{\mathsf{ref}} \leq 1$ guarantees that the response time of each task is finite. The response time of tasks is computed using the formula for the arbitrary deadline case in [39]. This is representative of the case in which only the end-to-end deadline constraints are relevant. Several values for B in $\hat{\kappa}$-monotonic were evaluated. The most performant proves to be $B = \frac{\kappa_{\max}}{2}$, which is selected for the comparison.

The experimental setup is the same of Section 5.1, except for the number of tasks per core that is lowered to compensate the increased running time of the optimizer because of the missing pruning rule based on the deadline constraint. Two types of experiments were carried out.

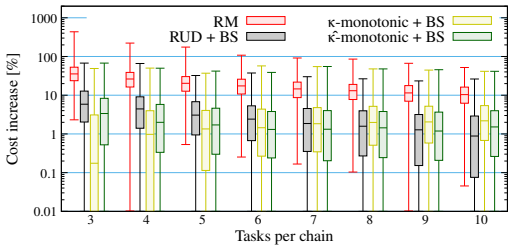

Figure 7 Cost increase over the optimal priorities as function of the number of tasks in a chain of RM (red) and the heuristic algorithms: RUD (gray), κ-monotonic (yellow) and $\hat{\kappa}$-monotonic (green), all after the bubble sort (BS).

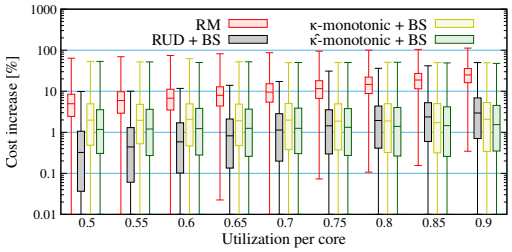

Figure 8 Cost increase over the optimal priorities as function of the per-core utilization of RM (red) and the heuristic algorithms: RUD (gray), κ-monotonic (yellow) and $\hat{\kappa}$-monotonic (green), all after the bubble sort (BS).

Varying the number of tasks per chain. The number of tasks per chain is varied in the range $[3, 10]$, generating 1000 task sets for each value in the range. U_{ref} is set to 0.8 and the number of tasks per core fixed to 6. The effects on the costs are shown in Figure 7. All the heuristic algorithms are able to obtain better solutions than RM. The RUD heuristic improves as chains grow in size, whilst κ-monotonic does the opposite. $\hat{\kappa}$-monotonic inherits both behaviors and keeps the average cost increase between 1% and 4%, fairly close to the optimum. Further, the experiments conducted in the same setting but with 4 tasks per core revealed that $\hat{\kappa}$-monotonic maintains the average cost below 0.05%, and in many cases the solution found is, in fact, the optimal value.

Varying U_{ref}. We varied U_{ref} in $[0.5, 0.9]$, generating 1000 task sets for each value. The number of tasks per chain is randomly picked in $[4, 10]$, while the number of tasks per core is fixed to 6. From Figure 8 it can be seen again that the heuristics find better solutions than RM that are close to the optimal ones, with the $\hat{\kappa}$-monotonic algorithm being the most balanced.

7 Related work

Task chains have been extensively studied in previous works from several perspectives. The distinguishing factor of our work is that it explicitly addresses the optimization of the priority assignment by leveraging the analytical properties of constant-latency chains.

Early contributions related to our research can be traced back to reactive systems [31, 7], which led to the development of programming languages such as LUSTRE [29] and ESTEREL [8]. Task chains were also studied in the context of distributed real-time systems [54, 44, 48]. The same analysis was extended to the case of tasks running upon reservation servers [41]. A alternate approach was proposed by Jayachandran and Abdelzaher [33], which reduced chains in distributed systems to the single-processor case.

