Latency-Aware Generation of Single-Rate DAGs from Multi-Rate Task Sets

Micaela Verucchi*†, Mirco Theile†, Marco Caccamo† and Marko Bertogna*
*University of Modena and Reggio Emilia, Italy
{micaela.verucchi, marko.bertogna}@unimore.it
†Technical University of Munich, Germany
{mirco.theile, mcaccamo}@tum.de

***Abstract*—Modern automotive and avionics embedded systems integrate several functionalities that are subject to complex timing requirements. A typical application in these fields is composed of sensing, computation, and actuation. The ever increasing complexity of heterogeneous sensors implies the adoption of multi-rate task models scheduled onto parallel platforms. Aspects like freshness of data or first reaction to an event are crucial for the performance of the system. The Directed Acyclic Graph (DAG) is a suitable model to express the complexity and the parallelism of these tasks. However, deriving age and reaction timing bounds is not trivial when DAG tasks have multiple rates. In this paper, a method is proposed to convert a multi-rate DAG task-set with timing constraints into a single-rate DAG that optimizes schedulability, age and reaction latency, by inserting suitable synchronization constructs. An experimental evaluation is presented for an autonomous driving benchmark, validating the proposed approach against state-of-the-art solutions.**

***Index Terms*—DAG, multi-rate, end-to-end latency, schedulability.**

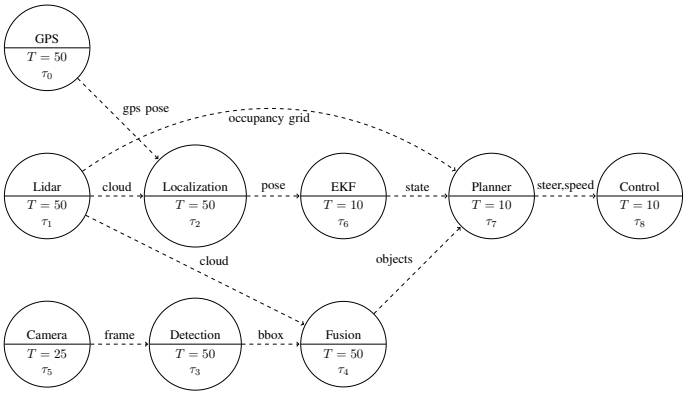


Fig. 1: An example of high-utilization automotive application with tasks at different periods.

I. Introduction

Modern automotive and avionics real-time embedded systems are composed of applications including sensors, control algorithms and actuators to regulate the state of a system in its environment within given timing constraints. Task chains are commonly adopted to model a sequence of steps performed along the control path. Complex data dependencies may exist between task chains with different activation rates, making it very hard to find reliable upper bounds on the end-to-end latency of critical effect chains [1].

This problem is exacerbated by the adoption of even more complex task models based on Directed Acyclic Graphs (DAG) to capture the parallel activation of multiple jobs executing on heterogeneous multi-core platforms. A recent example in the automotive domain is given in the WATERS industrial challenge [2], focusing on the minimization of the end-to-end latency of critical effect chains of an autonomous driving system involving several sensors. The application is modeled in Figure 1, with three sensors providing input to multiple task chains. Nodes represent tasks with different activation periods, while edges represent the exchange of data between tasks, forming effect chains. Reaction to input stimuli and freshness of data are key factors to consider when deploying the application on a selected computing platform. *Data age* quantifies for how long an input data affects an output of a task chain, i.e., it is the maximum delay between a valid sensor input until the *last* output related to that input in the chain. *Data age* constraints are commonly found in control systems, where the age of the data can directly influence the quality of the control. In the considered application, key effect chains to optimize for *data age* are connected to the processing of camera frames and LiDAR point clouds: the older the input, the less precise is the localization of the ego vehicle and the detection of obstacles.

Another key metric to optimize is the *reaction latency*, a parameter that measures the reactivity of the system to a change in the input. It is defined as the maximum delay between a valid sensor input until the *first* output of the event chain that reflects such an input. It measures how much time it takes for a new event to propagate through a chain. In the considered autonomous driving example, key *reaction times* to optimize are the detection of an obstacle in the driving path, and the related actuation on the steering and breaking system to safely avoid it in due time.

The aim of this work is to consider such systems composed of DAG tasks having multiple rates with given constraints on age and reaction latencies. Starting from a high-level representation, a method is presented to create a single-rate DAG that fulfills the given restrictions, optimizing schedulability and end-to-end delays. To do so, a set of DAG candidates is generated and evaluated by a constrained cost function designed to pick the best DAG meeting the given requirements.

The paper is organized as follows: in the next section, an overview of the state-of-the-art is given before introducing the system model of the proposed approach. In Section IV, the mathematical basis of the following sections is derived. Section V gives a detailed explanation of the conversion from the multi-rate task set to the single-rate DAG. Section VI focuses on the requirements: end-to-end latency, schedulability and their evaluation. Finally, we conclude with an experimental part, in which the results of our framework are shown, comparing them with other existing methods.

II. Related Work

A. End-to-end latency

A task chain is a sequence of communicating tasks in which every task receives data from its predecessor. In literature, two types of task chains can be found: periodic chains and event-driven chains [3]. In the former, each task is activated independently at a given rate, and it communicates with its successor by means of shared variables; in the latter, task executions are triggered by an event issued from a preceding task. The propagation delays of a task chain affect responsiveness, performance and stability of an application.

We hereafter focus on the periodic model, which is the most common in the automotive domain [1]. Di Natale et al. [4] proposed a method to evaluate the worst-case latency of mixed chains of real-time tasks and Controller Area Network (CAN) messages. Zeng et al. [5], [6] computed the probability distribution, via statistical analysis, of end-to-end latencies for CAN message chains.

Feiertag et al. [7] were the first to define *data age* and *reaction time* and to propose a framework to calculate end-to-end latencies in automotive systems, where each task operates according to the read-execute-write semantic, also known as the implicit communication model of AUTOSAR [8].

Becker et al. presented in [9] [10] a method to compute worst- and best-case *data age* for periodic tasks with implicit deadlines using implicit, explicit and Logical Execution Time (LET) communication models. The analysis is based on Read Interval (RI) and Data Interval (DI), which respectively are the interval in which a task can possibly read its input data in order to complete its execution before the deadline, and the interval for which the output data of a task can be available to the successor task in the chain. Multiple Data Propagation Trees (DPT) are constructed in order to compute the *data age*. A method is also described to constrain the maximum latency by inserting job-level dependencies. A tool, called MECHAniSer [11], is presented to compute latency values for a given task set. Regarding the LET model, Biondi et al. [12] and Martinez et al. [13] addressed the problem of computing end-to-end latency bounds on multi-cores, improving the results of Becker et al. in [10]. Our paper does not focus on the LET model, but it aims at deriving better latency bounds for the implicit model.

There exist other works that aim at selecting the best periods or deadlines to minimize data age in simpler task models. In [14], this is done on a single core platform, without considering task chains. In [15], the authors propose a method to find the best period to bound data freshness of task chains, assuming the task set given in input be already schedulable. Adapting these solutions to our setting is not trivial, because we assume periods and deadline to be given.

B. Multi-rate DAG

In [16], Saito et al. present a framework developed for the Robot Operating System (ROS) to handle automotive applications with multi-rate tasks. The model assumes an event-driven data-flow system in which a node starts when the predecessor nodes are completed. In order to handle multi-rate tasks, a synchronization system is adopted consisting of two kinds of additional nodes: synch driver nodes and synch nodes. The synch driver node is used to adjust the publishing period of the sensors, buffering the data of the highest rate one, in order to have a node with a unique rate for all the sensors. Synch nodes are then inserted before the tasks to handle buffered data. In this way, a single-rate DAG is obtained and scheduled using a fixed-priority algorithm based on the HLBS scheduler [17].

Forget et al. [18] faced the same problem for autopilot applications, considering periodic tasks modeled as nodes in a DAG with two kinds of edges: simple and extended. Simple edges are precedence constraints between tasks having the same rate, while extended edges are data dependencies between tasks having different rates. To handle extended edges, a method is proposed to generate multiple conversions from extended edges through simple precedence constraints between jobs, selecting a permutation that guarantees Earliest Deadline First (EDF) schedulability.

Another conversion method from a multi-rate DAG to a single-rate one has been proposed by Saidi et al. in [19] for a similar DAG model. The output DAG has a period equal to the hyper-period of the input task set. The nodes are the job instances activated in a hyper-period for each task. Edges are precedence constraints between jobs, which are inserted based on the ratio between the periods of the communicating tasks. A multi-core heuristic is proposed to schedule the DAG, while minimizing a cost function related to task schedulability.

Converting the original task set to a DAG is a very convenient approach that allows seamlessly inserting explicit precedence constraints to control end-to-end latency. To our knowledge, most of the other methods in the literature perform similar conversions to impose such precedence constraints for limiting latency. While the work of Becker [9] may appear different, as it does not explicitly consider DAGs, it ends up implementing a similar approach by inserting precedence constraints between different jobs. In Section VII, we will highlight the differences between the presented methods and our approach.

