42nd IEEE Real-Time Systems Symposium

RTSS 2021

Industry Challenge

Edited by:

Nan Guan, City University of Hong Kong

Benny Akesson, ESI (TNO) & University of Amsterdam

Zheng Dong, Wayne State University

Content

Real-Time Scheduling and Analysis of an Autonomous Driving System

IEEE RTSS 2021 Industry Challenge Problem

http://2021.rtss.org/industry-session/

Shaoshan Liu[1], Bo Yu[1], Nan Guan[2], Zheng Dong[3], and Benny Akesson[4,5]

[1]PerceptIn
[2]City University of Hong Kong
[3]Wayne State University
[4]ESI (TNO)
[5]University of Amsterdam

1 Background

The commercialization of autonomous machines, including mobile robots, drones, and autonomous vehicles is a thriving sector, and likely to be the next major computing demand driver, after PC, cloud computing, and mobile computing. However, autonomous machines are complex and safety-critical systems with strict real-time and resource constraints.

Autonomous machines have a deep processing pipeline with strong dependencies between different stages and various local and end-to-end timing constraints [1]. Figure 1 shows an example of the processing graph of an autonomous driving system. Starting from the left side, the system consumes raw sensing data from mmWave radars, LiDARs, cameras, and GNSS/IMUs, and each sensor produces raw data at a different frequency. The processing components are invoked with different frequencies, performing computation using the latest input data, and periodically producing outputs to downstream components.

The cameras capture images at 30 FPS (frame per second) and feed the raw data to the *2D Perception module*, the LiDARs capture point clouds at 10 FPS and feed the raw data to the *3D Perception module*, as well as the *Localization module*, the GNSS/IMUs generate positional updates at 100 Hz and feed the raw data to the *Localization module*, the mmWave radars detect obstacles at 10 FPS and feed the raw data to the *Perception Fusion module*. The result of 2D and 3D Perception are fed into the *Perception Fusion module* to create a

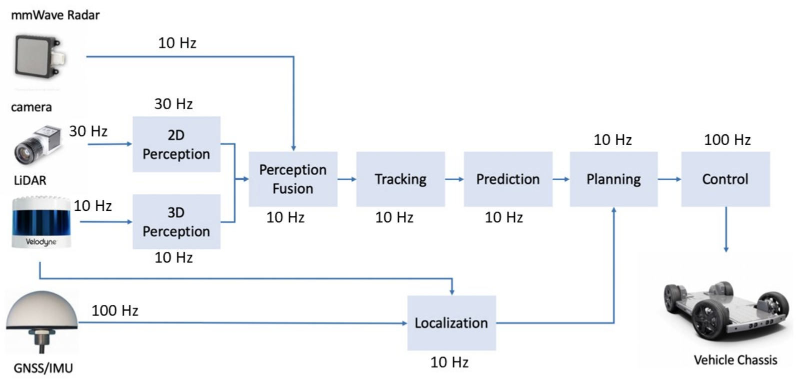


Figure 1: Processing Graph of an Autonomous Driving System

comprehensive perception list of all detected objects. The perception list is then fed into the *Tracking module* to create a tracking list of all detected objects. The tracking list then is fed into the *Prediction module* to create a prediction list of all objects. After that, both the prediction results and the localization results are fed into the *Planning module* to generate a navigation plan. The navigation plan then is fed into the *Control module* to generate control commands, which are finally sent to the autonomous vehicle for execution at 100 Hz. The computation components are deployed on different types of processing platforms. For example, the sensor data processing may be deployed on DSPs, the perception and localization may be deployed on GPUs as they typically require vector processing, and planning and control tasks can be deployed on CPUs as they mainly involve scalar processing. In general, the processing graph could be more complicated than the example shown in Figure 1. There could be more processing steps in the system. The activation frequency of each component could also be different from the example.

The system must comply with timing constraints in several aspects to guarantee that the final control command outputs can be executed correctly and timely. First, if some object appears to be close to the vehicle, it must be guaranteed that its related information can be perceived, processed, and finally used to generate control commands within a certain time limit. Second, the control command should be performed based on status information (e.g., the GNSS/IMU data) sufficiently fresh. Third, when some component receives data originating from different sensors, the time difference among the timestamps of the corresponding raw data must be no larger than a pre-defined threshold, so that information from different sensors can be synchronized and fused. There have been many attempts on the problem with *ad hoc* hardware [3] and software solutions [2], but a high-level understanding and modeling of the problem is still missing.

2 Problem Model

In the following, we introduce a problem model based on the architecture and timing constraints of real-time computing systems for autonomous machines. Note that this abstract problem model could be more general than the processing systems that we have already deployed in reality. However, studying the problem in a more general setting is meaningful to explore a larger design space and deal with more complex systems that may be developed in the future.

The system consists of a number of tasks executing on a hardware platform comprising several processing units (e.g., CPU, GPU or DSP). Each task is statically mapped to a processing unit and the mapping is fixed apriori. The worst-case execution time (WCET) of each task on the corresponding processing unit is known in advance.

Each task is activated and releases a *job* with a given frequency. The frequencies of different tasks are not necessarily the same or harmonic. Each task reads input data tokens from one or more input ports, and produce output data tokens to one output port. Tasks are connected through their input/output ports, with buffers of size 1 in between, as shown in Figure 2. The old data token in the buffer is over-written when a new data token comes. Tasks read and write data tokens from/to the buffers in a non-blocking manner. In each activation, a task reads the current data token in the buffer of each input port when it starts execution, and writes the output data token to the buffer at its output port at the end of its execution. We assume that the data communication delay among different tasks is zero or can be bounded by constants.