End-to-end latency metrics were first taken into account at the stage of design optimization in the seminal work of Davare et al. [18]. Schlatow et al. [49] presented a holistic method based on mathematical programming to optimize the task set parameters. They did not rely on the LET paradigm and focused on data-age latencies only. Many authors worked on the analysis of end-to-end latencies, either providing exact solutions or upper bounds [50, 4, 26, 25, 37, 28, 27]. Unlike ours, these works did not consider the LET paradigm, nor did they propose effective linear-time analysis methods to compute latencies. Comparison of latencies between chains made of classic and LET tasks has been proposed, e.g., in [5, 6, 43].

Bini et al. [12] proposed a method to compare pairs of LET tasks, which we use here to assign priorities. Our work significantly differs from [12], as it studies chains with an arbitrary number of tasks, it treats four forms of end-to-end latency, it provides an algorithm to bound end-to-end latency in linear time, and, above all, it presents priority assignment algorithms. Martinez et al. [42, 43] analyzed end-to-end latencies for LET tasks to improve control performance. In [42], the authors proposed a heuristic algorithm to assign offsets to LET tasks to reduce end-to-end latencies. However, unlike our work, the algorithm focuses on one chain at a time and changes the offsets of the contained tasks, so the approach can only optimize chains having no tasks in common. Our strategy instead provides benefits for all the chains in the system. Choi et al. [17] designed a chain-aware scheduler for ROS2 to reduce end-to-end latencies of chains of callbacks. In [16], the authors introduced a chain-based fixed-priority scheduler for chains made of periodic tasks following the *read-execute-write* semantics. Verucchi et al. in [55] presented an approach to turn DAG-based applications made of multi-rate tasks into single-rate DAGs where tasks communicate following the *read-execute-write* model. Forget et al. [23] proposed a language to specify end-to-end constraints and a method to verify them. For the sake of analysis, they all explored task jobs up to the hyper-period of tasks in chains.

A related problem is the one of task placement. Pazzaglia et al. [45] proposed a solution based on mathematical programming to address the placement of runnables with end-to-end constraints under the LET paradigm. Casini and Biondi [14] proposed a method to allocate tasks to heterogeneous cores with end-to-end constraints under EDF scheduling. Recently, Sun et al. [52] proposed an ILP formulation to jointly address end-to-end constraints and EDF schedulability on each core. Han and Kim [30] addressed the problem of synthesizing a schedule aiming at reducing probabilistic end-to-end latencies.

Klaus et al. [36] proposed an execution model that maps task sets with data-age latency constraints to event-triggered systems to reduce overheads and improve schedulability. Sinha and West used constraint programming to find execution times and periods of pipelined tasks that meet the end-to-end constraints and schedulability requirements [51]. The priority assignment of tasks within distributed systems was addressed by Tang et al. [53], however, not in the context of LET communication. Bai et al. [3] studied end-to-end latency constraints in autonomous driving software.

8 Conclusion and future work

This paper addresses the selection of optimal priorities that minimizes any function of any of the different definitions for end-to-end latencies. The core of the efficiency of our method resides in the polynomial-time method for computing the latencies of the chains with the proper insertion of publisher tasks. We firmly believe that the efficiency of this novel analysis is a key enabler for further research in this area which includes the task mapping problem, the progress-based resource management, and more. Finally, another possible research direction is the combination of our approach to build constant-latency chains with techniques [42] that reduce the end-to-end latency with proper offset assignments.

References

1 Neil C Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001. doi:10.1016/S0020-0190(00)00165-4.

2 Neil C. Audsley, Alan Burns, Mike Richardson, Ken W. Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, September 1993. doi:10.1049/sej.1993.0034.

3 Yunhao Bai, Zejiang Wang, Xiaorui Wang, and Junmin Wang. Autoe2e: end-to-end real-time middleware for autonomous driving control. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 1101–1111. IEEE, 2020. doi:10.1109/ICDCS47774.2020.00092.

4 Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. Synthesizing job-level dependencies for automotive multi-rate effect chains. In *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 159–169. IEEE, 2016. doi:10.1109/RTCSA.2016.41.

5 Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. End-to-end timing analysis of cause-effect chains in automotive embedded systems. *Journal of Systems Architecture*, 80:104–113, 2017. doi:10.1016/j.sysarc.2017.09.004.

