Data-Age Analysis for Multi-Rate Task Chains under Timing Uncertainty

Pourya Gohari
Eindhoven University of Technology
Eindhoven, The Netherlands

Mitra Nasri
Eindhoven University of Technology
Eindhoven, The Netherlands

Jeroen Voeten
Eindhoven University of Technology
Eindhoven, The Netherlands

ABSTRACT

Safety and control functions of modern automotive systems are implemented as chains of periodic data producer/consumer tasks run at different rates. To simplify the development of such systems, automotive standards relax synchronization requirements between tasks, allowing a task to run even if its dependent tasks are inactive. This gives birth to more complex timing constraints such as data age, which specifies the maximum duration of time that input data of a task chain is still allowed to affect the output of the chain.

We develop a technique to compute lower and upper bounds on the data age of multi-rate task chains that execute upon a heterogeneous computing platform using a job-level fixed-priority scheduling policy. To the best of our knowledge, we are the first to consider uncertainties in the timing parameters (namely, both the release jitter and execution-time variation) of the tasks. Such an assumption makes the problem more challenging as it increases the number of possible schedules that the system may encounter during its lifespan. We represent these uncertainties by timing intervals and devise an analysis that uses those intervals to explore possible dependencies between jobs. We incorporated various pruning rules to make the analysis much faster and far less pessimistic than the state of the art. Our evaluations on an industrial case study as well as synthetic task sets show that our analysis reduces the overestimation of the data age by 36% on average (and up to 42%) in comparison to the state of the art when the number of tasks varies from 10 to 50.

CCS CONCEPTS

• **Computer systems organization → Real-time systems**.

KEYWORDS

Real-time systems, Cause-effect chains, Data age, Multi-rate DAG.

ACM Reference Format:
Pourya Gohari, Mitra Nasri, and Jeroen Voeten. 2022. Data-Age Analysis for Multi-Rate Task Chains under Timing Uncertainty. In *Proceedings of the 30th International Conference on Real-Time Networks and Systems (RTNS '22), June 7–8, 2022, Paris, France.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3534879.3534893

RTNS '22, June 7–8, 2022, Paris, France

ACM ISBN 978-1-4503-9650-9/22/06.
https://doi.org/10.1145/3534879.3534893

1 INTRODUCTION

In modern automotive systems, a large number of safety and control functionalities are implemented in the form of *data-dependent* periodic tasks that interact with their environment at different rates [3, 4, 21]. These systems have a large number of data types that are being updated or used by tasks (or runnables) across sub-systems designed by various parties.

To allow independent development, enhance compatibility with existing or legacy software, simplify distribution on large heterogeneous computing platforms, and satisfy reliability requirements when a subsystem fails, automotive engineering standards relax the synchronization between data-dependent tasks [1, 8]. The relaxation is technically achieved by allowing tasks to read the most recent data from shared buffer spaces (e.g., global variables) and write their output to these buffers independently of other tasks [10, 27]. Examples of this semantic can be found in the implicit communication model of AUTOSAR [1] and the publisher-subscriber model in robot operating system (ROS) [34].

However, this relaxed data synchronization model comes at a cost. In addition to traditional deadline constraints, it imposes new types of end-to-end latency constraints related to the propagation of data through sequences of tasks that together fulfill a particular system function. Such sequences of tasks are called *task chains* or *cause-effect chains* [1, 2, 25]. One of the key classes of end-to-end latency requirements is *data age* (DA) constraints. A DA constraint specifies the maximum amount of time that input data of a task chain is still allowed to have an effect on the output of this chain [6, 15, 27]. In other words, it specifies that the input data used throughout a chain of tasks should not be 'too old', otherwise, it would not accurately represent the current state of the environment.

The goal of our work is to derive relatively tight lower and upper bounds on the data age of data-dependent tasks. The problem is challenging because (i) not all input data is propagated through a task chain as consumers and producers have different rates, and (ii) data age depends on the order of data-related events (e.g., read and write), which is influenced by the execution-time variations and release jitter of the tasks. In order to obtain such bounds, all possible schedules that can be observed for a task set must be considered.

Fig. 1 shows a subset of schedules of a task chain with three data-dependent tasks, τ_1, τ_2, and τ_3, generated by a non-preemptive fixed-priority scheduling policy (in which τ_1 has the highest and τ_3 the lowest priority) deployed on one processing element. These tasks release jobs periodically (starting from time 0). As it can be seen, the execution-time variations of each task could impact the start time of the other tasks, resulting in several possible schedules. Considering that all jobs run until their worst-case execution time, the first response to the data produced by the first job of τ_1 is generated at the latest by time 4, whereas the first response to the

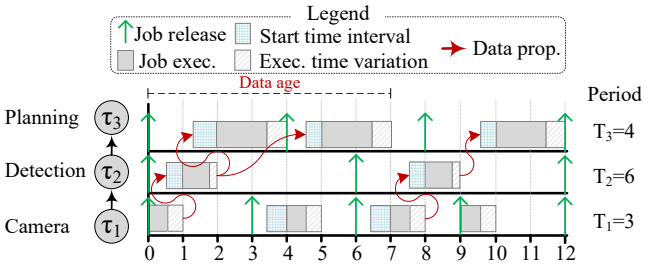

Figure 1: A set of schedules of a task chain with three tasks during one hyperperiod (from time 0 to 12).

third job of τ_1 which releases at time 6 is available at the latest by time 12. Note that input data that is not propagated through a task chain does not impact the output of a task chain. This is the case for the data of the second and fourth jobs of τ_1 as they are not processed by task τ_2. For the schedules shown in this example, the data age is 7 because the longest time that a job of τ_1 can affect the output (a job of τ_3) is the time between the release of the first job of τ_1 and the completion time of the second job of τ_3.

To support the growing number (and computation demands) of tasks and to satisfy the data-age requirements, the automotive industry is moving toward adopting heterogeneous systems-on-chip platforms. However, to provide isolation between the tasks and follow standards, each task is assigned statically to a processing element [21]. Such a design choice in turn adds to the complexity of deriving bounds of data age as it allows data producer and data consumer tasks to run concurrently.

Related work. Feiertag et al. [15] are among the first to provide a formal definition for different end-to-end latency semantics (reaction latency, data age, etc.). Additionally, they introduce a method to calculate data age for a given schedule, however, their method does not account for a collection of schedules resulted from the timing variations in the tasks' release and execution time.

Some works in the literature, try to improve end-to-end latencies by scheduling algorithms [10, 11], or enforcing control-flow dependencies by adjusting effective parameters such as periods, priorities, task-to-processor mapping, and offsets to minimize end-to-end latency [12, 23, 39]. Pagetti et al. [33] introduce a framework using PRELUDE language [16], which is a synchronous language based on the principles of LUSTRE [20] for multi-rate real-time systems, to (i) guarantee the synchronization of multi-rate tasks and (ii) provide deterministic dependency between jobs. However, we focus on deriving bounds of data age for systems where it is not possible or advisable to change the scheduling policy or the task execution semantic (e.g., due to requirements enforced by standards).

Various types of end-to-end latency analyses have been addressed in the literature. For example, *reaction latency*, i.e., the longest delay between an input and the first output of a task chain that responds to that input has been analyzed by [2, 13, 19, 24, 25, 28, 35]. Reaction latency is different from data age in that reaction latency takes into account the time it takes for the next new output to appear, in addition to the data age. As a result, it can only provide an upper bound on data age [13]. In our experiments, we will show that this upper bound is indeed pessimistic (loose).

Becker et al. [6] introduce a schedule-agnostic data age analysis, where apart from the fact that the tasks must be schedulable (respect their deadline), no assumption about the scheduling policy being used is taken into consideration when obtaining data age. Because the analysis does not account the scheduling policy and the response time of the tasks, it is inherently pessimistic, as we will demonstrate in our experiments. Later, Becker et al. [7, 8] extended their previous work in [6] and proposed a method to compute the lower and upper bounds of data age subject to the availability of certain levels of knowledge about the system, such as knowing the response times or the exact schedule. Nevertheless, when the exact schedule is unknown, this work only can exploit the worst-case response time of jobs in order to calculate a more accurate data age and since it assumes the start time intervals of jobs are unknown, the calculated data age could be pessimistic.

Davare et al. [12] provide an upper bound on end-to-end latency by summing up the worst-case response times plus the periods of every task in the task chain. This provides an upper bound on reaction latency or data age. However, Dürr et al. [13] later on showed that this bound is very pessimistic. They also proposed an analysis to derive an upper bound of reaction latency and data age for *sporadic* tasks in distributed systems. However, when applied to periodic tasks, this work results in pessimistic bounds for the reaction latency and data age. Günzel et al. [19] propose an analysis to derive an upper bound of reaction latency and data age for periodic or sporadic task chains in globally asynchronous distributed systems. This paper supposes that tasks always execute with their worst-case execution time. This allows the paper to simulate the schedule of a few hyperperiods to obtain the bounds on data age. This method, however, would not work when tasks have release jitter or execution-time variations.

This paper. We provide a method to compute lower and upper bounds on the data age of multi-rate task chains that follow the read-execute-write semantics with an implicit communication model as suggested in the AUTOSAR standard [1, 27]. We consider a general system setup, where a set of periodic tasks is scheduled by a non-preemptive *job-level fixed-priority* (JLFP) scheduling policy, a general class of scheduling policies that includes *fixed priority* (FP) and the *earliest-deadline first* (EDF), deployed on a heterogeneous computing platform, where each task is statically assigned to a processing element. We are the first to consider the impact of timing uncertainties on task parameters, in particular, the release jitter (as well as execution-time variation).