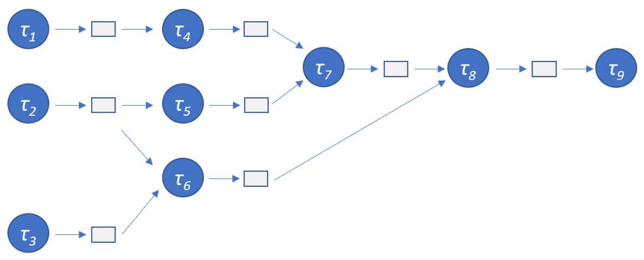
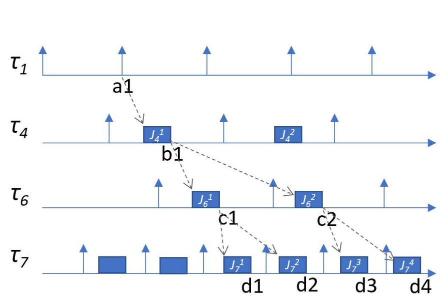
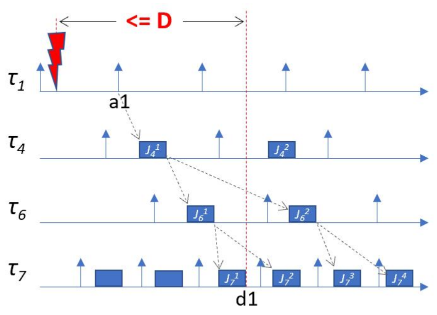


Figure 2: Task Model

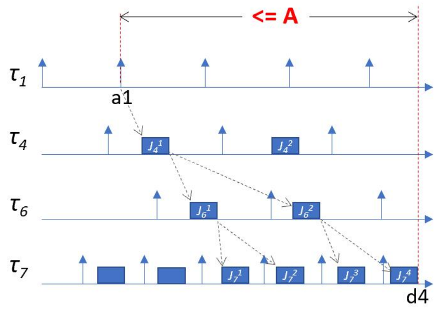
Some tasks simply generate data tokens based on a given frequency, but does not read any input data token. We call such tasks the *sensing tasks*. For example, in Figure 2, τ_1, τ_2 and τ_3 are sensing tasks. Some sensor data are *status* sensing data and some sensor data are *event* sensing data, so the sensing tasks are classified into two types: *status* sensing tasks and *event* sensing tasks. Status sensing data are used for reporting the status. Event sensing data are



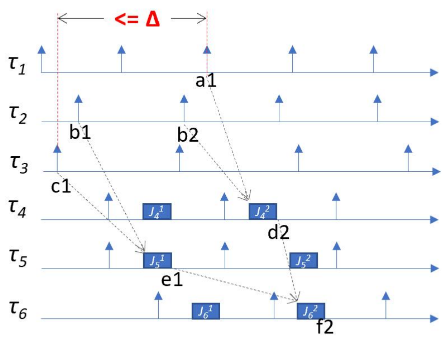
(a) *cause*, *source* and *consequence* data token



(b) **Constraint 1**



(c) **Constraint 2**



(d) **Constraint 3**

Figure 3: Illustration examples.

used for detecting some event. When an event happens, the output data token produced by the corresponding sensing task after that will capture this event. Each sensor data token is associated with a timestamp.

When a task reads several input data tokens (from different input ports) and produces an output data token, we say each of these input data tokens is the *cause* of the output data token. The original sensor data that is indirectly the cause of a data token is the *source* of this data token, and this data token is the *consequence* of the sensor data. In Figure 3-(a), job J_4^1 reads the sensor data token $a1$ produced by τ_1, and produces an output data token $b1$, so $a1$ is the cause of $b1$. $b1$ is the cause of both $c1$ and $c2$. $a1$ is the source of data tokens $b1$, $c1$, $c2$, $d1$, $d2$, $d3$ and $d4$, and these data tokens are the consequence of $a1$.

The system should satisfy the following timing constraints:

Constraint 1: For any event, it must be guaranteed that the first final data output caused by the event sensor data capturing this event is produced within a pre-defined time delay after the event occurs.

Let τ_1 in Figure 2 be an event sensing task, and an example illustrating **Constraint 1** is shown in Figure 3-(b). An event occurs after the first sensor data token produced by τ_1, and it will be detected by the second sensor data token $a1$. Job J_7^1 is the first job of τ_7 that produces a final output take token d_1 being the consequence of $a1$. The maximal time difference the occurrence of the event and when d_1 is produced should be bounded by a pre-defined value (D in this example).

Constraint 2: Let a be a sensor data token produced by some status sensing task and b be the final data output indirectly caused by a. If b is produced at t, then a must be produced no earlier than a pre-defined value before t.

Now suppose τ_1 in Figure 2 is a status sensing task, and an example illustrating **Constraint 2** is shown in Figure 3-(c). The source of the final data output $d4$ produced by job J_7^4 is the sensor data token $a1$ produced by τ_1. **Constraint 2** requires that the difference between the timestamp of $a1$ and when $d4$ is produced should be bounded by a pre-defined value (A in this example).

Constraint 3: For each task, the difference of the timestamps among all its source data tokens must be upper-bounded by a pre-defined value.

Figure 3-(d) illustrates **Constraint 3**. $f2$ has two causes, $e1$ and $d2$. $d2$ has two causes $a1$ and $b2$, and $e1$ has two causes $b1$ and $c1$. Therefore, $a1$, $b1$, $b2$ and $c1$ are all sources of $f2$. **Constraint 3** requires that the timestamps of $a1$, $b1$, $b2$ and $c1$ must be in a range bounded by a pre-defined value (Δ in this example).

3 Challenge

The problem to solve is to develop scheduling strategies and analysis techniques to guarantee the systems meet all the above timing constraints. The scheduling strategy on all processing units must be non-preemptive. We invite not only general solutions for the above described problem model, but also solutions for more restrictive versions based it. In other words, you may add more constraints to the problem model if they are helpful to improve the real-time performance and/or simply the design and analysis of the problem (e.g., the relative relation of frequency of different tasks, the maximal number of tasks mapped to each processing unit, various structural constraints of the processing graph and so on).

References

[1] Shaoshan Liu, Liyun Li, Jie Tang, Shuang Wu, and Jean-Luc Gaudiot. Creating autonomous vehicle systems. *Synthesis Lectures on Computer Science*, 8(2):i–216, 2020.