6 Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. Analyzing end-to-end delays in automotive systems at various levels of timing information. *ACM SIGBED Review*, 14(4):8–13, 2018. doi:10.1145/3177803.3177805.

7 Albert Benveniste, Paul Le Guernic, Yves Sorel, and Michel Sorine. A denotational theory of synchronous reactive systems. *Information and Computation*, 99(2):192–230, 1992. doi:10.1016/0890-5401(92)90030-J.

8 Gérard Berry and Georges Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of computer programming*, 19(2):87–152, 1992. doi:10.1016/0167-6423(92)90005-V.

9 Enrico Bini and Giorgio Buttazzo. Biasing effects in schedulability measures. In *16th Euromicro Conference on Real-Time Systems (ECRTS 2004), 30 June - 2 July 1004, Catania, Italy, Proceedings*, pages 196–203, 2004. doi:10.1109/ECRTS.2004.7.

10 Enrico Bini, Thi Huyen Châu Nguyen, Pascal Richard, and Sanjoy K. Baruah. A response-time bound in fixed-priority scheduling with arbitrary deadlines. *IEEE Transactions on Computers*, 58(2):279–286, February 2009. doi:10.1109/TC.2008.167.

11 Enrico Bini, Andrea Parri, and Giacomo Dossena. A quadratic-time response time upper bound with a tightness property. In *2015 IEEE Real-Time Systems Symposium*, pages 13–22. IEEE, 2015. doi:10.1109/RTSS.2015.9.

12 Enrico Bini, Paolo Pazzaglia, and Martina Maggio. Zero-jitter chains of periodic LET tasks via algebraic rings. *IEEE Transactions on Computers*, 72(11):3057–3071, 2023. doi:10.1109/TC.2023.3283707.

13 Alessandro Biondi and Marco Di Natale. Achieving predictable multicore execution of automotive applications using the LET paradigm. In *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2018, 11-13 April 2018, Porto, Portugal*, pages 240–250, 2018. doi:10.1109/RTAS.2018.00032.

14 Daniel Casini and Alessandro Biondi. Placement of chains of real-time tasks on heterogeneous platforms under EDF scheduling. In *2022 25th Euromicro Conference on Digital System Design (DSD)*, pages 149–156, 2022. doi:10.1109/DSD57027.2022.00029.

15 Jian-Jia Chen, Wen-Hung Huang, and Cong Liu. k2Q: A quadratic-form response time and schedulability analysis framework for utilization-based analysis. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 351–362. IEEE, 2016. doi:10.1109/RTSS.2016.041.

16 Hyunjong Choi, Mohsen Karimi, and Hyoseung Kim. Chain-based fixed-priority scheduling of loosely-dependent tasks. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pages 631–639, 2020. doi:10.1109/ICCD50377.2020.00109.

17 Hyunjong Choi, Yecheng Xiang, and Hyoseung Kim. PiCAS: New design of priority-driven chain-aware scheduling for ROS2. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 251–263, 2021. doi:10.1109/RTAS52030.2021.00028.

18 Abhijit Davare, Qi Zhu, Marco Di Natale, Claudio Pinello, Sri Kanajan, and Alberto Sangiovanni-Vincentelli. Period optimization for hard real-time distributed automotive systems. In *Proceedings of the 44th annual Design Automation Conference*, pages 278–283, 2007. doi:10.1145/1278480.1278553.

19 Robert I. Davis, Liliana Cucu-Grosjean, Marko Bertogna, and Alan Burns. A review of priority assignment in real-time systems. *Journal of Systems Architecture*, 65:64–82, 2016. doi:10.1016/j.sysarc.2016.04.002.

20 Marco Dürr, Georg Von Der Brüggen, Kuan-Hsun Chen, and Jian-Jia Chen. End-to-end timing analysis of sporadic cause-effect chains in distributed systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–24, 2019. doi:10.1145/3358181.

21 Rolf Ernst, Leonie Ahrendts, and Kai-Björn Gemlau. System level LET: Mastering cause-effect chains in distributed systems. In *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*, pages 4084–4089, 2018. doi:10.1109/IECON.2018.8591550.