The uncertainty in timing parameters of the tasks results in having many possible schedules. We combine these schedules and represent their effect on each job of a task set (within an observation window) using two uncertainty intervals; one for the start time and one for the completion time of the job. We use the schedule-abstraction graph analysis of Nasri et al. [29, 36, 37] to obtain tight bounds on the start and completion intervals of the jobs. Our analysis utilizes these bounds to obtain the possible set of data producers for every data consumer in a task chain until we find the earliest and latest possible data producers of the entire chain. To make this process more efficient, we propose pruning rules to ignore data dependencies that have no impact on data age.

Our results on an industrial case study [22, 40] and on synthetic task sets show that our method is able to reduce the pessimism of

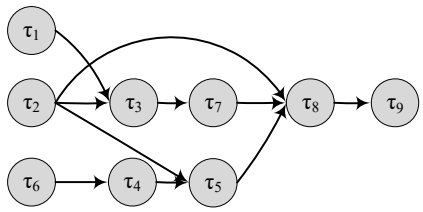

Figure 2: A data propagation graph in an automotive application [22, 40]

Table 1: The automotive application in Fig. 2 [22, 40].

Task	Function	Description $(\sigma_i, [C_i^{min}, C_i^{max}], T_i, D_i, X_i)$
τ_1	GPS	$(0, [5, 7], 50, 50, \text{core } 1)$
τ_2	Lidar	$(0, [10, 12], 50, 50, \text{core } 1)$
τ_3	Localization	$(0, [22, 28], 50, 50, \text{core } 1)$
τ_4	Detection	$(0, [25, 28], 50, 50, \text{core } 4)$
τ_5	Fusion	$(0, [18.9, 25], 50, 50, \text{core } 2)$
τ_6	Camera	$(0, [1.8, 2], 25, 25, \text{core } 4)$
τ_7	EKF	$(0, [3, 6.5], 25, 25, \text{core } 5)$
τ_8	Planner	$(0, [3.2, 5], 10, 10, \text{core } 6)$
τ_9	Control	$(0, [1.8, 4.5], 10, 10, \text{core } 3)$

the existing data-age analyses by 30% to 40% on average over all experiments, considering the impact of system utilization, number of tasks, number of task chains, and number of processing elements.

2 SYSTEM MODEL AND DEFINITIONS

We consider standard automotive applications [22, 26] which typically include a set of periodic tasks (or runnables) executing on a heterogeneous processing platform with *m processing elements* (PE) named $\mathcal{P} = \{\phi_1, \ldots, \phi_m\}$. Each task is statically assigned to a single PE on the platform and thus is not allowed to migrate. A PE can refer to a processor core or an accelerator that is embedded on a chip (such as NVIDIA TX2 or NVIDIA AGX Xavier). We assume that tasks are scheduled according to a *job-level fixed-priority* (JLFP) scheduling policy. JLFP includes commonly used policies such as the fixed-priority and EDF scheduling policies.

Task model. We consider a system that contains a set $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n data-dependent periodic tasks (or runnables), where each task follows the read-execute-write semantics specified by AUTOSAR [1] [1]. A task $\tau_i \in \Gamma$ is a tuple $\left(\sigma_i, \left[C_i^{min}, C_i^{max}\right], T_i, D_i, X_i\right)$, where σ_i denotes the maximum release jitter, T_i is the period, D_i is the relative deadline (where $D_i \leq T_i$), and $X_i \in \mathcal{P}$ is the PE on which τ_i execute. The release jitter could be the result of interrupt latencies or timer inaccuracies [29]. Interval $[C_i^{min}, C_i^{max}]$ models the range of possible execution times of task τ_i, where C_i^{min} and C_i^{max} represent the best-case (BCET) and worst-case (WCET) execution times, respectively, when it is executed on the processing element X_i. The execution-time variations can be due to cache behavior, input data variations, or diversity of program paths. The timing parameters can be discrete or continuous. For a set of periodic tasks that arrive at the time $t = 0$ with a constrained-deadline and without release offset, the hyperperiod is the shortest time interval after which the arrival pattern of the jobs repeat. The hyperperiod is the least common multiple of the periods (i.e., $H = lcm(T_1, T_2, \ldots, T_n)$ [9].

We assume a run-to-completion semantics for task execution. There will be a short discussion on how our analysis is applicable to preemptive task sets in Sec. 3.5.

Job model. Each task $\tau_i \in \Gamma$ releases *jobs* periodically with period T_i from the starting moment of the system, which we assume to be at time $t = 0$. The kth $(1 \leq k)$ job of a task τ_i is released non-deterministically within the interval $\left[r^{min}, r^{max}\right]$, where $r^{min} = (k-1) \cdot T_i$ is the earliest release time (a.k.a. *arrival time* [5]) and $r^{max} = r^{min} + \sigma_i$ is the latest release time of the job. Following Audsley's model, the absolute deadline of the job is at $(k-1) \cdot T_i + D_i$.

We form a finite job set $\mathcal{J} = \{J_1, J_2, \ldots, J_z\}$ consisting of *all* the jobs released by the task set Γ during the interval $[0, OW)$, where $z = |\mathcal{J}|$ and OW is an *observation window* which is an upper bound on two values: (i) the shortest interval of time after which the release pattern of the tasks repeats and all shared data buffers are filled with a data generated by a data-producer task (i.e., $2H$, where H is the hyperperiod of the task set), and (ii) the worst-case value of the data age among all task chains. We will discuss this further in Sec. 2.2 after we formally define data age.

A job $J_i \in \mathcal{J}$ is a tuple $\left(\left[r_i^{min}, r_i^{max}\right], \left[C_i^{min}, C_i^{max}\right], d_i, p_i, X_i\right)$, where r_i^{min} and r_i^{max} are the earliest and latest release times, d_i is the absolute deadline, X_i is the processing element, and p_i is the priority of the job (assigned by the JLFP scheduling policy). We assume a smaller value of p_i represents a higher priority, and ties are broken arbitrarily but consistently. Other job parameters (i.e., C_i^{min}, C_i^{max}, and X_i) are inherited from its task. Function $Task(J_i)$ maps jobs to their corresponding tasks, i.e., $Task(J_i) = \tau_j$ implies that J_i is a job of task τ_j and function $PE(J_i)$ maps jobs to their assigned processing element (X_i).

We use angle brackets to denote sequences and curly brackets to denote sets. For set X, we let X^+ denote the positive Kleene closure, i.e., the set of all non-empty sequences of elements of X.

Data model. We assume that the tasks follow the *read-execute-write* semantics according to the *implicit communication model* of AUTOSAR [1, 17], namely, a task reads all its required inputs as soon as it starts and stores a local copy of them into its memory. The task writes its outputs back to shared buffers when finishing its execution. The inter-task data transfer is done via global shared buffer. The time required to safely access these shared buffers have been accounted for in the tasks' WCET. Note that the implicit communication model does not enforce a synchronization between data consumers and producers.

Data propagation model. We assume the application to be modeled as a *data propagation graph* (DPG). This is a *directed acyclic task graph* (DAG) in which vertices represent tasks and edges represent their data dependencies. Namely, an edge from task τ to task τ' implies that some of the outputs of τ are used by τ'. In this case, τ is called a *data producer* for τ' and τ' is called a *data consumer* of τ.

[1] In this work, we assume that each task consists of one elementary software module or runnable. Thus, our task model is consistent with both the implicit model of AUTOSAR [1] as well as the task model introduced in the RTSS'21 industrial challenge [27].

We use $Pred(\tau)$ to refer to the set of data producers for task τ, also called the *predecessors* of τ.

Fig. 2 shows an example of a DPG in an autonomous driving application (from the WATERS'2019 challenge [22]). This DPG shows an *advanced driver-assistance system* (ADAS), covering the entire computation path, from sensor input to steering command. In this example, different sensor systems (LiDAR, cameras, and GPS) collect data according to their update rate. These data are then propagated to tasks that periodically perform operations such as localization, detection, and prediction, thereby computing the trajectory of the vehicle. The data-propagation paths in the DPG that perform a certain system functionality are called *task chains*.

Definition 2.1 (Task chain (a.k.a. cause-effect chain)). A sequence of data-dependent tasks $\alpha \subseteq \Gamma^+$ is a *task chain* only if for every two consecutive tasks τ and τ' in α, $\tau \in Pred(\tau')$.

We denote the set of all given task chains in a DPG with A. For every $\alpha \in A$, we let $|\alpha|$ denote the number of tasks in α. For brevity, we let α_k ($1 \leq k \leq |\alpha|$) denote the kth task of task chain α. The example in Fig. 2 contains four task chains defined by the user, $A = \{\langle\tau_6, \tau_4, \tau_5\rangle, \langle\tau_1, \tau_3, \tau_7, \tau_8, \tau_9\rangle, \langle\tau_2, \tau_3, \tau_7, \tau_8, \tau_9\rangle, \langle\tau_6, \tau_4, \tau_5, \tau_8, \tau_9\rangle\}$ (taken from [40]).

Further, we consider that $Pred(\tau, \alpha)$ returns the immediate predecessor of τ in task chain α and is defined as follows:

$$Pred(\tau, \alpha) = \{\tau' \mid \exists n, 1 \leq n \leq |\alpha|, \tau' = \alpha_n \wedge \tau' \in Pred(\tau)\}. \quad (1)$$

Eq. (1) returns an empty set when τ is the first task of the chain.

Schedule. A *schedule* $\mathcal{S}$ is an assignment of given jobs $\mathcal{J}$ to time intervals. More formally, a schedule $\mathcal{S} = \{(J_1, [s_1, f_1]), \ldots, (J_z, [s_z, f_z])\}$, is a set of tuples where each tuple determines the start time (s_i) and finish time (f_i) of job J_i [9]. In a *feasible* schedule, every job starts no earlier than its earliest release time and completes no later than its deadline [9].