III. System Model

This work shows how to convert a *Multi-Rate Task set with Constraints* into a *Single-Rate Directed Acyclic Graph* (DAG),

in order to analyze schedulability and end-to-end latency of task-chains.

A. Multi-Rate Task set with Constraints

The input to the proposed method is a task set Γ, modeling an application like the one in Figure 1, composed of N periodic tasks τ_x arriving at time $t = 0$. Each task τ_x is described by the tuple (WC_x, BC_x, T_x, D_x), where:

- $WC_x \in \mathbb{R}$ is the Worst Case Execution Time (WCET) of the task;
- $BC_x \in \mathbb{R}$ is the Best Case Execution Time (BCET);
- $T_x \in \mathbb{N}$ is the period;
- $D_x \in \mathbb{R}$ represents the relative deadline.

The exchange of data between two tasks is modeled with as *data edge*, a directed (dashed) edge between the producer and the consumer of the data. Moreover, precedence constraints may be specified between two tasks (τ_x, τ_y), stating that a job $\tau_{y,b}$ cannot start until all the jobs of τ_x released in τ_y's period completed their execution. For this reason, precedence constraints can be inserted only between tasks having the same period, corresponding to job level precedence constraints.

To constrain the latency of data propagation in task-chains, upper bounds on *data age* and on *reaction time* can be given. The latency constraints evaluation is described in more detail in Section VI.

Our approach is based on a global non-preemptive list scheduling approach, as described in Section VI-B. Such a policy allows different instances of the same task to run on different cores, while preventing a job to be migrated during its execution, mitigating the preemption overhead.

B. Directed Acyclic Graph

The output of the proposed method is a *single-rate Directed Acyclic Graph (DAG)*. Such a model is based on the parallel DAG model proposed by Baruah in [20] to capture the parallelism of a task to be scheduled on a multi-core platform. In this model, tasks are represented as directed acyclic graphs, each with a unique source vertex and a unique sink vertex. Each vertex represents a sequential job, while edges represent precedence constraints between jobs.

In this work, we use a similar model with a semantic difference, i.e., a DAG represents a full application, with each vertex representing a task instance, which we call job. In detail, the DAG is specified by a 3-tuple (V, E, HP) where:

- V represents the set of nodes, namely the jobs of the tasks of Γ, and $n = |V|$;
- E is the set of edges describing job-level precedence constraints;
- HP is the period of the DAG, namely the hyper-period of the tasks involved: $HP = lcm_{\forall \tau_x \in \Gamma}\{T_x\}$.

In this model, the communication between jobs utilizes buffers in shared memory, which can be accessed by all the cores. The time to write/read a shared buffer is included in the execution time of each task. We adopt the implicit communication model defined in AUTOSAR [8], solving mutual exclusion via double-buffering. Each task complies with a read-execute-write semantic, i.e., it reads a private copy before the execution, and it writes a private copy at the end of the execution [1].

C. Notation

For the sake of clarity, a standardized set of indexing names is adopted throughout the entire paper, i.e., $\{i, j, k\}$ denote general nodes in a DAG (jobs, synchronization or dummy nodes), $\{x, y, z\}$ indicate tasks, and $\{a, b, c\}$ are used for jobs.

IV. Background

This part describes the main algorithms used in the following sections. A DAG is represented as an adjacency matrix $\mathbf{T} \in \mathbb{B}^{n \times n}$, in which $T_{i,j} = 1$ iff there exists an edge $e(v_j, v_i)$[1]. Given this Boolean formulation of the DAG, Boolean algebra can be applied. Therefore, the Boolean matrix product is defined as:

$$\mathbf{C} = \mathbf{AB}, \quad \mathbf{A} \in \mathbb{B}^{n \times m}, \mathbf{B} \in \mathbb{B}^{m \times n}, \mathbf{C} \in \mathbb{B}^{n \times n} \tag{1}$$

for which the cells of $\mathbf{C}$ evaluate to

$$c_{i,j} = \bigvee_{k=0}^{m-1} a_{i,k} \wedge b_{k,j} \tag{2}$$

Cell-wise Boolean operations are denoted as $\wedge$ and $\vee$ for *and* and *or*, respectively. Additionally, a maximum matrix multiplication is used in this work to combine Boolean matrices with real matrices. It is defined as

$$\begin{gathered} \mathbf{C} = \mathrm{maxProduct}(\mathbf{A}, \mathbf{B}), \\ \mathbf{A} \in \mathbb{B}^{n \times m}, \mathbf{B} \in \mathbb{R}^{m \times n}, \mathbf{C} \in \mathbb{R}^{n \times n} \end{gathered} \tag{3}$$

where the cells of $\mathbf{C}$ are calculated as

$$c_{i,j} = \max_{k \in \{0,\dots,m-1\}} \{a_{i,k} b_{k,j}\}. \tag{4}$$

A. Transitive Closure

The proposed scheduling method and the related end-to-end latency computation make use of the mathematical principles of graph theory [21]. One principle is the transitive closure [22] of a DAG, defined as

$$\mathbf{D} = \bigvee_{k=1}^{n} \mathbf{T}^k \tag{5}$$

where the exponentiation of a Boolean matrix is calculated through the Boolean matrix product defined in (1). The transitive closure of a DAG describes the set of descendants of each node, where $d_{i,j} = 1$ if there exists a path from v_j to v_i, i.e., v_i is a descendant of v_j. Consequently, v_j is an ascendant of v_i. The transpose of the descendants matrix, $\mathbf{D}^{\mathrm{T}}$, therefore represents the ascendants matrix.

Computing the power of k of an adjacency matrix of a graph means calculating the nodes reachable through any k-step walk from every node v_i, which is a general result in graph theory

[1]We chose the column-row approach over the commonly used row-column approach to perform state and value propagation, described later in this section, by left-multiplying the transition matrix to a column state vector.

(Lemma 2.5 in [21]). Instead of computing the descendants matrix via (5), we can adopt a simpler formulation. By introducing a self-loop to every node, the power of k of the adjacency matrix calculates not only the reachable nodes of any k-step walk, but it also includes the reachable nodes through all shorter walks. Therefore,

$$\mathbf{D} = (\mathbf{T} \vee \mathbf{I})^n \wedge \neg\mathbf{I} \tag{6}$$

where $\mathbf{I} \in \mathbb{B}^{n\times n}$ is the identity matrix, and $\neg\mathbf{I}$ is the Boolean complement of $\mathbf{I}$. Given that $\mathbf{T}$ is an acyclic transition matrix, $\mathbf{T}^k$ has no element on the main diagonal $\forall k \in \mathbb{N}_{>0}$. Therefore, the elements introduced on the main diagonal are set back to zero.

B. State and Value Propagation

To use the DAG matrix $\mathbf{T}$ for the analysis of a DAG, two propagation methods are useful. The first is a Boolean state propagation and the second is a maximum value propagation. Let $\mathbf{x}_k \in \mathbb{B}^{n\times 1}$ denote a state describing which node of the DAG is visited at iteration k. Then, the state of the DAG in iteration $k+1$ can be calculated using the Boolean matrix multiplication as:

$$\mathbf{x}_{k+1} = \mathbf{T}\mathbf{x}_k \tag{7}$$

In this way $\mathbf{x}_{k+1}$ will contain 1 for the nodes that are reached with one step-walk from the ones in state $\mathbf{x}_k$, 0 for the others.

Similarly, a value can be propagated through the DAG. Let $\mathbf{v}_k \in \mathbb{R}^{n\times 1}$ denote a value for each node of the DAG at iteration k. This value can be propagated through the paths of the DAG by using

$$\mathbf{v}_{k+1} = \text{maxProduct}(\mathbf{T}, \mathbf{v}_k), \tag{8}$$

where the vector $\mathbf{v}_{k+1}$ describes the value $\mathbf{v}_k$ in the next iteration.

In this work, we are interested in propagating execution times along the DAG. Given that in a DAG more paths can converge to the same node, we will propagate the maximum value among converging paths. In the case of propagating execution times through the DAG, we can define a value function $\mathbf{v}$ as

$$\mathbf{v} = \text{maxProduct}(\mathbf{T}, \mathbf{v} + \mathbf{c}), \tag{9}$$

with $\mathbf{c}$ being the execution time of each node (WC or BC). In this equation, the value of a node is equal to the maximum of its predecessors' values plus its execution time. The fixed-point $\mathbf{v}^*$ solving Equation (9) can be found by iterating

$$\mathbf{v}_{k+1} = \text{maxProduct}(\mathbf{T}, \mathbf{v}_k + \mathbf{c}) \tag{10}$$

until it converges to $\mathbf{v}^*$ when $\mathbf{v}_{k+1} = \mathbf{v}_k$. Convergence is guaranteed to happen after at most n iterations, because the graph is acyclic and, therefore, all its paths are composed of n or fewer nodes.