22 Nico Feiertag, Kai Richter, Johan Nordlander, and Jan Jonsson. A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, 2008.

23 Julien Forget, Frédéric Boniol, and Claire Pagetti. Verifying end-to-end real-time constraints on multi-periodic models. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2017. doi:10.1109/ETFA.2017.8247612.

24 Goran Frehse, Arne Hamann, Sophie Quinton, and Matthias Woehrle. Formal analysis of timing effects on closed-loop properties of control software. In *2014 IEEE Real-Time Systems Symposium*, pages 53–62. IEEE, 2014. doi:10.1109/RTSS.2014.28.

25 Max J Friese, Thorsten Ehlers, and Dirk Nowotka. Estimating latencies of task sequences in multi-core automotive ecus. In *2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES)*, pages 1–10. IEEE, 2018. doi:10.1109/SIES.2018.8442095.

26 Alain Girault, Christophe Prévot, Sophie Quinton, Rafik Henia, and Nicolas Sordon. Improving and estimating the precision of bounds on the worst-case latency of task chains. *IEEE transactions on computer-aided design of integrated circuits and systems*, 37(11):2578–2589, 2018. doi:10.1109/TCAD.2018.2861016.

27 Mario Günzel, Kuan-Hsun Chen, Niklas Ueter, Georg von der Brüggen, Marco Dürr, and Jian-Jia Chen. Compositional timing analysis of asynchronized distributed cause-effect chains. *ACM Trans. Embed. Comput. Syst.*, March 2023. doi:10.1145/3587036.

28 Mario Günzel, Kuan-Hsun Chen, Niklas Ueter, Georg von der Brüggen, Marco Dürr, and Jian-Jia Chen. Timing analysis of asynchronized distributed cause-effect chains. In *27th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 40–52. IEEE, 2021. doi:10.1109/RTAS52030.2021.00012.

29 Nicholas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, 1991. doi:10.1109/5.97300.

30 Taeho Han and Kanghee Kim. Minimizing probabilistic end-to-end latencies of autonomous driving systems. In *IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 27–39, 2023. doi:10.1109/RTAS58335.2023.00010.

31 David Harel and Amir Pnueli. On the development of reactive systems. In *Logics and Models of Concurrent Systems - Conference proceedings, Colle-sur-Loup (near Nice), France, 8-19 October 1984*, volume 13, pages 477–498. Springer, 1984. doi:10.1007/978-3-642-82453-1_17.

32 Thomas A. Henzinger, Benjamin Horowitz, and Christoph M. Kirsch. Giotto: A time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1):84–99, 2003. doi:10.1109/JPROC.2002.805825.

33 Praveen Jayachandran and Tarek Abdelzaher. Delay composition in preemptive and non-preemptive real-time pipelines. *Real-Time Systems*, 40(3):290–320, 2008. doi:10.1007/s11241-008-9056-3.

34 Mathai Joseph and Paritosh K. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, October 1986. doi:10.1093/comjnl/29.5.390.

35 Christoph M. Kirsch and Ana Sokolova. The logical execution time paradigm. *Advances in Real-Time Systems*, pages 103–120, 2012. doi:10.1007/978-3-642-24349-3_5.

36 Tobias Klaus, Matthias Becker, Wolfgang Schröder-Preikschat, and Peter Ulbrich. Constrained data-age with job-level dependencies: How to reconcile tight bounds and overheads. In *IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 66–79. IEEE, 2021. doi:10.1109/RTAS52030.2021.00014.

37 Tomasz Kloda, Antoine Bertout, and Yves Sorel. Latency upper bound for data chains of real-time periodic tasks. *Journal of Systems Architecture*, 109:101824, 2020. doi:10.1016/j.sysarc.2020.101824.

38 Simon Kramer, Dirk Ziegenbein, and Arne Hamann. Real world automotive benchmarks for free. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

39 John P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *[1990] Proceedings 11th Real-Time Systems Symposium*, pages 201–209, 1990. doi:10.1109/REAL.1990.128748.