Bounds on the start and finish times of the jobs. Our analysis to derive data age for each task chain requires knowledge about all possible start times and finish times of a job for all feasible schedules. We model the set of these start and finish times each by an interval, denoted by $ST(J) = [EST(J), LST(J)]$ and $FT(J) = [EFT(J), LFT(J)]$, respectively, where $EST(J)$ and $LST(J)$ denote the *earliest and latest start times* and $EFT(J)$ and $LFT(J)$ denote the *earliest and latest finish times* of a job $J \in \mathcal{J}$, respectively.

Obtaining the earliest and latest start times and finish times of a job is challenging when tasks are subject to release jitter and execution-time variations because these timing variations result in having more than one possible schedule (ordering between the execution of the jobs). However, exploiting the recent class of reachability-based response-time analysis techniques, known generally as the schedule-abstraction graphs (SAG) [29–31, 36, 37], it is possible to calculate a tight bound for these values for any possible schedule a system may encounter over the course of its lifetime.

2.1 Data Age for a Single Schedule

As the tasks release jobs periodically, task chains can be instantiated multiple times resulting in so-called *chain instances*. These are formally defined as follows.

Definition 2.2 (Chain Instance). A partial order of jobs β that appear in a schedule $\mathcal{S}$ is a chain instance iff it satisfies the following conditions:

(i) There exists a task chain α that has the same task sequence as β. More formally:

$$\exists \alpha \in A,\ (|\alpha| = |\beta|) \wedge (\forall k, 1 \leq k \leq |\alpha|, Task(\beta_k) = \alpha_k), \quad (2)$$

where the function β_k returns the kth job in the chain instance β.

(ii) For every two consecutive jobs J_l and J_p in the job chain β from the schedule $\mathcal{S}$, we must have:

$$\nexists J_x \in \mathcal{J},\ (Task(J_x) = Task(J_l)) \wedge (f_l < f_x \leq s_p). \quad (3)$$

We call the first job in the chain instance (i.e. β_1) the *source job* and the last job in the chain instance (i.e. $\beta_{|\beta|}$)) *sink job* of β. We also define function $Inst(\alpha)$ which returns the set of all chain instances of task chain α in schedule $\mathcal{S}$.

Delay. Before we introduce data age, we define a delay function $\Delta(J_p, J_c)$ for two arbitrary jobs J_p and J_c, s.t. $f_c > f_p$. It returns the difference between the finish-time of J_c and the release time of J_p that finishes *after* J^c in a schedule $\mathcal{S}$. Namely,

$$\Delta(J_p, J_c) = f_c - r_p^{min}. \quad (4)$$

Data age. Data age determines for how long an input data impacts outputs of a task chain [15]. More formally, the data age of a task chain α is defined as the longest delay between any source job and the latest sink job that is related to that source job via one of the chain instances in the schedule $\mathcal{S}$, namely,

$$DA(\alpha) = \max\left\{\Delta\left(\beta_1, \beta_{|\beta|}\right) \mid \beta \in Inst(\alpha)\right\}. \quad (5)$$

2.2 A Safe Bound on the Observation Window

Our data-age analysis is based on finding the lower and upper bounds of data age for all chain instances that are found in any schedule generated by a set of jobs with timing uncertainties. Therefore, it is crucial for us to ensure that the largest and smallest values of data age will be present (somewhere) among the set of schedules generated for the jobs we consider in our observation window. We form the observation window according to the following criteria: (i) for any shared buffer accessible by a data-consumer task, there must be at least one new data in the buffer produced by a data-producer task (otherwise, we may not see the longest chain instance), (ii) after all shared buffers have been updated at least once by a data-producer task, the OW must be large enough to include all possible patterns in the arrival times of the tasks (namely, it must be larger than one hyperperiod since a hyperperiod is a tight bound on the interval of time after which the arrival pattern of the tasks repeats [9]), (iii) the largest and smallest values of data age should be observable in at least one of the schedules that are built for the jobs in the observation window, and (iv) the OW must be an integer multiple of H, otherwise the schedule-abstraction graph will not give us safe bounds on the WCRT of jobs at the end of OW.

To satisfy (i), we must ensure that each data-producer task is executed at least once, hence, $OW \geq T^{max}$, where $T^{max} = \max\{T_i \mid 1 \leq i \leq n\}$. A simple upper bound on T^{max} is H. To satisfy (ii), the observation window must be as large as $H + T^{max}$ or simply $2H$ (to respect (iv)). Finally, to satisfy (iii), we use a safe

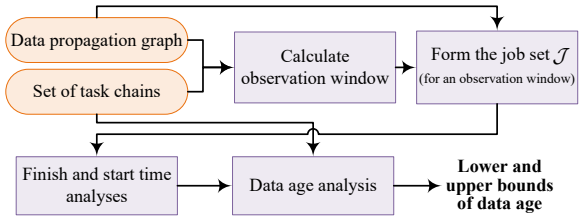

Figure 3: Overview of proposed data age analysis.

upper bound of data age (called the *worst-case data age* (*WCL*)) defined by Becker et al. [6] for a task chain α, as follows:

$$WCL(\alpha) = \sum_{\tau_i \in \alpha} 2 \times T_i. \tag{6}$$

Since $WCL(\alpha)$ might be larger than H, we take the maximum between $2H$ and the largest WCL among all task chains:

$$OW = \max\left\{2H,\ \max_{\forall \alpha \in A}\left\{\left\lceil \frac{WCL(\alpha)}{H} \right\rceil\right\} \times H\right\}. \tag{7}$$

3 DATA-AGE ANALYSIS

Our analysis has two phases: a preparation phase (shown in Fig. 3) and an analysis phase. The preparation phase starts by obtaining the observation window according to Eq. (7). We then form the job set ($\mathcal{J}$) from the jobs of the periodic tasks that could be released within the observation window (i.e., any job which earliest release time is no larger than OW). We extract the start- and finish-time intervals of each job in $\mathcal{J}$ utilizing schedule-abstraction graph [29, 36, 37] since it provides tight bounds on these intervals when tasks are statically assigned to processing elements. This tightness implies that for any job $J \in \mathcal{J}$ and for any time instant $t \in [EST(J), LST(J)]$, there exists a schedule in which J starts at time t, and similarly, for any $t' \in [EFT(J), LFT(J)]$, there exists a schedule in which J finishes at time t'.

We first illustrate the motivation and the main idea of our proposed method with an example (in Sec. 3.1), then we provide the steps of calculating the lower and upper bounds of data age for each task chain in detail (Sections 3.2 to 3.4).

3.1 Motivation and Basic Idea

Consider the task set shown in Fig. 4, scheduled by a fixed-priority non-preemptive scheduling policy on one processing element, where the priority ordering between the tasks is as follows: $\tau_1 < \tau_2 < \tau_4 < \tau_3$. This task set is subject to timing uncertainties (due to execution time variations of tasks τ_2 and τ_3). These uncertainties result in having several possible schedules (shown in Fig. 4(a) and (b)). In Fig. 4(a), all tasks are executed with their worst-case execution time. In this case, two chain instances $\langle J_3, J_8, J_{10}\rangle$ and $\langle J_3, J_8, J_{11}\rangle$ emerge in the schedule where the chain instance $\langle J_3, J_8, J_{11}\rangle$ determines the data age which is equal to 12. However, if task τ_1 executes with its best-case execution time (as shown in Fig. 4(b)), then there would be two chain instances: $\langle J_1, J_7, J_{10}\rangle$ and $\langle J_3, J_8, J_{11}\rangle$. The chain instance $\langle J_1, J_7, J_{10}\rangle$ determines the data age which is equal to 19. This example shows that a task set may have several execution scenarios and schedules. Each of these schedules may result in a different set of chain instances and hence observe a different data age.

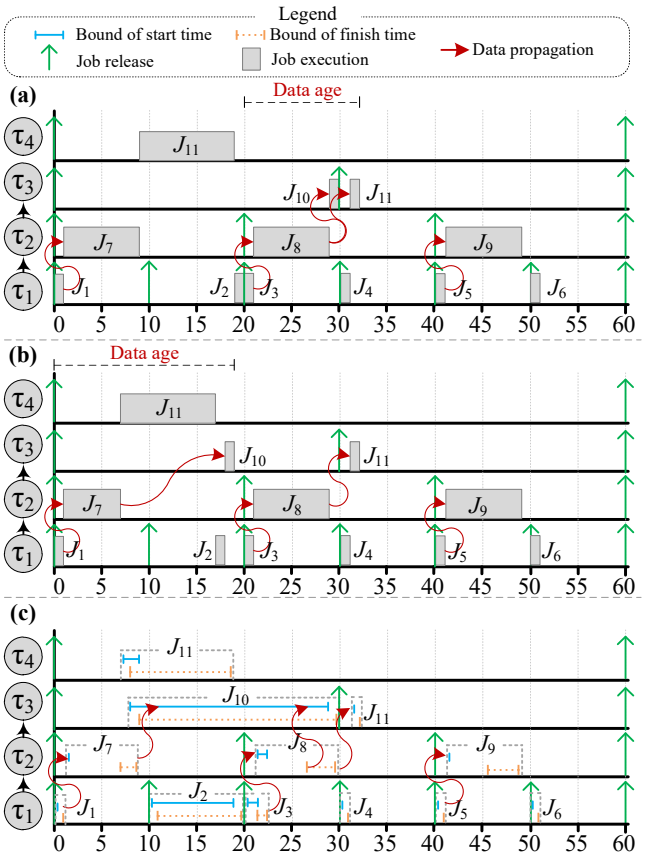

Figure 4: Two different scheduling scenarios of a task set $\Gamma = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ with no release jitter. The execution time range for τ_1 and τ_3 is $[1, 1]$, for τ_2 is $[6, 8]$, and for τ_4 is $[1, 10]$. There is one task chain $\alpha = \langle \tau_1, \tau_2, \tau_3\rangle$. (a) An execution scenario where all tasks execute with their worst-case execution time. (b) An execution scenario where task τ_1 executes with its best-case execution time. (c) The abstraction of all possible schedules with start and finish time intervals for each job.