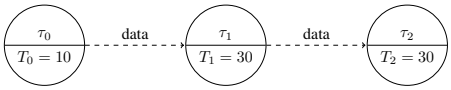
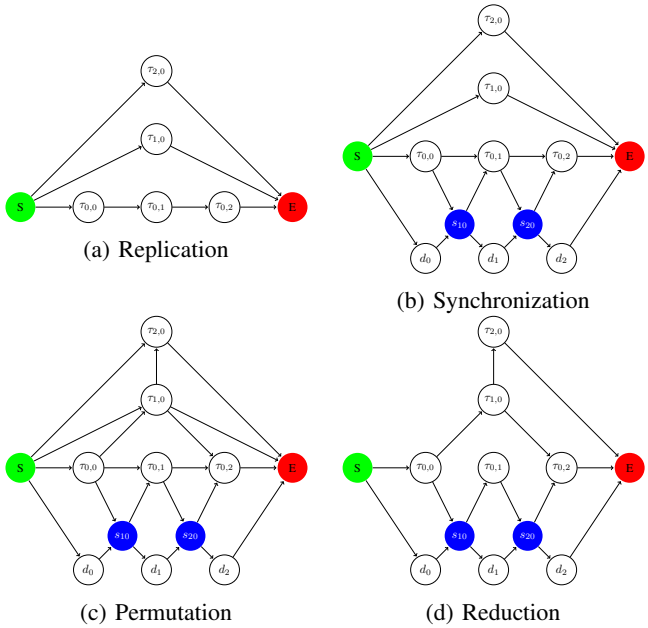


Fig. 2: The simple task set defined in Example 1.



(a) Replication (b) Synchronization

(c) Permutation (d) Reduction

Fig. 3: The 4-Stage DAG generation depicted.

V. DAG Generation

In this section, we explain how to convert a task set of periodic tasks with constraints to a set of potential *single-rate* DAGs. The explanation and mathematical derivations are augmented with an example to illustrate the conversion.

Example 1. *We consider an application modeled as a Multi-Rate task set* $\Gamma = \{\tau_0 = (7, 5, 10, 10), \tau_1 = (13, 10, 30, 30), \tau_2 = (10, 8, 30, 30)\}$*, with a constraint on the maximum data age of chain* $\{\tau_0, \tau_1, \tau_2\}$ *to be smaller than 50. The Multi-Rate task set is represented in Figure 2.*

A set of DAGs is generated using a 4-Stage DAG Generation. The set is subsequently pruned to accelerate the analysis in the next sections.

A. 4-Stage DAG Generation

We aim at generating a set of DAGs that have the potential to meet all the constraints. The DAG generation can be split into four stages:

1) The respective jobs of the tasks are created.
2) The jobs are synchronized to meet their respective deadlines.
3) The job-level precedence edges are added to address the *data edges*.
4) The DAGs are simplified by removing redundant edges.

The four steps for the example are depicted in Figure 3. We hereafter detail each step.

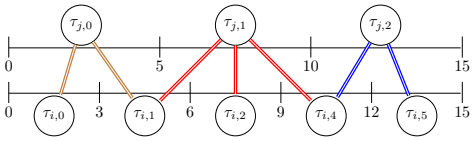
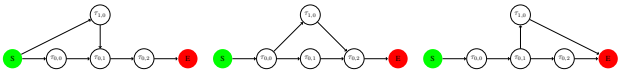


Fig. 4: Example of jobs of non-harmonic tasks limited to their super-period. Doubled arches indicate possible interaction between jobs.



(a) $\mathcal{A}_{0,1} = (0, 2, 1)$ (b) $\mathcal{A}_{0,1} = (1, 1, 1)$ (c) $\mathcal{A}_{0,1} = (2, 0, 1)$

Fig. 5: Arrangement permutations with one parallel job of τ_x; there are three permutations because $Q = 3$ and $\psi_0 = 1$, therefore $Q + 1 - \psi = 3$. The permutation (b) corresponds to the example in 3c.

1) Replication: Each task has to execute a number of jobs within one hyper-period. For a task τ_x, the number of jobs is $\frac{HP}{T_x}$. Since jobs are just instances of the same task, they should always run sequentially, therefore job-level precedence edges are added between successive jobs $\tau_{x,a}$ and $\tau_{x,a+1}$ where $a \in \{0, \ldots, \frac{HP}{T_x} - 1\}$. Additionally, the start node of the DAG is connected to each first job of each task, and each last job is connected to the end node. The resulting DAG for the example is shown in Figure 3a. To synchronize the jobs in the following step, each job gets an offset and deadline. For $\tau_{x,a}$, the offset is aT_x and the deadline is $aT_x + D_x$.

2) Synchronization: To be sure that tasks instances maintain their original period and deadlines in the DAG, a synchronisation mechanism has to be applied. In this way, we can enforce a job to start after its offset and to finish before its deadline. To accomplish this, we add additional nodes for synchronization purposes, as in Figure 3b. Firstly, we add a synchronization node σ_t, with $WC = BC = 0$, for each unique value t in the list of offsets and deadlines of all jobs. Secondly, we add dummy nodes δ between each two consecutive synchronization nodes σ_t and $\sigma_{t'}$, with $WC = BC = t' - t$, i.e., the difference in the timestamps of the corresponding synchronization nodes. The source and sink of the DAG are synchronization nodes too, with a timestamp of 0 and HP, respectively.

To enforce the jobs to execute in a time-window within its offset and deadline, an edge to the job is added from the synchronization node of the corresponding offset, and another one from the job to the synchronization node corresponding to its deadline.

3) Permutation: The various instances of tasks with different periods may be scheduled in multiple ways. We would like to enforce a suitable execution order between such instances, in order to minimize the latency of a given set of task chains. Thus, we convert the original multi-rate task set into several single-rate DAGs, each representing a possible activation pattern of the considered tasks. To do so, we include additional precedence edges to the DAG obtained at the previous step.

Consider two tasks τ_x and τ_y with periods T_x and T_y, assuming $T_y \geq T_x$ without loss of generality. Let $SP_{x,y}$ be the super-period of tasks τ_x and τ_y, defined as the least common multiple of their periods, i.e., $SP_{x,y} = lcm(T_x, T_y)$. Note there are $\frac{HP}{SP_{x,y}} - 1$ super-periods in the hyper-period HP of the whole task set.

There exist multiple ways to insert precedence edges between jobs of τ_x and τ_y in each super-period of length $SP_{x,y}$. Each possible edge assignment that complies with the multi-rate task specification is called "job arrangement".

To find all the possible permutations, two cases must be considered: harmonic and non-harmonic periods. In the former case, there exists $q \in \mathbb{N}$ for which $q = \frac{T_y}{T_x}$ and $SP_{x,y} = T_y$. Therefore, finding all the permutations between one job of τ_y and q of τ_x allows finding all the job arrangements in their super-period. The non-harmonic case is slightly more complicated. For two non-harmonic tasks, $q \in \mathbb{N}$ can be computed as $q = \lceil \frac{T_y}{T_x} \rceil$, but $SP_{x,y} \neq T_y$. In this case, one job of τ_y can be arranged with q or $q+1$ jobs of τ_x, because of the non-harmonicity. To better understand the problem, let us consider an example in which $T_x = 3$ and $T_y = 5$, as in Figure 4.

When periods are harmonic, a job of τ_x always interact with exactly one job of τ_y (and respectively, τ_y interacts with exactly q jobs of τ_y) . However, for non-harmonic periods, a job of τ_x can interact with 1 or 2 (at most) jobs of τ_y, as shown in Figure 4. For this reason, in the non-harmonic scenario some jobs of τ_y will interact with q (in the example $\lceil \frac{5}{3} \rceil = 2$, as for $\tau_{y,0}$ and $\tau_{y,2}$) jobs of τ_x, while others with $q + 1$ (in this case 3, as for $\tau_{y,1}$). In general, a job $\tau_{y,a}$ can interact with all the jobs between $\tau_{x,b}$ and $\tau_{x,c}$, where b and c can be obtained as:

$$b \in \mathbb{N} \mid o(\tau_{x,b}) \leq o(\tau_{y,a}) \wedge o(\tau_{x,b}) + T_x > o(\tau_{y,a}) \quad (11)$$

$$c \in \mathbb{N} \mid o(\tau_{x,c}) < o(\tau_{y,a}) + T_y \wedge o(\tau_{x,c}) + T_x \geq o(\tau_{y,a}) + T_y \quad (12)$$

where $o(\tau_{x,b})$ stands for the offset of the job $\tau_{x,b}$.

Once the interacting job of τ_x and τ_y have been associated, this case can be traced back to the harmonic one.

Now, let us consider a job $\tau_{y,s}$ and all the possible arrangements with Q jobs of τ_x (which is either q or $q+1$), denoted as $\mathcal{A}_{x,y}(s) = (pre_s, post_s, \psi_s)$ in the super-period $SP_{x,y}$. In this tuple, pre_s (resp. $post_s$) denotes the number of jobs of τ_x executing before (resp. after) each job of τ_y. ψ_s denotes the number of jobs of τ_x that can execute in parallel to the job of τ_y. This parameter is critical for the data update *variability*, that is defined as the difference between the maximum and minimum number of data updates. Since pre_s, $post_s$, and ψ_s comprise all the jobs of τ_x interacting with $\tau_{y,s}$, it follows that

$$pre_s + post_s + \psi_s = Q. \quad (13)$$

Three example arrangements for two tasks, τ_0 with $T_0 = 10$ and τ_1 with $T_1 = 30$, are shown in Figure 5. In all three arrangements, the job of τ_1 is parallel to one job of τ_0 ($\psi_0 =$

1). The number of permutations of arrangements with each $\tau_{y,s}$ can be calculated as:

$$perm(\mathcal{A}_{x,y}(s)) = \sum_{\psi=\{0...Q\}} (Q+1-\psi), \tag{14}$$

while the permutations can be found combining all the possible edges between the jobs of the two tasks.