40 C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973. doi:10.1145/321738.321743.

41 José L. Lorente, Giuseppe Lipari, and Enrico Bini. A hierarchical scheduling model for component-based real-time systems. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006), Proceedings, 25-29 April 2006, Rhodes Island, Greece*, pages 8–pp. IEEE, 2006. doi:10.1109/IPDPS.2006.1639405.

42 Jorge Martinez, Ignacio Sañudo, and Marko Bertogna. Analytical characterization of end-to-end communication delays with logical execution time. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2244–2254, 2018. doi:10.1109/TCAD.2018.2857398.

43 Jorge Martinez, Ignacio Sañudo, and Marko Bertogna. End-to-end latency characterization of task communication models for automotive systems. *Real-Time Systems*, 56:315–347, 2020. doi:10.1007/s11241-020-09350-3.

44 José Carlos Palencia and Michael Gonzalez Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *Proceedings of the 20th IEEE Real-Time Systems Symposium, Phoenix, AZ, USA, December 1-3, 1999*, pages 328–339. IEEE, 1999. doi:10.1109/REAL.1999.818860.

45 Paolo Pazzaglia, Alessandro Biondi, and Marco Di Natale. Optimizing the functional deployment on multicore platforms with logical execution time. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 207–219, 2019. doi:10.1109/RTSS46320.2019.00028.

46 Paolo Pazzaglia and Martina Maggio. Characterizing the effect of deadline misses on time-triggered task chains. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(11):3957–3968, 2022. doi:10.1109/TCAD.2022.3199146.

47 Paolo Pazzaglia, Luigi Pannocchi, Alessandro Biondi, and Marco Di Natale. Beyond the weakly hard model: Measuring the performance cost of deadline misses. In *30th Euromicro Conference on Real-Time Systems, ECRTS 2018, July 3-6, 2018, Barcelona, Spain*, volume 106, pages 10:1–10:22. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ECRTS.2018.10.

48 Rodolfo Pellizzoni and Giuseppe Lipari. Improved schedulability analysis of real-time transactions with earliest deadline scheduling. In *11th IEEE Real Time and Embedded Technology and Applications Symposium*, pages 66–75. IEEE, 2005. doi:10.1109/RTAS.2005.28.

49 Johannes Schlatow, Mischa Mostl, Sebastian Tobuschat, Tasuku Ishigooka, and Rolf Ernst. Data-age analysis and optimisation for cause-effect chains in automotive control systems. In *13th IEEE International Symposium on Industrial Embedded Systems, SIES 2018, Graz, Austria, June 6-8, 2018*, pages 1–9. IEEE, 2018. doi:10.1109/SIES.2018.8442077.

50 Simon Schliecker and Rolf Ernst. A recursive approach to end-to-end path latency computation in heterogeneous multiprocessor systems. In *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 433–442, 2009. doi:10.1145/1629435.1629494.

51 Soham Sinha and Richard West. End-to-end scheduling of real-time task pipelines on multiprocessors. *Journal of Systems Research*, 2(1), 2022. doi:10.5070/sr32158647.

52 Jinghao Sun, Kailu Duan, Xisheng Li, Nan Guan, Zhishan Guo, Qingxu Deng, and Guozhen Tan. Real-time scheduling of autonomous driving system with guaranteed timing correctness. In *IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 185–197, 2023. doi:10.1109/RTAS58335.2023.00022.

53 Yue Tang, Xu Jiang, Nan Guan, Dong Ji, Xiantong Luo, and Wang Yi. Comparing communication paradigms in cause-effect chains. *IEEE Transactions on Computers*, 72(1):82–96, 2022. doi:10.1109/TC.2022.3197082.

54 Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and microprogramming*, 40(2-3):117–134, 1994. doi:10.1016/0165-6074(94)90080-9.

55 Micaela Verucchi, Mirco Theile, Marco Caccamo, and Marko Bertogna. Latency-aware generation of single-rate DAGs from multi-rate task sets. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 226–238, 2020. doi:10.1109/RTAS48715.2020.000-4.