[2] Hsin-Hsuan Sung, Yuanchao Xu, Jiexiong Guan, Wei Niu, Shaoshan Liu, Bin Ren, Yanzhi Wang, and Xipeng Shen. Enabling level-4 autonomous driving on a single \$1k off-the-shelf card. *arXiv preprint arXiv:2110.06373*, 2021.

[3] Bo Yu, Wei Hu, Leimeng Xu, Jie Tang, Shaoshan Liu, and Yuhao Zhu. Building the computing system for autonomous micromobility vehicles: Design constraints and architectural optimizations. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1067–1081. IEEE, 2020.

End-To-End Processing Chain Analysis

Mario Günzel*, Niklas Ueter*, Kuan-Hsun Chen†, Georg von der Brüggen‡, Junjie Shi*, Jian-Jia Chen*
*TU Dortmund University, Germany, {mario.guenzel, niklas.ueter, junjie.shi, jian-jia.chen}@tu-dortmund.de
† University of Twente, The Netherlands, {k.h.chen}@utwente.nl
‡Max Planck Institute for Software Systems, Germany, {vdb}@mpi-sws.org

***Abstract*—In this paper we address the timing verification for processing graphs used to implement autonomous driving systems by the company *Perceptin*. We demonstrate how to extend our previous results on end-to-end timing analysis in terms of maximum reaction time and maximum data age for cause-effect chains in automotive systems for the posed challenge.**

I. Introduction

Autonomous driving systems require to implement complex functions consisting of data dependent modules as illustrated in the processing graph in Figure 1. The nodes in the processing graph denote functional modules such as sensor data pre-processing, perception, tracking, trajectory planning, and control. The directed edges in the processing graph denote the data dependencies between the modules, e.g., the data produced by the *localization* module is used by the *planning* module in its computation. In the processing graph model, each module is implemented as an individual task, that is activated either periodically or sporadically and can be scheduled on a designated hardware component. The data is communicated implicitly and asynchronously, i.e., the data token is read from and written to a shared resource by each task individually whenever they start and finish their executions.

With respect to guaranteed quality of service, the processing graph must comply with several aspects of timing constraints. For instance, the control command must be executed in time (i.e., each task instance must be finished before its deadline) and react timely to events such as object detection. More precisely, a trajectory must be planned and tracked after a pedestrian is detected by the camera system and the appropriate actuation must be commanded within bounded time. Furthermore, it must be guaranteed that the control command is based on the freshest available sensor data, and the time stamp difference of the data samples used for sensor fusion algorithms can only differ by a maximum specified value.

These kinds of timing constraints have been researched intensively in the context of cause-effect chains in AUTOSAR compliant automotive systems, e.g., [1]–[6]. In the literature, the concepts of maximum reaction time and maximum data age are used to address end-to-end latencies of processing chains and to analyze the freshness of the data. The maximum reaction time refers to the longest latency of the first response (reaction) of a system to a corresponding stimulus (cause), e.g., the latency from a camera sample to an object detection, or to the final actuation. The maximum data age denotes the largest time interval between the sampling time of a data token by a task until the last point in time when the system produces an output related to that data token.

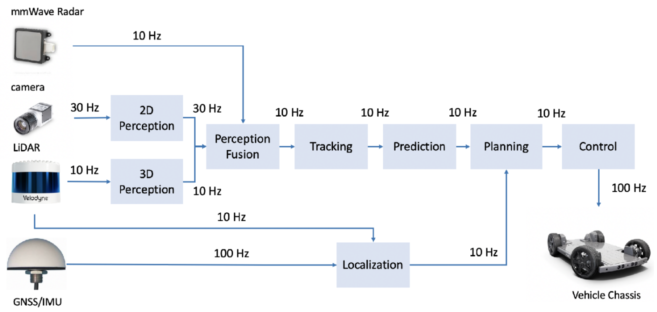


Figure 1: Exemplary processing graph as posed in the RTSS 2021 Industry challenge. Each block in the graph denotes a sporadic or periodic task that is scheduled and executed non-preemptively on a dedicated hardware with the annotated frequency. Data tokens are propagated asynchronously via buffers where a producer writes a data token at the end of its execution and a data token consumer reads data at its beginning.

Most published results in this context only consider preemptive scheduling policies. Nevertheless, we conjecture that the results in the literature may be extendable without too much effort. In this paper, we extend our previous results in the context of end-to-end timing analyses of automotive cause-effect chains [2], show how to address all posed objectives.

Our Contributions:

- We reduce all posed objectives in the industry challenge into maximum reaction time and maximum data age problems and propose an analysis for maximal time stamp differences of multiple sources of data tokens that merge in a common task.

II. Problem Definition

The architecture and task system of the proposed autonomous driving system is formally described as a number of tasks $\tau_i \in \mathbb{T}$ that are statically mapped onto different processing units, e.g., CPU (Central Processing Unit), GPU (Graphics Processing Unit), or DSP (Digital Signal Processor) that are available on the autonomous driving hardware platform.

Definition 1 (Sporadic Task)**.** Each sporadic task is defined by the 4-tuple $\tau_i = (C_i^{min}, C_i^{max}, T_i^{min}, T_i^{max})$ where $C_i^{max} \leq T_i^{min}$. Each task releases an infinite sequence of instances (jobs) $\tau_{i,\ell}$ with execution time $C_{i,\ell}$ (with $C_i^{\min} \leq C_i \leq C_i^{max}$) where the inter-arrival time between two job releases is at least T_i^{min} and at most T_i^{max}. The worst-case execution time (WCET) is given by C_i^{max} and best-case execution time (BCET) of each task is given by C_i^{min}.

In our model a periodic task is a specialization of a sporadic task where T_i^{min} coincides with T_i^{max}.

Definition 2 (Chain)**.** Let $E_j = \tau_{j_1} \to \tau_{j_2} \to \ldots \to \tau_{j_{K_j}}$ denote a chain (path) in the processing graph such that τ_{j_2} reads the data token produced by τ_{j_1}, τ_{j_3} reads the data token produced by τ_{j_2}, etc.