For the harmonic case, this value is also the total number of permutations of a super-period:

$$perm_{SP_{x,y}} = perm(\mathcal{A}_{x,y}(s)). \tag{15}$$

On the other hand, for the non-harmonic case, the number of permutations for the super-period is obtained as:

$$perm_{SP_{x,y}} = \prod_{\forall \tau_{y,s} \in \{0...\frac{SP_{x,y}}{T_y}\}} perm(\mathcal{A}_{x,y}(s)). \tag{16}$$

Finally, considering all the super-periods contained in a hyper-period, the total number of permutations can be given by:

$$perm_{total} = \prod_{\forall x,y} perm_{SP_{x,y}}^{\frac{HP}{SP_{x,y}}}, \tag{17}$$

where $x \neq y$ and τ_x and τ_y are consecutive tasks in a given task chain. Each combination of arrangement permutations generates a new DAG that can be analyzed. Therefore, it is critical to keep the number of possible permutations as small as possible. A reduction of the exploration space is discussed in Section V-B.

Figure 3c shows one of the obtained DAG, whose simplified[2] adjacency matrix $\mathbf{T}$ and transitive closure matrix $\mathbf{D}$ (obtained with (6)) are the following:

$$\mathbf{T} = \begin{array}{c} \\ S \\ \tau_{0,0} \\ \tau_{0,1} \\ \tau_{0,2} \\ \tau_{1,0} \\ \tau_{2,0} \\ E \end{array} \begin{bmatrix} & & & & & & \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{D} = \begin{array}{c} \\ S \\ \tau_{0,0} \\ \tau_{0,1} \\ \tau_{0,2} \\ \tau_{1,0} \\ \tau_{2,0} \\ E \end{array} \begin{bmatrix} & & & & & & \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

4) Reduction: While constructing the DAGs, it is possible to end up generating redundant edges. There is a redundant edge between two nodes when there exist both a direct edge and a non-direct path. Redundant edges can be removed using a technique called transitive reduction, firstly proposed by Aho et al. in [23]. The transitive reduction of a DAG uniquely describes the sub-graph of this DAG with the fewest possible edges, while maintaining the same reachability relation.

The transitive reduction of a DAG can be calculated in different ways. Since in this work we need the transitive reduction as well as the transitive closure of the DAG, we compute the transitive reduction using

$$\mathbf{T}_r = \mathbf{T} \wedge \neg(\mathbf{T} \cdot \mathbf{D}), \tag{18}$$

where $(\mathbf{T} \cdot \mathbf{D})$ has 1 in (j, i) if the node j can reach the node i in more than one step, 0 otherwise. Applying equation (18) means removing direct edges $e(v_j, v_i)$ in $\mathbf{T}$ that are redundant because a non-direct path already exists between node j and node i.

In Figure 3d, the obtained DAG with reduced edges is presented. For that example, the matrix $\mathbf{T} \cdot \mathbf{D}$ and $\mathbf{T_r}$ (obtained with (18)) are the following[3]:

$$\mathbf{T} \cdot \mathbf{D} = \begin{array}{c} \\ S \\ \tau_{0,0} \\ \tau_{0,1} \\ \tau_{0,2} \\ \tau_{1,0} \\ \tau_{2,0} \\ E \end{array} \begin{bmatrix} & & & & & & \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{T_r} = \begin{array}{c} \\ S \\ \tau_{0,0} \\ \tau_{0,1} \\ \tau_{0,2} \\ \tau_{1,0} \\ \tau_{2,0} \\ E \end{array} \begin{bmatrix} & & & & & & \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The interaction of the different *data edges* during the Permutation stage can result in DAGs that are inherently not schedulable. These DAGs can be removed to speed up the analysis. Then, two factors are further inspected: potential cycles in the generated DAG, and length of the longest chain.

DAGs containing cycles need to be removed, as they are inconsistent with the task semantics and they could not be feasibly scheduled. Finding cycles in a graph is a common problem which can be solved with several approaches. In this work, we use state propagation, described in Section IV-B. We adopt a state vector $\mathbf{x}$, whose elements indicate whether a path exists (1) or not (0). Initially, $\mathbf{x}_0 = \mathbf{1}$ to consider the potential paths from all the nodes. Then, we apply state propagation in Equation (7), multiplying the state vector with the adjacency matrix $\mathbf{T}$. This means stepping from a node to its successor: if it has any, the resulting vector will have a 1 in the corresponding position, otherwise it will have a 0. Repeating this operation means going through all the possible paths. Since the graph is acyclic and it has n nodes, there should be no path with a length greater than n. In other words, the resulting vector should have all 0's after at most n steps, indicating that all the paths have ended, i.e., there are no more nodes to step into. If this is not the case, it means the DAG contains cycles, and it can be discarded.

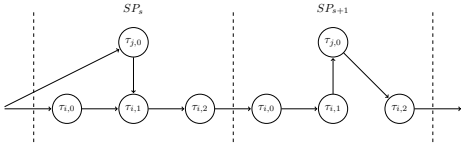
Finally, the longest chain in the DAG corresponds to the chain with the longest execution time. This chain can be explicitly found by calculating the fixed-point of (9) with $\mathbf{c} = \mathbf{WC}$, the WCET of each node. If any value in $\mathbf{v}^* + \mathbf{WC}$ is bigger than the hyper-period HP, it means that there exists a path whose sum of WCETs exceeds the hyper-period, which makes the DAG not schedulable. Also these DAGs are discarded.

In the example, the vectors of $\mathbf{WC}$ and $\mathbf{v}^*$ are the following:

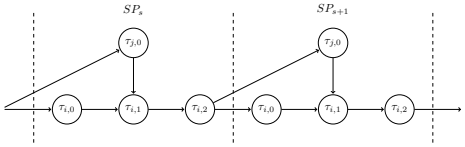
$$\mathbf{WC} = \begin{bmatrix} 0 \\ 7 \\ 7 \\ 7 \\ 10 \\ 13 \\ 0 \end{bmatrix} \quad \mathbf{v}^* = \begin{bmatrix} 30 \\ 23 \\ 7 \\ 0 \\ 10 \\ 0 \\ 0 \end{bmatrix}$$

[2]Without synchronization and dummy nodes, removed for a clearer representation, but used in the actual algorithm.

[3]Given the previously mentioned simplification, consecutive jobs of the same task have precedence constraints between them, rather than having edges to and from synchronization nodes. In the example, there is an edge from $\tau_{0,0}$ to $\tau_{0,1}$ and one from $\tau_{0,1}$ to $\tau_{0,2}$.



(a) Heterogeneous arrangement example with $\mathcal{A}_{x,y}(s) = (0, 2, 1)$ and $\mathcal{A}_{x,y}(s+1) = (2, 1, 0)$



(b) Homogeneous arrangement example with $\mathcal{A}_{x,y}(s) = \mathcal{A}_{x,y}(s+1) = (0, 2, 1)$

Fig. 6: Examples showing heterogeneous and homogeneous arrangements.

B. Permutation Space Reduction

The worst-case number of permutations, and thus the total number of DAGs created, is given in (17), i.e., it is scaling exponentially with the size of the task set. Therefore, a reduction of the permutation space is essential to keep the approach computationally tractable for larger task sets. To reduce the permutation space, we inspect the inter-super-period-arrangement.

Given the previously adopted tasks τ_0 and τ_1, Figure 6 shows two possible arrangements, omitting synchronization nodes for simplicity.