As we discussed before, in order to find chain instances in the schedule, we need to know the start and finish times of each job. However, as shown by the previous example, one task set can have several schedules resulting in the different start and finish times for each job. Our solution to deal with multiple schedules is to abstract the start time and finish time of a job in all possible schedules that can happen for that job with two intervals (one for all possible start times and one for all possible finish times). Fig. 4(c) shows these intervals for the given task set (derived from the SAG technique [37]).

Using these start-time and finish-time intervals, we can find *potential* data dependency between jobs by identifying jobs of the data producer task that possibly finish before a data consumer job without being overwritten by another job (see Sec. 3.2). By repeating the process of finding the data dependency between jobs of every two consecutive tasks in a task chain, we can form all possible chain instances in the schedules and determine the source jobs in

each chain instance. These chain instances allow us to identify all source jobs for each sink job, and hence, derive a safe bound for the data age (Sec. 3.3). Furthermore, since intermediate chain instances have no impact on the lower and upper bounds of the data age, the intermediate possible data producer jobs in each iteration that form these ineffective chain instances can be discarded (Sec. 3.4).

3.2 Data Dependency Between Jobs

When a data-consumer job has an uncertain start time, it may have multiple jobs that can produce data for it. Let $J^c \in \mathcal{J}$ be any job of a data-consumer task $\tau^c = Task(J^c)$ that is part of a task chain α. We identify the potential data-producer jobs for J^c by ruling out jobs that *can never produce data* for J^c using the following facts:

Fact 1. *The shared buffer between two consecutive tasks in the task chain keeps only the most recent data written by a job of the data-producer task.*

Fact 2. *A data-consumer job reads data from the shared buffer at the beginning of its execution (according to the implicit communication model [1]).*

According to Fact 1, a data-producer job that finishes just before the earliest start time of J^c, has overwritten the previous data in the shared buffer. We denote this job by $P^{first}(J^c, \alpha)$. None of the jobs of the data-producer task with a release time earlier than $P^{first}(J^c, \alpha)$ can produce data that is used by J^c. For example, in Fig. 5(a), J_1 cannot be a data producer for the job J_6 under any scenario because job J_2 will certainly overwrite the data in the shared buffer after J_1. It is important to note that in this example, J_3 is not the first potential data producer for J_6 since it is possible that J_6 starts its execution before J_3 completes. In that case, J_6 will read the data that is produced by J_2 and not J_3.

According to Fact 2, jobs of a data-producer task that certainly finish after the latest start time of J^c, cannot produce data for J^c. Equivalently, all other jobs of a data-producer task whose finish-time interval intersects with the start-time interval of J^c are potential data producers for J^c. We denote these jobs by $P^{rest}(J^c, \alpha)$. In the example in Fig. 5(a), $P^{rest}(J^c, \alpha)$ includes J_3 and J_4 but not J_5 because it is impossible for J_5 to finish before J_6 starts. Note that a data producer job produces its data only at its finish time.

Obtaining $P^{first}(J^c, \alpha)$. To identify $P^{first}(J^c, \alpha)$, we consider two cases depending on whether the data producer task for τ^c is assigned to another PE (Case 1), or to the same one (Case 2).

Case 1. When the data producer and consumer are not on the same PE, they could execute in parallel without blocking each other. In that case, $P^{first}(J^c, \alpha)$ is a data-producer job J' that certainly finishes before the earliest start time of J^c (i.e., $LFT(J') < EST(J^c)$) and satisfies the following condition:

$$\begin{aligned} &Task(J') = Pred(\tau^c, \alpha) \ \wedge \ PE(J') \neq PE(J^c) \ \wedge \\ &LFT(J') < EST(J^c) \ \wedge \ \nexists J'' \in \mathcal{J}, Task(J'') = Pred(\tau^c, \alpha) \ \wedge \\ &LFT(J') < LFT(J'') < EST(J^c). \end{aligned} \tag{8}$$

In the example in Fig. 5(a), τ_1 and τ_2 are assigned to two different PEs, hence, it is possible for J_3 to run in parallel with J_6. Therefore, J_6 may read data from J_2 if J_3 has not yet finished its execution. That is why $P^{first}(J_6, \alpha) = \{J_2\}$.

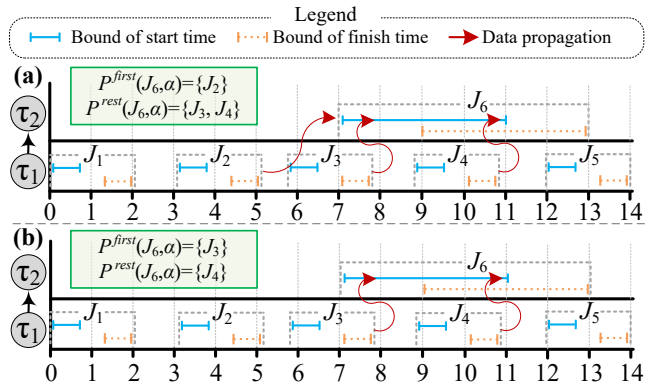

Figure 5: Potential data-producer jobs for a data-consumer job J_6. (a) τ_1 and τ_2 are assigned to different PEs, $P^{first}(J_6, \alpha) = \{J_2\}$ and $P^{rest}(J_6, \alpha) = \{J_3, J_4\}$. (b) τ_1 and τ_2 are on the same PE, $P^{first}(J_6, \alpha) = \{J_3\}$ and $P^{rest}(J_6, \alpha) = \{J_4\}$.

Case 2. When the data producer and data consumer tasks are assigned to the same PE and run non-preemptively, a data-producer job $J' \in \mathcal{J}$ (where $Task(J') = Pred(\tau^c, \alpha)$) could potentially 'block' J^c and hence forces J^c to run after its completion even though there might be an intersection between the finish-time interval of J' and the start-time interval of J^c. In this case, $P^{first}(J^c, \alpha)$ is a data-producer job J' that certainly starts before J^c (namely, $LST(J') < EST(J^c)$) and there is no other job of the data-producer task that certainly starts after J' and before J^c, namely, J' must satisfy the following condition:

$$\begin{aligned} &Task(J') = Pred(\tau^c, \alpha) \ \wedge \ PE(J') = PE(J^c) \ \wedge \\ &LST(J') < EST(J^c) \ \wedge \ \nexists J'' \in \mathcal{J}, Task(J'') = Pred(\tau^c, \alpha) \ \wedge \\ &LST(J') < LST(J'') < EST(J^c). \end{aligned} \tag{9}$$

Consider the example in Fig. 5(b), where τ_1 and τ_2 are assigned to the same PE. Since J_3 is the last data-producer job that certainly starts before J_6 and occupies the PE until it completes (recall that jobs execute non-preemptively), J_3 certainly rewrites the shared buffer, and consequently, the data produced by the previous jobs such as J_2 will be overwritten by J_3. Therefore, in this case, $P^{first}(J_6, \alpha) = \{J_3\}$. Combining the two cases, we have:

$$P^{first}(J^c, \alpha) = \{J' \mid J' \in \mathcal{J} \text{and } J' \text{ satisfies (9) or (8)}\}. \tag{10}$$

Obtaining $P^{rest}(J^c, \alpha)$. Next, we specify other potential data-producer jobs that start after $P^{first}(J^c, \alpha)$ and could possibly finish before the latest start time of J^c, denoted by $P^{rest}(J^c, \alpha)$:

$$\begin{aligned} &P^{rest}(J^c, \alpha) = \{J' \mid J' \in \mathcal{J} \wedge Task(J') = Pred(\tau^c, \alpha) \ \wedge \\ &EST(P^{first}(J^c, \alpha)) < EST(J') \ \wedge (FT(J') \cap ST(J^c) \neq \emptyset)\}. \end{aligned} \tag{11}$$

In summary, the set of potential data-producer jobs for the job J^c are obtained as follow:

$$AP(J^c, \alpha) = P^{first}(J^c, \alpha) \cup P^{rest}(J^c, \alpha). \tag{12}$$

Lemma 3.1. *There is no job $J' \in \mathcal{J} \setminus AP(J^c, \alpha)$ and $Task(J') = Pred(J^c, \alpha)$ that can produce a data for J^c in any feasible schedule.*

Proof. It is trivial to see that a job J' is a potential data-producer for J^c only if it was a job of τ^c's predecessor in task chain α (namely, $Task(J') = Pred(\tau^c, \alpha)$). By contradiction, assume that J' is a data producer for J^c and it does not belong to $AP(J^c, \alpha)$. Since $J' \in \mathcal{J} \setminus AP(J^c, \alpha)$, it must be a job that either (i) finishes before $P^{first}(J^c, \alpha)$ (from Eq. (8) and Eq. (9)) or (ii) finishes after the latest start time of J^c (from Eq. (11)). If J' finishes before $P^{first}(J^c, \alpha)$, then its data in the shared buffer is overwritten by $P^{first}(J^c, \alpha)$ before the start of J^c (according to Fact 1), which is in contradiction to the assumption that J' can be a data producer for J^c. It is trivial to see that if J' certainly finishes after the latest start time of J^c (case (ii)), then by definition, it cannot be a data producer for J^c because the latest start time of J^c, which is the latest time at which it reads data from the buffer, will be before J' starts (i.e., it violates Fact 2). ■

3.3 Finding Possible Source Jobs for a Sink Job

So far, we have discussed the data dependency between two jobs of two consecutive tasks in a task chain. As the data age is defined as the maximum delay of the source and sink jobs (i.e., jobs of the last task in the task chain) in chain instances, finding potential source jobs for each sink job is essential. In the remainder of this section, we discuss how to find source jobs for a sink job by obtaining the potential data dependency between jobs of every two consecutive tasks in a task chain. This is done in Algorithm 1, which is an iterative procedure that starts from one job of the sink task. In each iteration, Algorithm 1 determines the potential data dependency between jobs of two consecutive tasks in a task chain.