Note that a task τ_i can be part of multiple chains, e.g., $\tau_{k_\ell} = \tau_{j_z} = \tau_i$, which implies that task $\tau_i \in \mathbb{T}$ is the ℓ-th task in chain E_k and the z-th task in chain E_j.

Two adjacent tasks in a chain E_j, e.g., τ_{j_i} and $\tau_{j_{i+1}}$, have immediate data dependency. That is, a job of task $\tau_{j_{i+1}}$ consumes the latest available data token produced by a job of task τ_{j_i}. This relation is illustrated by the directed edges in the processing graph in Figure 1.

Each data token is stored in a buffer that can be read and written atomically without synchronization. The old data token in the buffer is over-written when a new data token is produced. Each task reads the latest data token from each of its (many) input ports and produces output data tokens that are written to their designated buffers in its output ports. Under this scheme, each task reads and writes data from and to the buffers in a non-blocking manner. In each task activation (job), a job reads the current data token of each input port when it starts execution and writes the output data token to the buffer of its port at the end of its execution.

Tasks that periodically generate data tokens but do not read any data tokens are called *sensing tasks*. The authors of the challenge further distinguish between *status sensing tasks* and *event sensing tasks*. When an event occurs, the designated *event sensing task* is triggered and produces relevant data tokens. From our analysis point of view, status sensing tasks are periodic and event sensing tasks are sporadic.

III. Verification Constraints & Analysis

Given a processing graph and a mapping of each task onto hardware components, the following timing constraints must be satisfied:

Maximum Reaction Time: For any event, it must be guaranteed that the first final data output caused by the event sensor data capturing this event is produced within a predefined time delay after the event occurs.

Maximum Data Age: For each final output data, the time between the output and the sensor data token leading to that output indirectly is bounded.

Maximum Time Stamp Difference: For each task, the difference of the time stamps among all its source data tokens must be bounded from above by a predefined value.

The objective in the remainder of this paper is to verify if the above constraints can be satisfied for a given set of processing chains, sub-chains, task priorities, and mappings. Moreover, we assume that the scheduling algorithms on each processing unit is non-preemptive, i.e., once a job is scheduled to be executed, that job is run until completion.

The provided analyses for maximum reaction time, maximum data age, and maximum time stamp difference are based on our previous analyses described in [2]. The main idea is to construct **immediate forward and immediate backward job chains** that consist of the all the jobs (of tasks that are involved in the data propagation of interest) along the direction of data propagation. An immediate backward job chain is constructed in an iterative manner by starting with a job of the last task in a processing chain and then collecting the latest job that produced the consumed data token. Analogously, an immediate forward job chain starts from a job of the first task in a processing chain and the iterative process of collecting the earliest job (of an adjacent task in the processing chain), that consumes the produced data token.

Whereas there are already analyses to validate the maximum reaction time and the maximum data age constraint, presented in Section III-A and Section III-B, for the maximum time stamp difference, we need to reduce this problem to computation of maximum data age, presented in Section III-C.

A. Constraint Maximum Reaction Time

To compute the maximum reaction time according to [2], we need to rely on worst-case response times R_{j_i} for each task τ_{j_i} in the chain E_j. They can be computed beforehand by appropriate analyses on the respective processing units.

Theorem 3 (Maximum Reaction Time [2])**.** *The* **maximum reaction time** *of chain E_j is no more than*

$$MRT(E_j) := T_{j_1}^{max} + R_{j_{K_j}} + \sum_{i=1}^{K_j-1} \max\{R_{j_i},\, T_{j_{i+1}}^{max} + R_{j_i} \cdot \sigma\} \tag{1}$$

where the predicate σ evaluates to 1 if either of the following conditions are satisfied:

- *τ_{j_i} has lower priority than $\tau_{j_{i+1}}$*
- *τ_{j_i} and $\tau_{j_{i+1}}$ are scheduled on different processing units*

and 0 otherwise. □

B. Constraint Maximum Data Age

As for the maximum reaction time, for the maximum data age upper bounds on the worst-case response times for each task in E_j have to be provided beforehand as well.

Theorem 4 (Maximum Data Age [2])**.** *The* **maximum data age** *of chain E_j is no more than*

$$MDA(E_j) := R_{j_{K_j}} + \sum_{i=1}^{K_j-1} (T_{j_i}^{max} + R_{j_i} \cdot \sigma) \tag{2}$$

where the predicate σ evaluates to 1 if either of the following conditions are satisfied:

- *τ_{j_i} has lower priority than $\tau_{j_{i+1}}$*
- *τ_{j_i} and $\tau_{j_{i+1}}$ are scheduled on different processing units*

and 0 otherwise. □

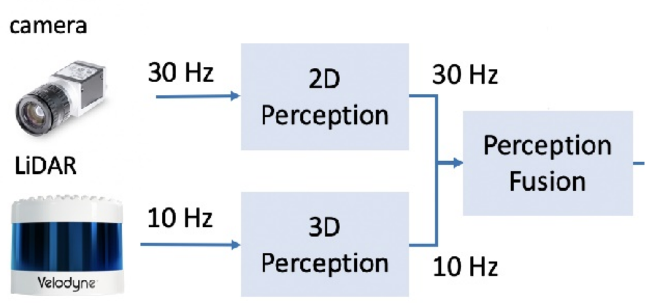


Figure 2: Two processing-chains merge in task *Perception Fusion*. The time stamp difference between the two source data tokens must be bounded from above.

C. Maximum Time Stamp Difference

The maximum time stamp difference constraint requires that the maximal difference between any two time stamps of data tokens that are consumed at a job of a merging task is bounded from above. For instance, there are two processing sub-chains E_1 and E_2 that merge at the *Perception Fusion* task as depicted in Figure 2, i.e., $E_1 =$ *Camera* $\rightarrow$ *2D Perception* $\rightarrow$ *Perception Fusion* $=: (\tau_1, \tau_2, \tau_3)$ and $E_2 =$ *LiDAR* $\rightarrow$ *3D Perception* $\rightarrow$ *Perception Fusion* $=: (\tau_4, \tau_5, \tau_3)$. The constraint requires that the time stamp of the data tokens which are produced by the *Camera* (τ_1) and by the *LIDAR* (τ_4) and used indirectly when starting a job of *Perception Fusion* (τ_3) have a bounded difference.