Let us consider a couple of harmonic tasks τ_x, a τ_y in their super-period $SP_{x,y}$. For a given parallelism ψ_s, relative to job $\tau_{y,s}$, the execution order of the parallel jobs is not defined. Therefore, a bounded number of jobs of τ_x, denoted as $prePar_s \in \{0, \dots, \psi_s\}$, can execute before $\tau_{y,s}$. Consequently $\psi_s - prePar_s$ jobs of τ_x will execute after $\tau_{y,s}$. The probability distribution of $prePar_s$ is not relevant since the only values that affect the latency variability are, by definition, the extremes, i.e.,

$$\max(prePar_s) = \psi_s \text{ and } \min(prePar_s) = 0 \tag{19}$$

Based on these definitions, the number of jobs of τ_x between two consecutive jobs $\tau_{y,s}$ and $\tau_{y,s+1}$ (in the following super-period) is given by

$$n_{s,s+1} = post_s + (\psi_s - prePar_s) + pre_{s+1} + prePar_{s+1} \tag{20}$$

The upper and lower bound of this value are given by

$$\max(n_{s,s+1}) = post_s + \psi_s + pre_{s+1} + \psi_{s+1} \tag{21}$$

$$\min(n_{s,s+1}) = post_s + pre_{s+1} \tag{22}$$

The *variability* of the data updates, i.e., the difference between the maximum and the minimum data updates in between, can be formalized as:

$$Var_{x,y} = \max_s\{\max(n_{s,s+1})\} - \min_t\{\min(n_{t,t+1})\}, \tag{23}$$

using t in the minimum to highlight that the maximum and minimum do not need to consider the same job of τ_y, and thus the same arrangement. However, in a homogeneous arrangement, $\mathcal{A}_{x,y}(s) = \mathcal{A}_{x,y}(s+1) = (pre, post, \psi)$ and $s = t$. Therefore,

$$Var_{x,y,hom} = 2\psi_s \tag{24}$$

Comparing to heterogeneous arrangements, in which $\mathcal{A}_{x,y}(s) \neq \mathcal{A}_{x,y}(s+1), \forall s$, two observations can be made. On the one hand, a higher value of ψ_s for a job $\tau_{y,s}$ increases schedulability of the related super-period, since it allows for more parallelism and shortens the longest path. Given that the schedulability of all the super-periods determines the schedulability of the hyper-period, the value of ψ_s is crucial. On the other hand, from an application side, the data update *variability* should be as low as possible to constrain end-to-end latency.

To reduce the permutation space while investigating all the permutations that optimize latency, we chose to sacrifice optimality w.r.t. schedulability. Homogeneous arrangements are better at this compromise. To show it, we prove that

$$Var_{x,y,het} > Var_{x,y,hom} \tag{25}$$

Proposition: Given two tasks τ_x and τ_y with periods $gT_x = T_y, \quad hT_y = HP, \quad g, h \in \mathbb{N}_+$, a heterogeneous arrangement results in a strictly higher *variability* than a homogeneous arrangement.

Proof: In a heterogeneous arrangement, $\mathcal{A}_{x,y}(s) \neq \mathcal{A}_{x,y}(s+1)$, which means that $(pre_s, post_s, \psi_s) \neq (pre_{s+1}, post_{s+1}, \psi_{s+1})$. Let us define $\alpha_s \in \mathbb{Z}$ (resp. $\beta_s \in \mathbb{Z}$) as the difference between the jobs of τ_x that execute before (resp. after) $\tau_{y,s+1}$ and the jobs of τ_x that execute before (resp. after) $\tau_{y,s}$[4]

$$\alpha_s = pre_{s+1} - pre_s \tag{26}$$

$$\beta_s = post_{s+1} - post_s \tag{27}$$

Consequently, considering that $pre_s + post_s + \psi_s = pre_{s+1} + post_{s+1} + \psi_{s+1} = Q$, we derive

$$\psi_{s+1} = \psi_s - \alpha_s - \beta_s \tag{28}$$

With this definition, Equation (20) provides

$$\begin{aligned} n_{s,s+1} &= post_s + (\psi_s - prePar_s) + pre_{s+1} + prePar_{s+1} \\ n_{s,s+1} &= post_s + (\psi_s - prePar_s) + pre_s + \alpha_s + prePar_{s+1} \\ &= Q + \alpha_s - prePar_s + prePar_{s+1}. \end{aligned}$$

Then,

$$\begin{aligned} \max(n_{s,s+1}) &= Q + \alpha_s + \psi_{s+1} \\ &= Q + \psi_s - \beta_s \\ \min(n_{s,s+1}) &= Q + \alpha_s - \psi_s \end{aligned}$$

[4]Remember that there is only one job of $\tau_{y,s}$ in each super-period $SP_{x,y}$. Therefore, $\tau_{y,s+1}$ refers to the next super-period.

The *variability* in Equation (23) can then be simplified to

$$\begin{aligned} Var_{x,y,het} &= \max_s\{\max(n_{s,s+1})\} - \min_t\{\min(n_{t,t+1})\} \\ &= \max_s\{\psi_s - \beta_s\} - \min_t\{\alpha_t - \psi_t\} \\ &= \max_s\{\psi_s - \beta_s\} + \max_t\{\psi_t - \alpha_t\} \\ &= 2\psi_s + \max_s\{-\beta_s\} + \max_t\{-\alpha_t\}. \end{aligned}$$

As the full arrangement is the same in each hyper-period, the super-period arrangement is cyclic. Since α_s and β_s denote the change of the arrangement, the cyclicity of $\mathcal{A}$ requires

$$\sum_{s\in\{0,\dots,\frac{HP}{T_y}\}} \alpha_s = \sum_{s\in\{0,\dots,\frac{HP}{T_y}\}} \beta_s = 0. \tag{29}$$

Therefore, $\exists\alpha_s < 0$ and $\exists\beta_s < 0$ such that

$$Var_{x,y,het} > 2\psi_s \tag{30}$$

Since $Var_{x,y,het} > 2\psi_s$, it then follows $Var_{x,y,het} > Var_{x,y,hom}$, proving the proposition. □

We can therefore omit heterogeneous arrangements without affecting the resulting end-to-end latency, since no such arrangement can provide a better compromise with respect to *variability*. By discarding the heterogeneous arrangements in the permutations, the value of $perm_{total}$ in (17) can be reduced to

$$perm_{total} = \prod_{\forall x,y} perm_{SP_{x,y}}, \tag{31}$$

where $x \neq y$ and τ_x and τ_y are consecutive tasks in a given task chain, and the full hyper-period arrangement is defined by a unique super-period arrangement. This is valid both for harmonic and non-harmonic tasks.

C. Computational Complexity

The computational cost of the overall method can be summarized as $\mathcal{O}(perm_\psi \times perm_\mathcal{A} \times n^4)$. The first term $perm_\psi$ represents all the permutations for all the possible ψ values. From (31), it can be expressed as

$$perm_\psi = \prod_{(e_x,e_y)\in E} \frac{\max(T_x,T_y)}{\min(T_x,T_y)}. \tag{32}$$

The second term $perm_\mathcal{A}$ represents all the arrangement permutations for a fixed ψ. From (14), it can be expressed as

$$perm_\mathcal{A} = \prod_{(e_x,e_y)\in E} \Big(\frac{\max(T_x,T_y)}{\min(T_x,T_y)} - \psi + 1\Big). \tag{33}$$

Lastly, $\mathcal{O}(n^4)$ is the maximum cost of all the math operations applied on the obtained DAGs, which are matrix-vector multiplication $\mathcal{O}(n^2)$, matrix multiplication $\mathcal{O}(n^3)$ and matrix exponentiation $\mathcal{O}(n^4)$. Let us define R as the maximum ratio between periods of the taskset, i.e., $R = \frac{max(T_x)}{min(T_y)} \forall x,y \in \{0\dots N-1\}$. The computational cost of the method can then be expressed as:

$$\mathcal{O}(R^{|E|}R^{|E|}(RN)^4) = \mathcal{O}(R^{2|E|}(RN)^4). \tag{34}$$

job	EST	LST	EFT	LFT
$\tau_{0,0}$	0	0	5	7
$\tau_{0,1}$	10	13	15	20
$\tau_{0,2}$	20	23	25	30
$\tau_{1,0}$	5	7	15	20
$\tau_{2,0}$	15	20	23	30

TABLE I: Timing attribute for the for Example 1.

The complexity is thus exponential in the number of edges $|E|$. Such a high cost is mainly determined by the need to take into account all the permutations at once. However, this is also the reason why the proposed conversion method allows better controlling end-to-end latencies, jointly optimizing data and reaction times of all the task chains given in input. This is achieved by picking up the best configuration out of all the permutations generated by means of a cost function.

VI. End-to-end Latency and Schedulability

In this section, a method to calculate an upper bound on *data age* and *reaction time* is proposed. As explained in the introduction, data age defines the maximum time a data produced by the first task of the chain can influence the last one. Reaction time is the maximum interval between the acquisition of a stimulus in the first task of a chain and the moment the first instance of the last task in the chain reacts to it.

We first define a set of additional timing attributes, that will be used to compute the end-to-end latency. The schedulability of the DAG is verified by deriving a static schedule. If more than one generated DAG meets the latency and schedulability constraints, we select the DAG that maximizes a weighted sum of the end-to-end latencies, taking into account all the tasks chains in input.

For each job, we define the following timing attributes: Earliest Finishing Time (EFT), Latest Finishing Time (LFT), Earliest Starting Time (EST) and Latest Starting Time (LST). The earliest a node can start is the maximum of all its predecessors' earliest finishing times. Similarly, the latest a node can finish is the minimum of its successors' latest starting times. These values can be iteratively calculated using the operators defined in Section IV, initializing $EST_j = 0$ and $EFT_j = HP$ for all nodes j:

$$\begin{aligned} EST_i &= \max_{\forall j}\{(EST_j + BC_j)\mathbf{T}_{j,i}\} \\ LFT_i &= \min_{\forall j}\{(LFT_j - WC_j)\mathbf{T}_{i,j}\} \\ EFT_i &= EST_i + BC_i \\ LST_i &= LFT_i - WC_i. \end{aligned}$$

Table I reports the timing attributes computed for Example 1.