In the example in Fig. 6(a), α is a task chain $\langle \tau_1, \tau_2, \tau_3 \rangle$, where the tasks are assigned to different PEs. Fig. 6(a) shows the start and finish time intervals of each job. In order to find possible chain instances, first, we start from jobs of τ_3 due to the fact that they can be sink jobs for chain instances. Note that if no schedule exists in which a data-consumer job J^c of a task at the end of the task chain is a part of a chain instance, J^c cannot be a sink job.

In the example in Fig. 6(a), the data-consumer task τ_3 has only one job J_{10}. Eq. (12) provides the set of potential data-producer jobs for J_{10}, i.e., $AP(J_{10}, \alpha) = \{J_6, J_7, J_8\}$ shown in Fig. 6(b). Among these jobs, J_6 comes from Eq. (8) and J_7 and J_8 are from Eq. (11). We repeat the process to find data-producer jobs for J_6, J_7, and J_8. Fig. 6(c) shows the resulting data producers. Now we reach to the source task of the chain, i.e., τ_1 that has no data-producer in this chain. The jobs that we found in this step, form the set $\{J_1, J_2, J_3, J_4\}$ of possible source jobs for sink job J_{10}. Also, as shown in Fig. 6(d) by connecting the data-dependent jobs that we found in two iterations, we can form the possible chain instances.

Algorithm 1 iteratively finds all potential data-producer jobs O that could contribute to the upper or lower bounds of data age for a given sink job which is defined in Algorithm 2 in line 7. This recursive algorithm in each iteration finds all potential data-producer jobs for jobs in C using Eq. (12) and calls itself again with the calculated set of data-producer jobs and the new task chain whose last task was removed (lines 18,19). The recursive algorithm exits when the length of task chain α equals two because it shows that the calculated list of data-producer jobs (O) is the list of potential source jobs and we reached the head of the task chain.

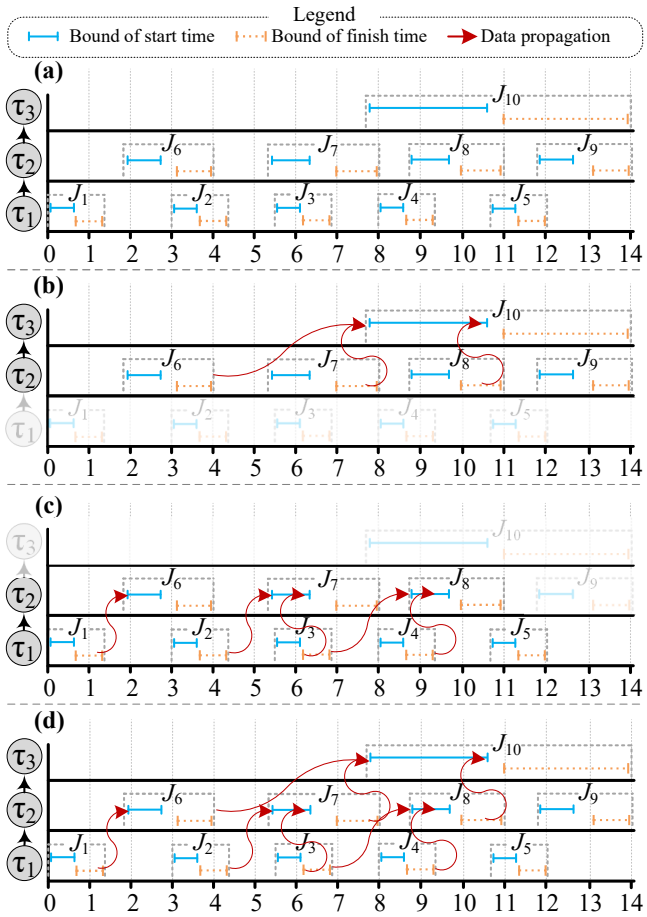

Figure 6: An example of possible chain instances of a task chain $\alpha = \langle \tau_1, \tau_2, \tau_3 \rangle$. (a) start and finish time intervals of each job. (b) all potential data-producer jobs for J_{10}. (c) all potential data-producer jobs for $\{J_6, J_7, J_8\}$. (d) possible chain instances created by connecting potential data-dependent jobs.

Algorithm 2 iterates over all jobs of the sink task in the observation window. For each job, it gets the effective source jobs from Algorithm 1. If the set is not empty, it draws one upper and lower bound on data age. These bounds are kept in DA^{min} and DA^{max} variables that are initialized in line 1 and updated in lines 9-10. If the task chain has only one task, the data age bounds are the same as the response-time bounds of the task, which can directly be obtained from the schedule-abstraction graph in line 3.

Lemma 3.2. *Algorithm 1 (excluding line 14) returns all potential source jobs of any given job J_i of the sink task in a task chain α.*

Proof. Proof trivially follows Lemma 3.1. Let C be the set of data-consumer jobs in iteration k of the Algorithm 1. For each job $J^c \in C$, the algorithm uses Eq. (12) to find all possible data producers in any feasible schedule (proven in Lemma 3.1). Moreover, since the threshold L that is passed to the Algorithm 1 is equal to the latest start time of job J_i, and since a data consumer job (here, the sink

Algorithm 1: Find potential source jobs for a job

Input: Job set of data consumers C, Start time upper bound L, Task chain α.
Output: Potential source jobs O.

```
1  Function findSources(C, L, α):
2      O ← {};
3      if C = ∅ then
4          return ∅;
5      foreach job J ∈ C do
6          Obtain AP(J, α) from Eq. (12);
7          O ← O ∪ AP(J, α);
8      foreach job J ∈ O do
9          if U ≤ EST(J) then
10             O ← O \ {J};
11     if |O| > 2 then
12         J^min ← The job with the smallest r^min in O;
13         J^max ← The job with the largest r^max in O;
14         O ← {J^min} ∪ {J^max};
15     if |α| = 2 then
16         return O ; // the exit condition of the recursive algorithm
17     else
18         α' ← ⟨α_1, α_2, ..., α_(|α|−1)⟩;
19         return findSources(O, L, α');
```

job) can only read data at its start time, none of the data-producer jobs in O whose earliest start time is larger than L could possibly be a data-producer for J_i. Hence, line 7 does not remove any possible data producer from the output of the algorithm. ■

3.4 Pruning Ineffective Chain Instances

As discussed before, a specific sink job can receive data via several possible chain instances which in turn cause the sink job to reach several possible source jobs (in different schedules). However, since we are only interested in finding lower and upper bounds of data age, we can prune the data-producer jobs that do not affect these two bounds. For example, in Fig. 6(d) the sink job J_{10} could have four possible source jobs J_1, J_2, J_3, and J_4. However, the delay generated from J_1 is larger than that of J_2, J_3, and J_4, and hence only J_1 may contribute to finding an upper bound for data age.

Following this logic, we can skip the other jobs and only keep track of the job that has the earliest release time among the potential data-producer jobs. In the remainder of this section, first, we adjust the delay definition with respect to our start and finish time intervals and then we explain how to obtain the bounds of data age without processing intermediate possible data-producer jobs.

Recall that data age is a function of delay (Eq. (4)) between source and sink jobs in a chain instance. To be able to use the delay function with our finish-time intervals obtained from the schedule-abstraction graph, we derive an upper and lower bounds of delay that are valid for any feasible schedule:

$$\widehat{\Delta}^U(J_p, J_c) = LFT(J_c) - r_p^{min}, \tag{13}$$

$$\widehat{\Delta}^L(J_p, J_c) = \max\{(EFT(J_c) - r_p^{min}), 0\}, \tag{14}$$

Algorithm 2: Deriving the data age for a task chain

Input: Job set $\mathcal{J}$ start and finish times bound, Task chains α
Output: Bounds on the data age for a task chain

```
1  DA^min ← ∞, DA^max ← 0;
2  if |α| = 1 then
3      [DA^min, DA^max] ← [BCRT, WCRT] of task α(1);
4  else
5      Sort J by jobs' r^min in non-decreasing order;
6      foreach job J_i ∈ J that Task(J_i) = α_(|α|) do
           // Find possible source jobs using Algorithm 1.
7          O ← findSources({J_i}, LST(J_i), α);
8          foreach job J_o in O do
9              DA^min ← min{DA^min, Δ̂^L(J_o, J_i)};
10             DA^max ← max{DA^max, Δ̂^U(J_o, J_i)};
11 return DA^min, DA^max;
```

since by definition, $LFT(J^c)$ ($EFT(J^c)$) is an upper (lower) bound on the finish time of J^c in any feasible schedule, it is also an upper (lower) bound of f_c in Eq. (4). Note that in Eq. (14), although based on the definition of delay for a single schedule, (Eq. (4)), J_c should finish after J_p. However, since we have only an interval of finish times (instead of the exact finish time of J^c), we must ensure that J_c does not finish after J^p even if $EFT(J^c) < r_p^{min}$ since it must hold that J^p finishes before J^c. As a result, we ensure that the lower bound is the maximum of zero (an obvious lower bound) and the difference between the EFT of J_c and the earliest release time of J_p.