We denote by $E_j = \tau_{j_1} \rightarrow \tau_{j_2} \rightarrow \ldots \rightarrow \tau_{j_{K_j}}$ a sub-chain. Let $\{E_j \,|\, j \in \{1, \ldots, P\}\}$ be a set of sub-chains that merge into one identical task. For the analysis, we need to

1) find all jobs of τ_{j_1} that produce a data token, which is consumed by a job of $\tau_{j_{K_j}}$ and
2) bound the age of that data token at the time point at which it is used by $\tau_{j_{K_j}}$.

If we provide an upper bound δ_j and a lower bound ρ_j on the time stamp age for each sub-chain E_j, then the **maximum time stamp difference** can be computed by

$$\max_{i \neq j \in \{1,\ldots,P\}} (\delta_j - \rho_i). \tag{3}$$

The first objective 1) can be achieved by computing the backward job chains of E_j, i.e., each job of τ_{j_1} that produces a data token which is consumed of $\tau_{j_{K_j}}$ is at the start of an immediate backward job chain of E_j. In the following, let $\overleftarrow{c}_{j,k} = (J_{j_1,k_1} \rightarrow J_{j_2,k_2} \rightarrow \ldots \rightarrow J_{j_{K_j},k_{K_j}} = J_{j_{K_j},k})$ denote the k-th immediate backward job chain of E_j.

For the second objective 2), we consider the **time stamp age** for $\overleftarrow{c}_{j,k}$, which is given by $s_{j_{K_j},k} - f_{j_1,k_1}$. , i.e., the difference between start of the last job and finish of the first job in the chain. Since for job chains we have the property $f_{j_i,k_i} \leq s_{j_{i+1},k_{i+1}}$, we derive a **lower bound** ρ_j for the time stamp age by

$$\begin{aligned} s_{j_{K_j},k} - f_{j_1,k_1} &\geq f_{j_{K_j-1},k_{K_j-1}} - s_{j_2,k_2} && (4)\\ &\geq \sum_{i=2}^{K_j-1} (f_{j_i,k_i} - s_{j_i,k_i}) && (5)\\ &\geq \sum_{i=2}^{K_j-2} C_{j_i}^{min} =: \rho_j. && (6) \end{aligned}$$

On the other hand, we can express an **upper bound** δ_j by utilizing the maximum data age

$$\begin{aligned} s_{j_{K_j},k} - f_{j_1,k_1} &\leq (f_{j_{K_j},k} - C_{j_{K_j}}^{min}) - (s_{j_1,k_1} + C_{j_1}^{min}) && (7)\\ &= (f_{j_{K_j},k} - s_{j_1,k_1}) - (C_{j_{K_j}}^{min} + C_{j_1}^{min}) && (8)\\ &\leq MDA(E_j) - (C_{j_{K_j}}^{min} + C_{j_1}^{min}) =: \delta_j. && (9) \end{aligned}$$

Since the upper and lower bound δ_j and ρ_j are valid for all jobs chains, they can be used in the computation of the maximum time stamp difference in Equation (3).

We make the **conjecture**, that there is even a tighter upper bound given by

$$MDA(E_j) - (C_{j_{K_j}}^{max} + C_{j_1}^{min}) =: \delta_j. \tag{10}$$

An artificial enlargement of the execution interval of the last job $J_{j_{K_j},k}$ in the sequence $\overleftarrow{c}_{j,k}$ to $C_{j_{K_j}}^{max}$ leads to the inequality $s_{j_{K_j},k} \leq f_{j_{K_j},k} - C_{j_{K_j}}^{max}$ which can be used instead of $s_{j_{K_j},k} \leq f_{j_{K_j},k} - C_{j_{K_j}}^{min}$ in (7). This artifical enlargement does not affect value of $s_{j_{K_j},k} - f_{j_1,k_1}$ and can therefore safely be considered. We plan to prove that tighter upper bound formally in future work.

References

[1] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte. Synthesizing job-level dependencies for automotive multi-rate effect chains. In *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 159–169, 2016.

[2] M. Dürr, G. von der Brüggen, K.-H. Chen, and J.-J. Chen. End-to-end timing analysis of sporadic cause-effect chains in distributed systems. *ACM Trans. Embedded Comput. Syst. (Special Issue for CASES)*, 18(5s):58:1–58:24, 2019.

[3] 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 *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, 2009.

[4] M. Günzel, K. Chen, N. Ueter, G. von der Brüggen, M. Dürr, and J. Chen. Timing analysis of asynchronized distributed cause-effect chains. In *RTAS*, pages 40–52. IEEE, 2021.

[5] A. Rajeev, S. Mohalik, M. G. Dixit, D. B. Chokshi, and S. Ramesh. Schedulability and end-to-end latency in distributed ecu networks: formal modeling and precise estimation. In *International Conference on Embedded Software*, pages 129–138, 2010.

[6] R. Wyss, F. Boniol, C. Pagetti, and J. Forget. End-to-end latency computation in a multi-periodic design. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC*, pages 1682–1687, 2013.

Analysis of Time difference among Sensing Data in Autonomous Driving Systems

Xu Jiang†, Yue Tang†, Dong Ji†
† Northeastern University, China

***Abstract*—The 2021 RTSS Industry challenge considers timing constraints in autonomous driving systems. One real-time requirement is that the differences of timestamps among all source data of an output must be upper-bounded by a pre-defined value, to make sure that data from different sources are correctly fused, e.g, object detection by using different sensing data. In this paper, we formally describe the challenge system as well as the requirements. In particular, we present timing analysis to bound the maximum differences of timestamps among all sources for an output. We show that an output may possibly be caused by two different data from the same source, when fork-join structure exists. This result suggests that data fusion after different long pipelined processing procedure must be avoided to meet required timing constraints.**

I. Introduction

The 2021 RTSS Industry challenge considers timing constraints in autonomous driving systems. The objective of the challenge is to study the possible scheduling strategies and analysis techniques to guarantee the systems to meet all the mentioned timing constraints.