A. Task Chain Propagation

In a DAG $\mathcal{G}$, a node j is defined to *react* to node i if there exists a direct or indirect edge from node i to node j. A node k *reacting* to node j also *reacts* to node i. Further, a node k *reacts* to the chain (i,j) if node j *reacts* to node i and node k reacts to node j.

Extending this definition to tasks and jobs:

- $\tau_{y,b}$ *reacts* to $\tau_{y,a}, \forall b > a$;
- Consequently, if $\tau_{y,a}$ *reacts* to $\tau_{x,c}$, it follows that $\tau_{y,b}$ reacts to $\tau_{x,c}$.

Given a task chain $(\tau_x, \ldots, \tau_z)$, the *reactions* of jobs of task τ_z to each job of τ_x can be found. Consider a job $\tau_{x,a}$ of the first task in the chain. The *first* (resp. *last*) *reaction* to $\tau_{x,a}$ is defined as the first (resp. last) job of the last task τ_z that reacts to $\tau_{x,a}$. The *reaction time* (resp. *data age*) is then defined as the maximum interval between a stimulus in a job $\tau_{x,a}$ and the finishing time of the first (resp. last) reaction, taken over all instances $\tau_{x,a}$, for all $a \in [0, \frac{HP}{T_x}]$. Since the structure of the DAG repeats after each hyper-period, it is sufficient to consider only the first hyper-period.

Algorithm 1: findReactions

Input: $\mathcal{C} = \{\tau_{start}, ..., \tau_{end}\}$
Output: $1^{st}reactions, lastreactions$

```
1  forall a ∈ {0, ..., HP/T_start + 1} do
2      fr_job = τ_{start,a};
3      lr_job = null;
4      forall τ_x ∈ C \ τ_start do
5          b = 0;
6          while τ_{x,b} does not react to fr_job do
7              b++;
8          fr_job = τ_{x,b};
9          if b > 0 then
10             lr_job = τ_{x,b-1};
11     if a ≤ HP/T_start then
12         1st reactions.insert(τ_{start,a}, fr_job);
13     if (b ≠ null) and (a > 0) and
         (1st reactions(τ_{start,a-1}) ≠ fr_job) then
14         lastreactions.insert(τ_{start,a-1}, lr_job);
15 return 1st reactions, lastreactions;
```

A method to compute the *first* and *last reactions* is shown in Algorithm 1. The algorithm considers every job of the starting task of the chain in one hyper-period, plus an additional job (to cover the last reactions). The first reacting job (fr_job) is set to $\tau_{start,a}$. Then, for each task in the chain, we find the first job $\tau_{x,b}$ that reacts to fr_job, and we use it to update fr_job. This can happen either in the same hyper-period of fr_job, or in the next one. The preceding job $\tau_{x,b-1}$ is instead used to update lr_job, which keeps track of the last reaction to $\tau_{start,a-1}$. Once the whole chain has been considered, $1^{st}reactions$ and $lastreactions$ are updated. The latter is updated only if b is not null and if the first reaction to $\tau_{start,a}$ is different from the first reaction to $\tau_{start,a-1}$. *Reaction time* and *data age* can then be simply derived as

$$RT = \max_{\tau_{x,a} \in 1^{st}reactions} \{LFT_{1^{st}reactions} - EST_{\tau_{x,a}}\} \quad (35)$$

$$DA = \max_{\tau_{x,a} \in lastreactions} \{LFT_{lastreaction} - EST_{\tau_{x,a}}\}, \quad (36)$$

i.e., reaction time (resp. data age) is the difference between the first (resp. last) moment some data is used by a job of the last task in the chain (LFT) and the first moment the same data is read from the job of the first task in the chain (EST). Since the schedule repeats identically after each hyper-period, it is sufficient to consider all the jobs of the first task in the first hyper-period.

We hereafter prove that Algorithm 1 correctly finds the first and last reactions. The algorithm considers all the jobs of the starting task in the chain (line 1). For each of the starting jobs, it iterates over all the other tasks in the chain, always looking for the first and last reacting job (lines 4-14). Let us consider two consecutive tasks in the chain τ_x and τ_y and only one job a of τ_x.

- To find the maximum reaction time, the jobs of τ_y that are said to react to $\tau_{x,a}$ are those that are definitely executing after $\tau_{x,a}$, i.e., they belong to $\tau_{x,a}$'s descendants, or their EST is greater than the LFT of $\tau_{x,a}$. Since the DAG is schedulable, a reacting job can always be found (and the loop at line 6 is not infinite) either in the same hyper-period of $\tau_{x,a}$, or in the next one. Once a job $\tau_{x,b}$ is found to react to $\tau_{x,a}$, it becomes the starting job to find the first reaction between τ_y and the next task in the chain.
- The maximum data age of $\tau_{x,a}$ is strictly related to the first reaction to $\tau_{x,a+1}$. Indeed, the first reaction to $\tau_{x,a+1}$ assures that the data from $\tau_{x,a}$ are no longer used: the last time they were used was by the job preceding the one that surely reacts to $\tau_{x,a+1}$. Thus, when finding the first reaction $\tau_{x,a+1}$, the last reaction of $\tau_{x,a}$ can be found (line 14).

In the example DAG in Figure 3d, *data age* is 30, while *reaction time* is 50. The chains leading to these values are $\{\tau_{0,0}, \tau_{1,0}, \tau_{2,0}\}$ for *data age* and $\{\tau_{0,1}, \tau'_{1,0}, \tau'_{2,0}\}$ for *reaction time*, where a prime indicates that the job is in the next hyper-period.

B. Schedulability

To build a feasible schedule for a given number of cores, we apply a list-scheduling heuristic for non-preemptive DAG, very similar to the Heterogeneous Earliest Finishing Time (HEFT) algorithm presented in [24]. We decided to use a (node-level) limited preemptive scheduling for (i) avoiding job-level migrations, (ii) reducing cache-related preemption delays, and (iii) minimizing the input-output delay and jitter [25].

The list-scheduling algorithm is summarized in Algorithm 2. Jobs are sorted in increasing LFT order (line 3). Given p homogeneous processors, a job is scheduled at time t only if it is ready and a processor is available. A job enters the ready queue (line 8) at time t only if (i) its EST is greater than or equal than t, (ii) all its predecessors in the DAG have been executed, and (iii) its LFT is the smallest between all the remaining jobs' LFT. The ready queue is sorted in increasing LFT order (line 9). A ready job is scheduled if a processor is available and if its execution time, starting from the current t, does not exceed its LFT (lines 13,16,17). If

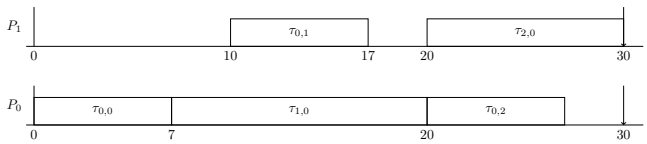


Fig. 7: Schedule produced for the DAG in Figure 3d with 2 cores

this last condition is not met, the algorithm declares the DAG not schedulable (lines 13,14). An example of the schedule obtained for Example 1 is shown in Figure 7.

Algorithm 2: isDAGSchedulable

Input: $V, pred, p, EST, LFT, WC$
Output: *true* if the DAG is schedulable on p processors, *false* otherwise
Data: Ready queue of jobs $rq = \{\}$, procExec vector

1 $pq_i = \{\}, \forall i = 1, \ldots, p$;
2 $nodes = \{v_0, \ldots, v_{n-1}\}, n = |V|$;
3 sort($nodes$) `sort by ascending value of LFT`
4 **for** $t = 0, 1, \ldots, HP$ **do**
5 **forall** $node \in nodes$ **do**
6 **if** $EST[node] > t$ ***and*** *all* $pred[node]$ *have finished* ***and*** $LFT[node] \leq$ *all other nodes LFTs* **then**
7 $nodes = nodes \setminus node$;
8 $rq.push(node)$;
9 sort(rq) `sort by ascending value of LFT`
10 **for** $i = 1, ..., p$ **do**
11 **if** $rq \neq \{\}$ ***and*** $procExec_i == 0$ **then**
12 $readyJob = rq.pop()$;
13 **if** $t + WC[readyJob] > LFT[readyJob]$ **then**
14 **return** *false*;
15 **else**
16 $procExec_i = WC[readyJob]$;
17 $pq_i.push(readyjob)$;
18 **if** $procExec_i > 0$ **then**
19 $procExec_i = procExec_i - 1$;
20 **return** *true*;

VII. Evaluation

To evaluate our approach, we first use simulation to validate end-to-end latency bounds as well as schedulability and then compare the proposed method with the state-of-the-art using a realistic automotive benchmark.

A. Evaluation via Simulation

To validate that the DAGs generated with the method presented in this paper comply with the constraints, we developed a simulation tool. The tool uses the DAG to schedule the individual tasks, which tracks the data propagation through the task chains under analysis. The execution time of each task is identically and independently sampled from the BC to WC interval. The schedule is generated according to EDF, and the deadline is set equal to LFT.