In each iteration of Algorithm 1, our pruning technique (lines 11-14) identifies two special jobs of the data-producer task in the job set O; the one that has the earliest arrival time (J^{min}) and the one that has the latest arrival time (J^{min}). Then it discards the rest of the jobs since they have no impact on the bounds of data age.

Let J_a, $J_b \in O$, $r_a^{min} < r_b^{min}$ be two jobs of a data producer task obtained by Algorithm 1 in lines 2 to 10. Let $\bar{r}^{min}(J) = \min\{r_i^{min} \mid J_i \in AP(J, \alpha)\}$ return the earliest release time among all potential data-producer jobs of an input job J in chain α. The following lemma establishes that $\bar{r}^{min}(J_a) \leq \bar{r}^{min}(J_b)$, and hence, the delay calculated from the data-producers of J_b will be smaller than the delay calculated by the data producers of J_a. Hence, when finding an upper bound on data age, J_b can be ignored.

Lemma 3.3. *For every two jobs of a data-consumer task $J_a, J_b \in O$, $r_a^{min} < r_b^{min}$, we have $\bar{r}^{min}(J_a) \leq \bar{r}^{min}(J_b)$.*

Proof. The proof follows the definition of P^{first} (Eq. (10)). The first potential data producer is a data-producer job whose LFT (or LST, in Case 2) is smaller than the EST of a data-consumer job. Since $r_a^{min} < r_b^{min}$, $EST(J_a) < EST(J_b)$. Hence, the $P^{first}(J_a, \alpha) \leq P^{first}(J_b, \alpha)$. This concludes the claim. ■

Lemma 3.4. *The pruning step of Algorithm 1 (in lines 11 – 14) does not change the bounds of data age calculated by Algorithm 2.*

Proof. Let J^c be the sink job for which Algorithm 1 was called in line 7 of Algorithm 2. Furthermore, let J^{min} and J^{max} be the jobs with the smallest and largest release time in O.

By contradiction, assume that J' is a source job for J^c that was not in O and leads to a larger upper bound for the delay, namely, $\widehat{\Delta}^U(J', J^c) > \widehat{\Delta}^U(J^{min}, J^c)$. From Eq. (13), which is used in line 10 of Algorithm 2, it is clear that such a relation holds only if J' has a smaller release time than J^{min}. However, according to Lemma 3.3, we know that none of the pruned jobs in any intermediate level of the task chain (i.e., during any iteration of Algorithm 1) could result in a data-producer job that has an earlier release time than J^{min}. This leads to a contradiction. The proof for the lower bound follows the same reasoning. ■

Theorem 3.5. *For any task chain α in a schedulable task set Γ scheduled by a non-preemptive JLFP scheduling policy, the data age of task chain α is never larger than DA^{max} (the upper bound returned by Algorithm 2), and is never smaller than DA^{min} (the lower bound returned by Algorithm 2).*

Proof. From the discussions in Sec. 2.2, we know that the job set provided for the schedule-abstraction graph as well as Algorithm 2 includes all job-arrival patterns that could potentially contain the schedule that results in the largest (smallest) data age, respectively. From schedule-abstraction graph analysis, we also know that the start-time and finish-time intervals obtained for each job are tight.

Based on the data age definition (Eq. (5)), the upper (lower) bound of data age are safe only if no source and sink jobs in one of the chain instances of a feasible schedule have a larger (smaller) delay than the upper (lower) bound of the data age reported by Algorithm 2.

Lemmas 3.2 and 3.4 establish that for any given job J_i of a sink task in a task chain, Algorithm 1 returns the source jobs with the earliest and latest release time that could potentially provide data for J_i. Moreover, J_i cannot have any other source job that has not been returned by Algorithm 1 and has an earlier release time (or a later release time) than the jobs returned by Algorithm 1. Finally, given that lines 8 to 10 update the lower-bound and upper bound of data age for any job that could potentially impact these two bounds, we conclude that the bounds of data age calculated by Algorithm 2 are safe, namely, data age of a task chain α is never larger than DA^{max} and never smaller than DA^{min}. ■

3.5 Discussions

Supporting preemptive execution. By assuming that there exists a response-time analysis that provides the start and finish time intervals of each preemptive job [18, 38], our analysis can be adapted to the preemptive task model with a few modifications. Since in the preemptive model, higher priority jobs can preempt a lower priority job, Eq. (9) does not hold and the first potential data-producer job for a job should always be calculated using Eq. (8).

Tightness of the analysis. As explained in Sec. 3, the start- and completion-time intervals obtained from the schedule-abstraction graph analysis of Nasri et al. [29] are tight. However, this does not mean that the schedules in which a data-producer completes at time t are the same schedules in which a data consumer starts at time $t + 1$. As a result, the bounds we obtain are not tight as they may still rely on certain infeasible schedules. In Sec. 4 we will show that despite this pessimism, our analysis effectively reduces the over-estimation of data age compared to the state of the art.

Computational complexity. In our analysis, there are two factors that influence time complexity; (i) the number of jobs in the observation window ($|\mathcal{J}|$) and (ii) the length of the task chain to analyze ($|\alpha|$). Algorithm 1 calls itself at most $|\alpha|$ times (where, $|\alpha|$ is the length of task chain). In each iteration, the complexity of both for-loops (in lines 5 and 8) is $O(|\mathcal{J}|)$ because in the worst-case, the number of data-consumer jobs (C) and the number of jobs in O is about the total size of the job set. Hence, the complexity of Algorithm 1 is $O(|\alpha| \cdot |\mathcal{J}|)$. Algorithm 2 determines the source jobs for jobs of the last task in the task chain. In which the number of jobs of the last task in the task chain is Y, where $Y = OW/T_x$, OW is the observation window and T_x is the period of last task in the task chain. Therefore, Algorithm 2 calls Algorithm 1 for Y times. Putting these together, the complexity of our analysis for a given task chain α is of $O(|\alpha| \cdot |\mathcal{J}| \cdot Y)$. Since the number of jobs in a hyperperiod could be exponential to the number of tasks, our method is pseudo-polynomial w.r.t. the number of tasks. The complexity could be improved by deriving a direct equation to obtain only the job with the earliest and latest release in P^{rest} (since the pruning step will discard the intermediate jobs).

4 EMPIRICAL EVALUATION

We conduct experiments on an industrial case study (Sec. 4.1) as well as synthetic task sets (Sec. 4.2) to answer the following questions: **(i)** does our analysis reduce the pessimism of derived bounds of data age compared to the state of the art? And **(ii)** is the runtime of our analysis practical?

4.1 Case Study

We considered an automotive application [22, 40] (detailed in Fig. 2 and Table 1) as a case study. We compare our method with Becker et al. [6] and Becker et al. [8]. The former paper does not incorporate scheduling policy information and the latter paper relies on WCRTs.

We considered the observation window of 350ms, extracted the jobs within this window, and calculated their start and finish time intervals using the schedule-abstraction graph [29] for the EDF scheduling policy. The obtained response-times are also used for Becker et al. [8] method which needs information about the task response times. However, since these baselines do not assume any uncertainty in the execution time of the tasks, we considered two setups for the task sets in Table 1: one with execution-time variation and one without. The second and third columns of Table 2 show the result of our algorithm in these two setups. For the method of Becker, we only used the setup without execution-time variation.

As shown in Table 2, our method reduces the pessimism of data age bounds for the given task chains by up to 30% (27.65% on average for the four task chains) in comparison to Becker et al. [8].

4.2 Evaluation via Synthetic Task Sets

We design four experiments to study the impact of system utilization (U), the number of processing elements (m), the number of tasks in the task sets (n), and the length of task chains on the effectiveness of our solution. In each experiment, we generate 100 schedulable task sets for each data point reported in Fig. 7.

Task set generation. To generate a periodic task set with n tasks, we select n period values based on the probability distribution

Task chain	This paper	This paper (BCET=WCET)	Becker et al. [6] (schedule agnostic)	Becker et al. [8] (using WCRT)	Improvement (%) (w.r.t. Becker et al. [8])
Task chain 1: $\langle \tau_6, \tau_4, \tau_5 \rangle$	[68.9, 75]	[75, 75]	[75, 125]	[75, 100]	25%
Task chain 2: $\langle \tau_1, \tau_3, \tau_7, \tau_8, \tau_9 \rangle$	[61.8, 114.5]	[74.5, 114.5]	[59.5, 190]	[64.5, 164.5]	30.3%
Task chain 3: $\langle \tau_2, \tau_3, \tau_7, \tau_8, \tau_9 \rangle$	[61.8, 114.5]	[74.5, 114.5]	[59.5, 190]	[64.5, 164.5]	30.3%
Task chain 4: $\langle \tau_6, \tau_4, \tau_5, \tau_8, \tau_9 \rangle$	[81.8, 134.5]	[94.5, 134.5]	[84.5, 185]	[94.5, 179.5]	25%

Table 2: Data age bounds for task chains of the data propagation graph in the automotive application described by Fig. 2.