An autonomous machine operates on its own to complete tasks without human control or invertion. Autonomous machines have become increasing popular in various fields, such as smart factories and autonomous driving, and undoubtedly drawed much attention from both industry and research. In autonomous machines, a control path usually covers multiple steps that span over multiple software and hardware components, and satisfying the end-to-end timing constraints of control paths is a prerequisite for correct and safe systems. Take autonomous vehicles as example, the completion of obstacle avoidance depends on a chain of local tasks including sensing, perception, planning and control, and the success of obstacle avoidance requires that the task chain finishes before predefined deadline. For performance evaluation, testing and formal analysis are two major approaches. However, testing cannot fully cover the runtime behavior space of a system and thus does not provide hard real-time guarantee. This is not acceptable to safe-critical autonomous machines, where timing errors may lead to catastrophic consequences such as loss of human life. As a consequence, formal modeling and analysis must be performed to guarantee that the timing constraints are always honored at run-time. Although there has been a large amount of work targeting scheduling design and anslysis in real-time community, most of them does not fully explore critical factors in real applications, and thus can not be directly applied to autonomous machines.

In this paper, we first model the challenge system as a cause-effect graph, and then formally define the considered real-time constraints. Two of the constraints are considered as the problems of bounding the maximum reaction time and maximum data age, which have been formally defined and studied in previous literature. To address the third problem, we extend the concept of immediate backward job chains proposed in [1] to immediate backward job graph, and formally define the source differ of an immediate backward job graph. Then the third problem in the challenge is defined as bounding the maximum source differ among all possible immediate backward job graphs. Analysis techniques are developed to bound the maximum source differ of immediate backward job graphs. The results show that the source differ may significantly grow with the propagation of data through different cause-effect chains. In particular, the source differ could also be significant even if the source data are produced by the same task. This suggests that data fusion after different cause-effect chains with long pipelines must be avoided to meet required timing constraints.

II. Preliminary

The challenge model is inspired by realistic software in autonomous driving systems which have already been deployed in reality. In the following, we first review the challenge model and then present a formal characterization.

A. Challenge Model and Problems

The system is considered to be composed of several tasks, which are executing on different processing units, e.g., GPU and CPU. Each task is statically assigned onto a processing unit, and is activated periodically according to a given frequency. Each task reads input data tokens from one or multiple input channels, and produces output data tokens into one output channel. Tasks are connected with input/output channels in between. Each channel can be considered as a register, where the old data is over-written when the new data arrives. When activated, a task reads data from all its input channels and produces output data into its output channel. In particular, some components only produce output data tokens (but do not read). Such tasks are called sensing tasks, which are classified into two types: status sensing tasks generating status data tokens to report status, and event sensing tasks detecting events to generate event data tokens. When an event happens, all the output data token produced by the corresponding sensing task after that will capture this event.

When a task reads several input data tokens (from different input ports) and produces an output data token, the input data token is called the *cause* of the output data token. The original data token indirectly being the cause of a data token is the *source* of the data token. Each sensor data token is associated with a *timestamp* labeled by the time when it is produced.

Given such a system, the real-time problems of the challenge are summarized as follows:

- For each task, the difference of the timestamps among all its source data tokens must be upper-bounded by a given value.
- For any event, it must be guaranteed that the first final data output caused by the event sensor data capturing this event is produced within a pre-defined time delay after the event occurs.
- For a final output data token produced at time t, any source data token of it must be produced no earlier than a pre-defined value before t.

B. Cause-Effect Graph

Inspired by the non-blocking executing scenario and the pipeline of tasks, the system can be modeled as a directed acyclic graph (DAG), called *cause-effect graph*, denoted by $G = \langle V, E \rangle$, where V is the set of vertices and E the set of edges. A vertex v in V corresponds to a task τ_i. An edge $(u, v) \in E$ corresponds to the input channel/output channel of v/u and represents the data dependency between u and v. We call u a *predecessor* of v, and v a *successor* of u. A *cause-effect chain* is a path in G, i.e., a sequence of tasks $\pi = \{v_1, v_2, \cdots, v_p\}$ where v_j is a predecessor of v_{j+1} for each pair of consecutive elements v_j and v_{j+1} in π. We use I_k^π to denote the index of the k^{th} task in the cause-effect chain π. A task with multiple incoming edges is called a *fusion* task.

A *complete path* in G is a path in G that starts with a vertex with no predecessors and ends with a vertex with no successors. A task τ_i is characterized by a tuple (W_i, B_i, T_i). W_i and B_i denote the worst-case execution time (WCET) and best-case execution time (BCET) of τ_i, respectively, which are known in advance. T_i is the period of τ_i. In particular, we assume $W_i = B_i = 0$ if τ_i has no predecessors, i.e., a source task in the graph.

An example of the system is shown in Fig. 1. The number below each vertex denotes period of the corresponding task.

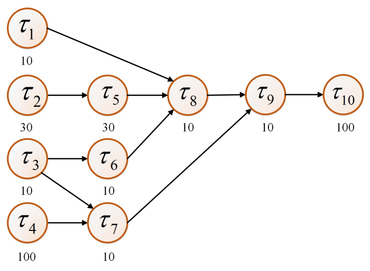


Fig. 1. An example of the system.