Taskset Γ	
$\tau_i = (WC_i, BC_i, P_i, D_i)$	Task
$\tau_0 = (7, 5, 50, 50)$	GPS
$\tau_1 = (12, 10, 50, 50)$	Lidar
$\tau_2 = (28, 22, 50, 50)$	Localization
$\tau_3 = (28, 25, 50, 50)$	Detection
$\tau_4 = (25, 18.9, 50, 50)$	Fusion
$\tau_5 = (2, 1.8, 25, 25)$	Camera
$\tau_6 = (6.5, 3, 10, 10)$	EKF
$\tau_7 = (5, 3.2, 10, 10)$	Planner
$\tau_8 = (4.5, 1.8, 10, 10)$	Control
Task chains	
chain $\{\tau_{start}, \ldots, \tau_{end}\}$	(Age, Reaction)
$\{\tau_5, \tau_3, \tau_4\}$	$(120, 120)$
$\{\tau_0, \tau_2, \tau_6, \tau_7, \tau_8\}$	$(120, 150)$
$\{\tau_1, \tau_2, \tau_6, \tau_7, \tau_8\}$	$(120, 150)$
$\{\tau_5, \tau_3, \tau_4, \tau_7, \tau_8\}$	$(150, 150)$
Scheduling constraints	
6 processors	

TABLE II: Periodic taskset and constraints used for the simulation, referring to the application of Figure 1.

We simulated the best DAG, in terms of schedulability and end-to-end latency, produced for the application introduced in Figure 1. The task set specification and constraint are detailed in Table II. The latency computed for the given chains is reported in Table III.

Using the simulation tool, the DAG is simulated for 10^9 ms, which leads to the following results. Two distributions of *reaction time* and *data age* of two task chains are shown in Figure 8 and 9. In Figure 8 the *reaction time* plot is showing two distributions, one for each camera frame. In the DAG, the camera jobs are serialized to the detection job, leading to only one distribution for the *data age*, because the detection job always receives the freshest camera frame. A similar distribution for the *reaction time* can be seen for the task chain in Figure 9, as the task chain, is extended with the planner and control task. The *data age*, however, shows several distributions. This is due to the higher rate of the planner and control task with respect to the fusion task. Nevertheless, the *data age* of the data corresponding to each control output is always based on the freshest camera frame, which can be seen by comparing the distribution shapes. The simulation showed that all the calculated upper bounds for *data age* and *reaction time* for the four task chains are not exceeded.

B. Evaluation via Benchmark

To further analyze the performance of the proposed method the detailed automotive benchmark proposed by BOSCH for the WATERS challenge in 2015 [26] has been adopted. Multi-rate periodic task sets and cause-effect chains are randomly generated while conforming with the char-

chain $\{\tau_x, \ldots, \tau_y\}$	(Age, Reaction)
$\{\tau_5, \tau_3, \tau_4\}$	$(75, 98.2)$
$\{\tau_0, \tau_2, \tau_6, \tau_7, \tau_8\}$	$(105, 65)$
$\{\tau_1, \tau_2, \tau_6, \tau_7, \tau_8\}$	$(105, 65)$
$\{\tau_5, \tau_3, \tau_4, \tau_7, \tau_8\}$	$(125, 108.2)$

TABLE III: Maximum *data age* and *reaction time* for task chains of the best DAG produced for the task set described by Table II

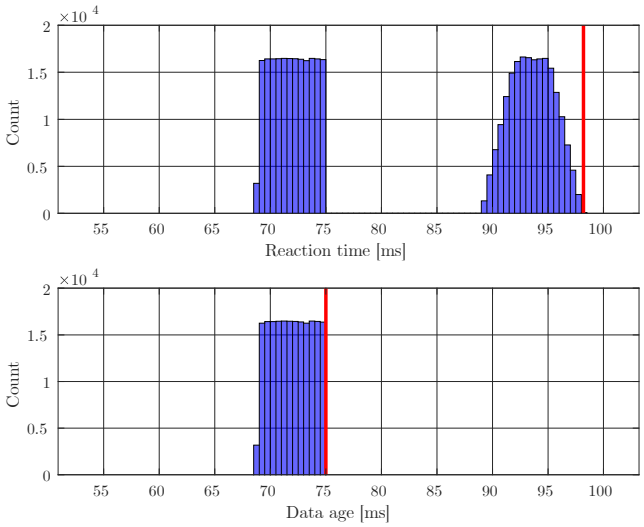


Fig. 8: *Reaction time* and *data age* of the chain {Camera, Detection, Fusion}, or $\{\tau_5, \tau_3, \tau_4\}$, evaluated in simulation with the red lines showing the calculated maximum.

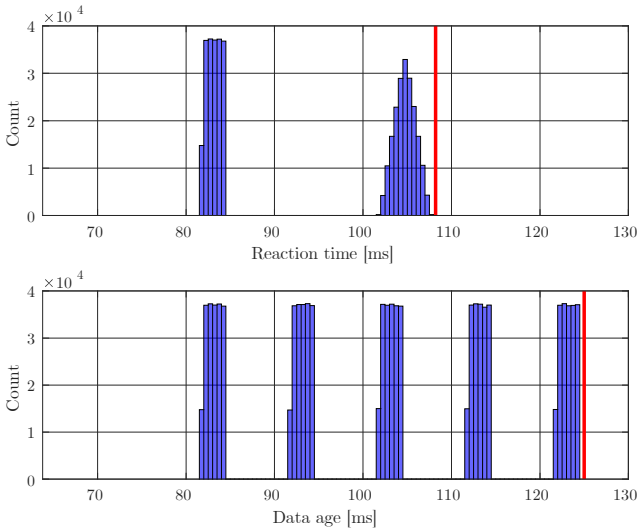


Fig. 9: *Reaction time* and *data age* of the chain {Camera, Detection, Fusion, Planner, Control}, or $\{\tau_5, \tau_3, \tau_4, \tau_7, \tau_8\}$, evaluated in simulation with the red lines showing the calculated maximum.

	permutation	admissible (%)	schedulable (%)
min	0.00	0.00	0.00
avg	1.830.48	62.10	61.60
max	18.148.00	100.00	100.00

TABLE IV: Statistics about DAG permutations, admissible and schedulable DAGs on 1000 different task set.

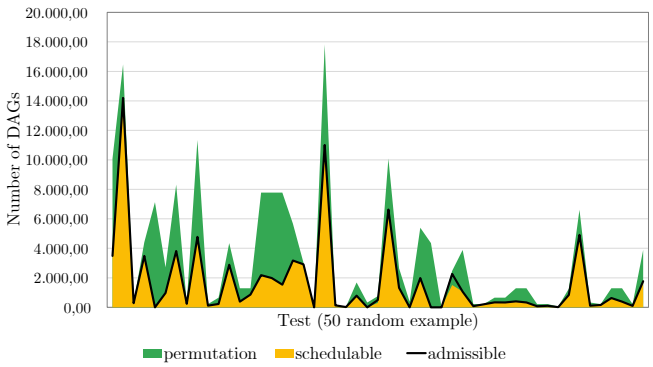


Fig. 10: Statistic about produced DAG on 50 randomly chosen task set of the 1000 analyzed.

acterization. Task periods are selected with given distribution, out of the periods found in automotive applications $[1, 5, 10, 20, 50, 100, 200, 1000]$ms. Cause-effect chains are generated to include tasks of either $1, 2$, or 3 different period wherein tasks of the same period can appear 2 to 5 times. To obtain a higher utilization, the individual task execution times are generated based on UUniFast [27]. For the experiments 1000 task set composed of 5 tasks and 15 chains have been taken into account, with a utilization equal to 1.5, considering 2 cores available.

Table IV reports some statistics about the DAG obtained from the 1000 multi-rate periodic task sets. From the initial generated permutations the 40% is on average removed due to cycles or a non-schedulable longest chain. However, between the admissible generated DAGs[5], almost the totality is also schedulable on 2 cores. Figure 10 shows 50 randomly selected examples in which the number of permutations, admissible DAGs and schedulable DAGs are compared.

C. Comparison with state-of-the-art

1) Qualitative: In [19], Saidi et al. present a method to convert a parallel multi-rate task set with precedence and data edges into a single-rate DAG. However, the end-to-end latency is not considered, and only one possible DAG is generated. Therefore, no guarantee is given on *reaction time* or *data age*. Moreover, there are no synchronization methods to force task instances to execute within their periods, potentially leading to a wrong implementation of the system.

In [18], Foget et al. present another conversion method, producing different DAGs. However, neither this work takes into account latency. Different solutions are created just for

[5]DAGs that have a correct structure (i.e., no cycles and no path greater than the hyper-period) but that may still be unschedulable.

	Forget [18]	Saidi [19]	Becker [9]	this paper
schedulable task set(%)	46.9	21.8	90.5[6]	**90.5**
1st lowest data age (%)	45.89	17.85	77.81	**96.82**
2nd lowest data age (%)	2.79	0.09	13.55	3.15
3rd lowest data age (%)	3.03	1.46	6.38	0.03
4th lowest data age (%)	0.00	4.58	2.25	0.00

TABLE V: Schedulability and *data age* results on 1000 task set compliant to [26] of 5 tasks and 15 chains, with utilization equals to 1.5.