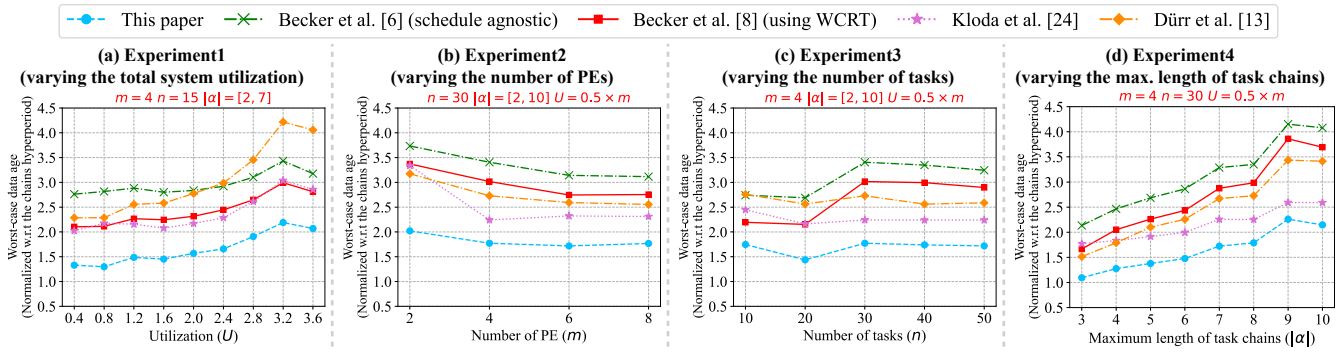

Figure 7: Normalized data age (averaged over all chains) as a function of (a) varying utilization (Exp1), (b) varying number of PEs (Exp2), (c) varying number of tasks (Exp3), and (d) varying length of task chains (Exp4).

of periods of automotive benchmark applications [26]. Using the *RandFixedSum* method of Emberson et al. [14], we generate a set of n random utilizations that sum to the target system utilization. We assumed the system is scheduled by the EDF policy, hence, the priority of each job is equal to its absolute deadline. Moreover, to be able to compare our results to other work in the state of the art, we assume that tasks do not have release jitter (since none of the methods we compare against supports release jitter). After generating the tasks, we assign them to PEs using the worst-fit partitioning. We consider that PEs are identical since our analysis is not affected by the type of PEs (as migration is not allowed).

DPG generation. To generate random data-propagation graphs, we go through each task and randomly assign an edge from that task to another task with a probability of 0.4. We made sure that the edges do not form a cycle by processing the tasks in a random but consistent order and adding edges in one direction. We limit the maximum branching factor of a node (i.e., the number of immediate successors of a node) to 4, the maximum number of input nodes to 5, and the maximum depth of the graph to 10 to form task graphs that are not too far in terms of structure from the ones of our case study. A generated DAG contains between two and fifteen task chains, and a task could be a part of several task chains at once.

Baselines. We compare our analysis to the schedule-agnostic data-age analysis of Becker et al. [6], a recent work of Becker et al. [8] that uses WCRT information, the data age analysis of Dürr et al. [13] that uses WCRT information, and the work of Kloda et al. [24], which obtains the reaction latency (an upper bound on data age). To have a fair comparison between the three methods [8, 13, 24] which require information about the WCRT of the jobs, we use the accurate WCRTs derived by the SAG analysis (with partial-order reduction) [36, 37] which is the most accurate existing job-level response time analysis method.

Metrics. We report the upper bound of the data age of each task chain normalized by the hyperperiod of the task chain, namely, it is divided by the least common multiple of the periods of the tasks in the chain. The average values of normalized data age are then shown in Fig. 7. Later in Fig. 8, we report the runtime (CPU time) of our method for two experiments that focus on larger system sizes.

Execution platform. We instrumented the open-source code of schedule-abstraction graph approach [31, 32] to also report the start-time intervals per job, in addition to the finish-time intervals. We performed the experiments on Surfsara, the Dutch national high-performance computing cluster on AMD EPYC 7601 processors clocked at 2.2GHz with 1TB of RAM.

Experiment1 (impact of system utilization). We consider a system with four PEs and 15 tasks per task set. We vary the total utilization U from 0.4 to 3.6 in steps of 0.4.

Fig. 7(a) shows the impact of system utilization on the average data age. Despite the fact that the data age increases as the utilization increases, our method is able to produce significantly less-pessimistic upper bounds on data age than the state of the art. Namely, it reduces the bounds by up to 38% and on average by 31.8% compared to Becker et al. [8]. In comparison to Kloda's method [24], we improve the tightness by up to 40.3%, and on average 30%. This experiment also shows that the method of Dürr et al. [13] is sensitive to system utilization. For example, for $U = 0.4$ Dürr et al. [13] is 8% more pessimistic than Becker et al. [8], while at $U = 3.6$, this difference grows to 30%.

Experiment2 (impact of the number of PEs). In this experiment, we vary the number of PEs (m) from 2 to 8. We keep system

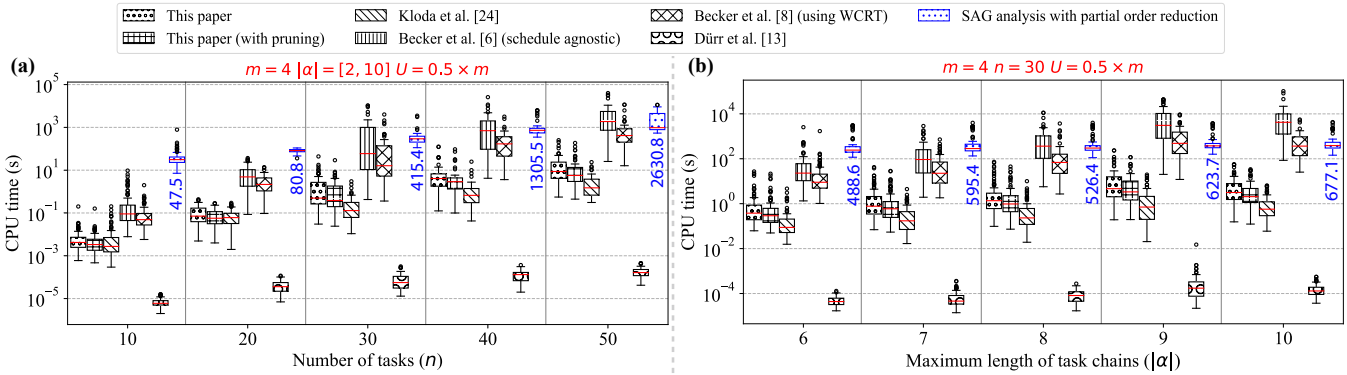

Figure 8: CPU time as a function of (a) the number of tasks, and (b) the maximum length of task chains.

utilization at $m \times 0.5$ and the number of tasks in a task set at $n = 30$. Fig. 7(b) shows that data age is not significantly affected by the number of PEs. This is mainly due to the fact that tasks are preassigned to PEs and hence are isolated from the rest of the system. A larger data age in cases where there are fewer PEs (2 and 4) is due to the blocking effect; tasks run non-preemptively and hence block each other which results in a larger WCRT and data age. Our method improves the accuracy of calculating data age by 38.7% in comparison to Becker [8] and 28.7% in comparison to Kloda [24].

Experiment3 (impact of the number of tasks). We vary the number of tasks from 20 to 50, while assuming a system with $m = 4$ PEs and the utilization equal to $m \times 0.5$. As shown in Fig. 7(c), the increase in the number of tasks has only a small impact on the data age of our method while it has a drastic impact on Becker's methods [6, 8]. This is mainly due to the appearance of longer task chains in the DPG with more tasks (we will investigate this further in Experiment4). Overall, we reduce the over-estimation of data age by 36.5%, 36.2%, and 25.7% on average, in comparison to Becker et al. [8], Dürr et al. [13], and Kloda et al. [24] methods, respectively.

Experiment4 (impact of the task chain's length). We vary the maximum length of task chains from 3 to 10 for a system with $m = 4$ PEs, $n = 30$ tasks, and $U = m \times 0.5$. As shown in Fig. 7(d), increasing the length of task chains significantly impacts the pessimism of data age obtained by both methods of Becker et al. [6, 8] and that of Dürr et al. [13]. For example, for $|\alpha| = 3$, our analysis is 34.3% less pessimistic than that of Becker et al. [8], while at $|\alpha| = 10$, this difference grows to 41.8% (Becker's method becomes much more pessimistic). This is due to the fact that the Becker method does not take the start-time intervals into account. Consequently, in each iteration where it finds data-dependent jobs, it may include impossible cases, resulting in more pessimistic bounds. In conclusion, our method reduces the over-estimation of data age by 39.7%, 33.9%, and 23.6% on average in comparison to Becker et al. [8], Dürr et al. [13], and Kloda et al. [24] methods, respectively.

Experiment5 (runtime). In this experiment, we measure the runtime (CPU time) of data age analysis for each method under two setups: **(i)** varying the number of tasks from 20 to 50 for a system with $m = 4$ PEs and the utilization of $m \times 0.5$, **(ii)** varying the maximum length of task chains for a system with $m = 4$ PEs, the utilization equal to $m \times 0.5$, and $n = 30$ tasks.

As our method, and three other methods [8, 13, 24] that require the WCRT information of the tasks have the same overhead of a method that provides the response-time bounds for each job, we separately reported the runtime of the SAG analysis with partial-order reduction [36, 37] in order to get more accurate insight into the runtime of data-age analysis methods. It is worth mentioning that, we selected SAG analysis because it is providing the most accurate response-time bounds for each job. Though our method is slightly slower than Kloda's [24] and Dürr's [13] methods, our method obtains a far less pessimistic upper bound on data age. Furthermore, our experiments show that even with the SAG's runtime, the average runtime of our solution in setup (i) is 902 seconds which is 1.8 times faster than Becker et al. [6] method.

5 CONCLUSION

In this paper, we presented a novel data-age analysis of multi-rate task chains with the presence of uncertainties in the release time and execution time of the tasks. We demonstrated these uncertainties by timing intervals on the start-time and completion-time of the jobs in an observation window. This enabled us to develop a new analysis that explores possible data dependencies between jobs for a collection of schedules (instead of only one, as opposed to the literature). To speed up the analysis, we introduced pruning rules to rule out jobs that would not impact the bounds of data age.