C. Run-Time Behavior

Each task τ_i is statically assigned on a processing unit. During run-time, τ_i releases an infinite sequence of *jobs*. The *period* T_i of τ_i is the time difference between the release times of two successive jobs of τ_i. Without loss of generality, we assume the system starts at time 0, and use J_i^k to denote the k^{th} job of task τ_i. Let r_i^k and f_i^k denote the release time and finish time of J_i^k. All tasks assigned on the same processing unit are scheduled together by a scheduler. Both migration and preemption are forbidden. In this paper, we do not specify the scheduler adopted on each processing unit. A job reads the current data in the register of each input channel when it starts execution, and writes the output data token to its output channel at the end of its execution, i.e., the *implicit communication* in automotive systems. In particular, if J_i^k is a job of a task τ_i without any predecessor, the output data token produced by this job is assumed to be written into its output channel at its release time r_i^k.

The worst-case response time (WCRT) of τ_i is the longest time interval between the release and finishing time of all its jobs, denoted by R_i. Note that, R_i is decided by the scheduler and other tasks scheduled together with τ_i, and can be upper bounded by some analysis techniques presented in previous literature [2]–[4]. In this paper, we do not consider the impact of scheduler to the challenge problems. For the rest of this paper, we assume that for every task τ_i in G, its WCRT is no more than T_i.

D. Job Chain and Job Graph

A *job chain* of a cause-effect chain π in G is a sequence of jobs with data dependency, i.e., a job reads the data token generated by its predecessor in the job chain for each pair of successive jobs. In particular, several job chains can merged into a *job graph* when they contain the same job.

Definition 1 (immediate forward job chain)**.** *An immediate forward job chain that starts from the k^{th} job of the first task $\tau_{I_1^\pi}$ (recall that I_1^π is the index of the first task in π) in the cause-effect chain π, denoted as $\overrightarrow{\pi_k}$, is defined iteratively from the job $J_{I_1^\pi,k}$. Let $r_{I_1^\pi}^k$ and $f_{I_1^\pi}^k$ denote the release time and the finish time of $J_{I_1^\pi}^k$ respectively. For each $i = 2, \cdots, |\pi|$, the i^{th} job in the $\overrightarrow{\pi_k}$ is the first job of $\tau_{I_i^\pi}$ that starts its execution no earlier than $f_{I_{i-1}^\pi}^k$, whose release time and finish time are denoted by $r_{I_i^\pi}^k$ and $f_{I_i^\pi}^k$. The* forward time *of the immediate forward job chain $\overrightarrow{\pi_k}$ is denoted by $len(\overrightarrow{\pi_k}) = f_{I_{|\pi|}^\pi}^k - r_{I_1^\pi}^k$.*

The worst-case forward time (WCFT) $\mathcal{F}_\pi$ of a cause-effect chain π is the maximum forward time of all possible immediate forward job chains of π:

$$\mathcal{F}_\pi = \max_{\forall k} len(\overrightarrow{\pi_k}).$$

Definition 2 (immediate backward job chain)**.** *An immediate backward job chain that ends at the k^{th} job of the last task $\tau_{I_{|\pi|}^\pi}$ (recall that $I_{|\pi|}^\pi$ is the index of the last task in π) in the cause-effect chain π, denoted as $\overleftarrow{\pi_k}$, is defined iteratively* *from the job $J_{I^{\pi}_{|\pi|},k}$. Let $r^k_{I^{\pi}_{|\pi|}}$ and $f^k_{I^{\pi}_{|\pi|}}$ denote the release time and the finish time of $J^k_{I^{\pi}_{|\pi|}}$ respectively. For each $i = 1, \cdots, |\pi|-1$, the i^{th} job in the $\overleftarrow{\pi_k}$ is the last job of $\tau_{I^{\pi}_i}$ that finishes its execution no later than the start time of $J^k_{I^{\pi}_i}$, whose release time and finish time are denoted by $r^k_{I^{\pi}_i}$ and $f^k_{I^{\pi}_i}$. The* backward time *of the immediate back job chain $\overleftarrow{\pi_k}$ is denoted by $len(\overleftarrow{\pi_k}) = f^k_{I^{\pi}_{|\pi|}} - r^k_{I^{\pi}_1}$. In particular, we let $len(\overleftarrow{\pi_k}) = 0$ if there is no immediate backward job chain ending at the k^{th} job of the last task $\tau_{I^{\pi}_{|\pi|}}$.*

The worst-case backward time (WCBT) $\mathcal{B}_\pi$ of a cause-effect chain π is the maximum backward time of all possible immediate backward job chains of π:

$$\mathcal{B}_\pi = \max_{\forall k} len(\overleftarrow{\pi_k}).$$

Since the last problem in the challenge refers to multiple cause-effect chains, we extend the concept of the immediate backward job chains to the *immediate backward job graph*. Let Π denote the set of cause-effect chains that end at the same task. When considering multiple cause-effect chains, we define the immediate backward job graph as follows.

Definition 3 (immediate backward job graph)**.** *The immediate backward job graph that ends at the k^{th} job of the common last task of all cause-effect chains in Π, denoted as $\overleftarrow{\Pi_k}$, is the joint graph by the immediate backward job chain $\overleftarrow{\pi_k}$ of each cause-effect chain π in Π.*

Let $\mathcal{S}^k_\Pi$ denote the set of all jobs of source tasks in $\overleftarrow{\Pi_k}$, and $\mathcal{P}^k_\Pi$ be the set of timestamps of jobs in $\mathcal{S}^k_\Pi$, the *source differ* of $\overleftarrow{\Pi_k}$ is defined as $\mathcal{O}_{\Pi_k} = \max_{\forall s_i, s_j \in \mathcal{S}^k_\Pi} |s_i - s_j|$.

The worst-case source differ (WCSD) $\mathcal{D}_\Pi$ of a set of cause-effect chains Π is the maximum source differ of all possible immediate backward job graph of Π:

$$\mathcal{D}_\Pi = \max_{\forall k} \mathcal{O}_{\Pi_k}.$$

E. Problem Definitions

To address the challenge problem, we need to analyze the worst-case time interval from cause to effect, i.e., the time interval from the moment where the first task in a cause-effect chain starts executing until time point when the last task in a cause-effect chain finishes. Two latency semantics are of specific interest: the maximum reaction time and maximum data age as introduced by Feiertag et al. [5]. The maximum reaction time refers to the first response of the system to an external cause, e.g., a button press or a value change of a register. Thus, the time interval between the worst-case occurrence of a cause and the first corresponding output of the system needs to be analyzed, which corresponds to the first problem in the challenge. The WCFT corresponds to the maximum reaction time.