	Forget [18]	Saidi [19]	Becker [9]	this paper
min [ms]	0.002	0.002	0.078	0.002
avg [ms]	0.571	0.022	3.001	21.410
max [ms]	4.422	0.116	16.614	433.033

TABLE VI: Execution times in milliseconds on an Intel(R) Core(TM) i7-7700HQ CPU @2.80GHz.

schedulability reasons, picking the version that makes the DAG schedulable with EDF.

Focusing on data latency, the most related approach is the one introduced by Becker et al. in [9], where the focus is on the computation of *data age*. In their work, they can compute *data age* for a given chain of periodic tasks, given a communication model (i.e. implicit, explicit or LET). The method allows generating job-level dependencies to meet latency requirements. However, *data age* is the only parameter under their analysis. Moreover, they can optimize end-to-end latency for only a single chain. Once job-level dependencies are inserted, all the other chains are affected. Finally, their work assumes the input task set is already schedulable. Our work has several improvements over their approach, i.e., (i) the model is more general and can jointly optimize the latency of multiple chains, (ii) we consider not only *data age*, but also *reaction time*, and (iii) our task allocation and scheduling algorithms also consider the schedulability of the system.

2) Quantitative: To show that our method dominates the state-of-the-art, we implemented the solutions proposed in [19], [18], [9], and tested them on the previously presented automotive benchmark by BOSCH. Table V shows the results for the 1000 task sets considered, and all the 15000 task chains, while Table VI offers a comparison of the running times of the considered methods for larger task sets, i.e., composed of 10 tasks with 15 task chains.

The proposed method not only dominates the others in term of schedulability, but also in terms of *data age*. Given that [18] and [19] do not propose a method to compute end-to-end latency, we adopted our algorithm for this scope. Considering *data age*, our method produces a DAG that leads to the lowest end-to-end latency bound in 96.82% of cases. There are some cases in which Becker [9] method obtains a tighter latency, since it optimizes a single chain. However, the limitation of that approach is that it is not able to optimize all the given chains for a task set, while our method optimizes them all. Therefore, we are willing to sacrifice the latency of some chains for a more balanced improvement of all chains.

[6]Since no method is proposed in [9] to check schedulability, we applied our method to derive the schedulable task sets.

On the other hand, when optimizing a single chain, our method allows finding a better solution than with the method presented in [9]. As an example, consider the task chain in Example 1. Using the method by Becker et al., a minimum *data age* of 40 can be achieved, inserting a precedence constraint between $\tau_{1,0}$ and $\tau_{2,0}$. Instead, our method allows achieving a *data age* of 30, picking a DAG with additional precedence constraints.

As can be expected, the improved performance of the proposed algorithm are obtained by paying a somewhat higher computational cost. Table VI shows that our method is on average about 7 times slower than [9]. We believe such a slow-down is acceptable for an offline analysis performed at system design time, as it allows obtaining the best solution for even complex task systems within a reasonable time.

VIII. Conclusions

This paper presented a detailed method that allows converting a multi-rate task set into a single-rate DAG which meets schedulability and timing requirements. To the best of our knowledge, this approach is the most general and complete w.r.t. the methods available in the literature. The transformation process maps the whole application into a DAG, using precedence constraints for synchronizing jobs to comply with task activation periods. Multiple DAGs are generated in four stages and a pruning process is applied to exclude the ones that are inherently not feasible. The set of feasible DAGs is narrowed down using a further analysis that considers *data age* and *reaction time* bounds on specific task chains, as well as the schedulability of the system on the considered homogeneous multi-core platform. The best DAG is selected based on the weighted sum of end-to-end latencies. Most of the operations performed are based on a matrix representation of the DAG. The conversion method and a simulation tool have been implemented and made available[7]. The efficiency of the proposed approach over existing methods has been extensively validated on real experimental benchmarks. In future works, we plan to extend this model to heterogeneous platforms. Moreover, we plan to integrate the proposed approach considering predictable execution models to solve the contention problems on the memory hierarchy [28].

Acknowledgment

The research leading to these results has received funding from the European Union's Horizon 2020 Programme under the CLASS Project (https://class-project.eu/), grant agreement n° 780622. Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research.

[7]https://github.com/mive93/multi-rate-DAG

References

[1] A. Hamann, D. Dasari, S. Kramer, M. Pressler, F. Wurst, and D. Ziegenbein, "Waters industrial challenge 2017," 2017.

[2] A. Hamann, F. Dasari, Dakshina Wurst, I. Sañudo, N. Capodieci, P. Burgio, and M. Bertogna, "Waters industrial challenge 2019," 2017.

[3] A. Vincentelli, P. Giusto, C. Pinello, W. Zheng, and M. Natale, "Optimizing end-to-end latencies by adaptation of the activation events in distributed automotive systems," in *13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07)*. IEEE, 2007, pp. 293–302.

[4] M. D. Natale, H. Zeng, P. Giusto, and A. Ghosal, *Understanding and Using the Controller Area Network Communication Protocol: Theory and Practice*. Springer Publishing Company, Incorporated, 2012.

[5] H. Zeng, M. D. Natale, P. Giusto, and A. L. Sangiovanni-Vincentelli, "Statistical analysis of controller area network message response times," in *IEEE Fourth International Symposium on Industrial Embedded Systems, SIES 2009, Ecole Polytechnique Federale de Lausanne, Switzerland, July 8-10, 2009*, 2009, pp. 1–10.

[6] H. Zeng, "Probabilistic timing analysis of distributed real-time automotive systems," Ph.D. dissertation, EECS Department, University of California, Berkeley, Dec 2008.

[7] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics," in *IEEE Real-Time Systems Symposium: 30/11/2009-03/12/2009*. IEEE Communications Society, 2009.

[8] "Autosar - specification of timing extensions," Tech. Rep., 2014.

[9] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. 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)*. IEEE, 2016, pp. 159–169.

[10] ——, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *Journal of Systems Architecture*, vol. 80, pp. 104–113, 2017.

[11] ——, "Mechaniser-a timing analysis and synthesis tool for multi-rate effect chains with job-level dependencies," in *7th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems WATERS*, vol. 16, no. 05, 2016.

[12] A. Biondi and M. D. Natale, "Achieving Predictable Multicore Execution of Automotive Applications Using the LET Paradigm," in *Proceedings of the 24th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2018)*, Porto, Portugal, April 2018.

[13] J. Martinez, I. Sañudo, P. Burgio, and M. Bertogna, "End-to-end latency characterization of implicit and let communication models," in *Proc. of the 8th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, ser. WATERS, 2017.

[14] M. Xiong, S. Han, K.-Y. Lam, and D. Chen, "Deferrable scheduling for maintaining real-time data freshness: Algorithms, analysis, and results," *IEEE Transactions on Computers*, vol. 57, no. 7, pp. 952–964, 2008.

[15] D. Golomb, D. Gangadharan, S. Chen, O. Sokolsky, and I. Lee, "Data freshness over-engineering: Formulation and results," in *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 2018, pp. 174–183.

[16] Y. Saito, F. Sato, T. Azumi, S. Kato, and N. Nishio, "Rosch: Real-time scheduling framework for ros," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 52–58.

[17] Y. Suzuki, T. Azumi, S. Kato *et al.*, "Hlbs: Heterogeneous laxity-based scheduling algorithm for dag-based real-time computing," in *2016 IEEE 4th International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*. IEEE, 2016, pp. 83–88.

[18] J. Forget, F. Boniol, E. Grolleau, D. Lesens, and C. Pagetti, "Scheduling dependent periodic tasks without synchronization mechanisms," in *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2010, pp. 301–310.

[19] S. E. Saidi, N. Pernet, and Y. Sorel, "Automatic parallelization of multi-rate fmi-based co-simulation on multi-core," in *Proceedings of the Symposium on Theory of Modeling & Simulation*. Society for Computer Simulation International, 2017, p. 5.

[20] S. K. Baruah, D. Chen, S. Gorinsky, and A. K. Mok, "Generalized multiframe tasks," *Real-Time Systems*, vol. 17, pp. 5–22, 1999.

[21] N. Biggs, N. L. Biggs, and B. Norman, *Algebraic graph theory*. Cambridge university press, 1993, vol. 67.

[22] P. Purdom, "A transitive closure algorithm," *BIT Numerical Mathematics*, vol. 10, no. 1, pp. 76–94, 1970.

[23] A. V. Aho, M. R. Garey, and J. D. Ullman, "The transitive reduction of a directed graph," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 131–137, 1972.

[24] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.

[25] G. Buttazzo and A. Cervin, "Comparative assessment and evaluation of jitter control methods," in *Proceedings of the 15th conference on Real-Time and Network Systems*, 2007, pp. 163–172.

[26] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

[27] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[28] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A predictable execution model for cots-based embedded systems," in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2011, pp. 269–279.