Using an industrial case study together with synthetic task sets, we demonstrated that our method reduces over-estimations of data age of all experiments between 30% and 40% compared to the state-of-the-art analyses. Moreover, unlike previous approaches, our method scales well in the length of task chains. We found that some of the existing analyses of data age [6, 8, 13] perform worse than the upper bound derived from the reaction latency (calculated by the Kloda method [24]). As such, our work delivers the first direct analysis of data age, deriving tighter upper and lower bounds than previous approaches. In the future, we intend to extend our analysis to include more complex timing constraints that require the consideration of multiple chains simultaneously (e.g., when input delivery to a set of chains must occur contemporaneously).

ACKNOWLEDGMENTS

This work made use of (i) the Dutch national e-infrastructure with support of the SURF Cooperative under grant agreement no EINF-2663, (ii) the EU ECSEL Joint Undertaking under grant agreement no 101007260 (project TRANSACT), and (iii) funding from the Eindhoven Artificial Intelligence Systems Institute (EAISI), Eindhoven University of Technology, PO Box 513, the Netherlands.

REFERENCES

[1] 2014. AUTOSAR - specification of timing extensions. https://www.autosar.org/
[2] Jakaria Abdullah, Gaoyang Dai, and Wang Yi. 2019. Worst-case cause-effect reaction latency in systems with non-blocking communication. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1625–1630.
[3] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I. Davis. 2020. An Empirical Survey-based Study into Industry Practice in Real-time Systems. In *Real-Time Systems Symposium (RTSS)*. 3–11.
[4] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I Davis. 2021. A comprehensive survey of industry practice in real-time systems. *Real-Time Systems Journal* (2021), 1–41.
[5] Neil Audsley, Alan Burns, Mike Richardson, Ken Tindell, and Andy J Wellings. 1993. Applying new scheduling theory to static priority pre-emptive scheduling. *Software engineering journal* 8, 5 (1993), 284–292.
[6] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. 2016. Synthesizing job-level dependencies for automotive multi-rate effect chains. In *Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 159–169.
[7] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. 2017. End-to-end timing analysis of cause-effect chains in automotive embedded systems. *Journal of Systems Architecture* 80 (2017), 104–113.
[8] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. 2018. Analyzing end-to-end delays in automotive systems at various levels of timing information. *ACM SIGBED Review* 14, 4 (2018), 8–13.
[9] Giorgio C Buttazzo. 2011. *Hard real-time computing systems: predictable scheduling algorithms and applications* (third ed.). Springer Science & Business Media.
[10] Hyunjong Choi, Mohsen Karimi, and Hyoseung Kim. 2020. Chain-Based Fixed-Priority Scheduling of Loosely-Dependent Tasks. In *International Conference on Computer Design (ICCD)*. 631–639.
[11] Hyunjong Choi, Yecheng Xiang, and Hyoseung Kim. 2021. PiCAS: New design of priority-driven chain-aware scheduling for ROS2. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 251–263.
[12] Abhijit Davare, Qi Zhu, Marco Di Natale, Claudio Pinello, Sri Kanajan, and Alberto Sangiovanni-Vincentelli. 2007. Period optimization for hard real-time distributed automotive systems. In *Annual Design Automation Conference*. 278–283.
[13] Marco Dürr, Georg Von Der Brüggen, Kuan-Hsun Chen, and Jian-Jia Chen. 2019. End-to-end timing analysis of sporadic cause-effect chains in distributed systems. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 5s (2019), 1–24.
[14] Paul Emberson, Roger Stafford, and Robert I Davis. 2010. Techniques for the synthesis of multiprocessor tasksets. In *Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. 6–11.
[15] Nico Feiertag, Kai Richter, Johan Nordlander, and Jan Jonsson. 2009. A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In *Real-Time Systems Symposium (RTSS)*.
[16] Julien Forget, Frédéric Boniol, David Lesens, and Claire Pagetti. 2008. A multi-periodic synchronous data-flow language. In *High Assurance Systems Engineering Symposium*. 251–260.
[17] Simon Fürst, Jürgen Mössinger, Stefan Bunzel, Thomas Weber, Frank Kirschke-Biller, Peter Heitkämper, Gerulf Kinkelin, Kenji Nishikawa, and Klaus Lange. 2009. AUTOSAR–A Worldwide Standard is on the Road. In *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, Vol. 62. 5.
[18] Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. 2009. New response time bounds for fixed priority multiprocessor scheduling. In *Real-Time Systems Symposium (RTSS)*. 387–397.
[19] Mario Günzel, Kuan-Hsun Chen, Niklas Ueter, Georg von der Brüggen, Marco Dürr, and Jian-Jia Chen. 2021. Timing Analysis of Asynchronized Distributed Cause-Effect Chains. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 40–52.
[20] Nicholas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. 1991. The synchronous data flow programming language LUSTRE. *Proc. IEEE* 79, 9 (1991), 1305–1320.
[21] Arne Hamann, Dakshina Dasari, Simon Kramer, Michael Pressler, and Falk Wurst. 2017. Communication Centric Design in Complex Automotive Embedded Systems. In *Euromicro Conference on Real-Time Systems (ECRTS) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 76)*. 10:1–10:20.
[22] Arne Hamann, Dakshina Dasari, Falk Wurst, Ignacio Sañudo, Nicola Capodieci, Paolo Burgio, and Marko Bertogna. 2019. WATERS Industrial Challenge 2019. http://archives.ecrts.org/fileadmin/WebsitesArchiv/ecrts2019/waters/waters-industrial-challenge/index.html
[23] Tobias Klaus, Matthias Becker, Wolfgang Schröder-Preikschat, and Peter Ulbrich. 2021. Constrained Data-Age with Job-Level Dependencies: How to Reconcile Tight Bounds and Overheads. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 66–79.
[24] Tomasz Kloda, Antoine Bertout, and Yves Sorel. 2018. Latency analysis for data chains of real-time periodic tasks. In *Conference on Emerging Technologies and Factory Automation (ETFA)*, Vol. 1. 360–367.
[25] Tomasz Kloda, Antoine Bertout, and Yves Sorel. 2020. Latency upper bound for data chains of real-time periodic tasks. *Journal of Systems Architecture* 109 (2020), 101824.
[26] Simon Kramer, Dirk Ziegenbein, and Arne Hamann. 2015. Real world automotive benchmarks for free. In *Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*.
[27] Shaoshan Liu, Bo Yu, Nan Guan, Zheng Dong, and Benny Akesson. 2021. RTSS 2021 Industry Session. http://2021.rtss.org/industry-session/
[28] Swarup Mohalik, Devesh B Chokshi, Manoj G Dixit, AC Rajeev, and S Ramesh. 2013. Scalable model-checking for precise end-to-end latency computation. In *Conference on Computer Aided Control System Design (CACSD)*. 19–24.
[29] Mitra Nasri and Björn B Brandenburg. 2017. An exact and sustainable analysis of non-preemptive scheduling. In *Real-Time Systems Symposium (RTSS)*. 12–23.
[30] Mitra Nasri, Geoffrey Nelissen, and Björn B Brandenburg. 2018. A response-time analysis for non-preemptive job sets under global scheduling. In *Euromicro Conference on Real-Time Systems (ECRTS)*. 9–1.
[31] Mitra Nasri, Geoffrey Nelissen, and Björn B Brandenburg. 2019. Response-time analysis of limited-preemptive parallel dag tasks under global scheduling. In *Euromicro Conference on Real-Time Systems (ECRTS)*. 21–1.
[32] Mitra Nasri, Geoffrey Nelissen, and Björn B Brandenburg. 2019. Schedulability tests for non-preemptive job sets. https://github.com/gnelissen/np-schedulability-analysis
[33] Claire Pagetti, Julien Forget, Frédéric Boniol, Mikel Cordovilla, and David Lesens. 2011. Multi-task implementation of multi-periodic synchronous programs. *Discrete event dynamic systems* 21, 3 (2011), 307–338.
[34] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, Vol. 3. 5.
[35] A. C. Rajeev, Swarup Mohalik, Manoj G. Dixit, Devesh B. Chokshi, and S. Ramesh. 2010. Schedulability and End-to-End Latency in Distributed ECU Networks: Formal Modeling and Precise Estimation. In *International conference on Embedded software (EMSOFT)*. 129–138.
[36] Sayra Ranjha, Mitra Nasri, and Geoffrey Nelissen. 2021. Work-in-Progress: Partial-Order Reduction in Reachability-based Response-Time Analyses. In *Real-Time Systems Symposium (RTSS)*. 544–547.
[37] Sayra Ranjha, Geoffrey Nelissen, and Mitra Nasri. 2022. Partial-Order Reduction for Schedule-Abstraction-based Response-Time Analyses of Non-Preemptive Tasks. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*.
[38] Ola Redell and Martin Törngren. 2002. Calculating exact worst case response times for static priority scheduled tasks with offsets and jitter. In *Real-Time and Embedded Technology and Applications Symposium*. 164–172.
[39] Johannes Schlatow, Mischa Mostl, Sebastian Tobuschat, Tasuku Ishigooka, and Rolf Ernst. 2018. Data-Age Analysis and Optimisation for Cause-Effect Chains in Automotive Control Systems. In *Symposium on Industrial Embedded Systems (SIES)*. 1–9.
[40] Micaela Verucchi, Mirco Theile, Marco Caccamo, and Marko Bertogna. 2020. Latency-Aware Generation of Single-Rate DAGs from Multi-Rate Task Sets. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 226–238.