The maximum data age is the time interval between the moment a cause task produces a source data token until the last time point when the effect task produces an output data token caused by the source data token, which corresponds to the second problem in the challenge. The WCBT of a cause-effect chain corresponds to the maximum data age.

The third problem in the challenge refers to the time differences among timestamps of all source jobs for an output data token, thus corresponding to the WCSD of a set of a cause-effect chains ending at the same task.

III. Analysis

Analysis techniques for bounding the maximum data age and reaction time are proposed in several previous literature, e.g., [6]–[10]. So in this paper, we focus on the third problem and then investigate the influence of the design to these different timing constraints.

We first consider a set Π of cause-effect chains comprising two cause-effect chains λ and π ending at the same task. Then it is satisfied:

Lemma 1. *Let $\Delta^U_\lambda = \sum_{i=1}^{|\lambda|-1}(T_{I^\lambda_i} + R_{I^\lambda_i}) + R_{I^\lambda_{|\lambda|}}$ and $\Delta^L_\lambda = \sum_{i=1}^{|\lambda|} B_{I^\lambda_i}$, the WCSD of Π is upper-bounded by*

$$\mathcal{D}_\Pi \leq \max\{\Delta^U_\lambda - \Delta^L_\pi, \Delta^U_\pi - \Delta^L_\lambda\}$$

Proof. Considering an arbitrary job $J_{I^{\pi}_{|\pi|},k}$ of the last task of both λ and π, we construct the immediate backward job graph $\overleftarrow{\Pi_k}$ of Π. By definition, the $(|\lambda|-1)^{th}$ job in the graph on λ is the last job of $\tau_{I^\lambda_{|\lambda|-1}}$ that finishes its execution no later than the start time of $J_{I^{\pi}_{|\pi|},k}$, so we know:

$$r^\lambda_{I^\lambda_{|\lambda|}} - r^\lambda_{I^\lambda_{|\lambda|-1}} \leq T_{I^\lambda_{|\lambda|-1}} + R_{I^\lambda_{|\lambda|-1}}$$

and

$$r^\lambda_{I^\lambda_{|\lambda|}} - r^\lambda_{I^\lambda_{|\lambda|-1}} \geq B_{I^\lambda_{|\lambda|-1}}$$

Similarly, for each $i = 1, 2, \cdots, |\lambda|-1$, we have:

$$r^\lambda_{I^\lambda_i} - r^\lambda_{I^\lambda_{i-1}} \leq T_{I^\lambda_{i-1}} + R_{I^\lambda_{i-1}}$$

and

$$r^\lambda_{I^\lambda_i} - r^\lambda_{I^\lambda_{i-1}} \geq B_{I^\lambda_{i-1}}$$

The it must be satisfied:

$$r^\lambda_{I^\lambda_{|\lambda|}} - r^\lambda_{I^\lambda_1} \leq \sum_{i=1}^{|\lambda|-1}(T_{I^\lambda_i} + R_{I^\lambda_i})$$

and

$$r^\lambda_{I^\lambda_{|\lambda|}} - r^\lambda_{I^\lambda_1} \geq \sum_{i=1}^{|\lambda|-1} B_{I^\lambda_i}$$

Since $B_{I^\lambda_{|\lambda|}} \leq f^\lambda_{I^\lambda_{|\lambda|}} - r^\lambda_{I^\lambda_{|\lambda|}} \leq R_{I^\lambda_{|\lambda|}}$, we have:

$$len(\overleftarrow{\lambda_k}) \leq \sum_{i=1}^{|\lambda|-1}(T_{I^\lambda_i} + R_{I^\lambda_i}) + R_{I^\lambda_{|\lambda|}} = \Delta^U_\lambda$$

and

$$len(\overleftarrow{\lambda_k}) \geq \sum_{i=1}^{|\lambda|} B_{I_i^\lambda} = \Delta_\lambda^L$$

Similarly, we know

$$len(\overleftarrow{\pi_k}) \leq \Delta_\pi^U \textit{ and } len(\overleftarrow{\pi_k}) \geq \Delta_\pi^L$$

Then it is satisfied:

$$\mathcal{O}_{\Pi_k} = \max_{\forall s_i, s_j \in \mathcal{S}_\Pi^k} |s_i - s_j| \leq \max\{\Delta_\lambda^U - \Delta_\pi^L, \Delta_\pi^U - \Delta_\lambda^L\}$$

Since $J_{I_{|\pi|}^\pi, k}$ is an arbitrary job, the lemma is proved. □

Since the time differ does not propagate on the same job chain, when computing the WCSD of two chains, we only need to construct sub-graph of these two chains until their last joint point. It can be observed that, the source differ could grow with the propagation of data tokens through different cause-effect chains.

In particular, if λ and π share the same source task τ_k, the following lemma can be given.

Lemma 2. *The WCSD of* Π *is upper-bounded by*

$$\mathcal{D}_\Pi \leq \lfloor \frac{\max\{\Delta_\lambda^U - \Delta_\pi^L, \Delta_\pi^U - \Delta_\lambda^L\}}{T_k} \rfloor T_k$$

Proof. Since λ and π share the same source task τ_k, the time difference between the release times of two jobs must be an integer multiple of its period. □

It can be observed from Lemma 2, the source differ could also be significant even if the source data tokens are produced by the same task.

At last, given a task and all its source tasks being considered, let Π denote the set of all cause-effect chains that start from one source task and end at the considered task, we enumerate each combination of two cause-effect chains π_i and π_j in Π and put them into the set Π' and then compute the upper bound of $\mathcal{D}_{\Pi'}$ according to Lemma 1 or Lemma 2. The source differ of Π is upper-bounded by $\mathcal{D}_\Pi \leq \max_{\forall \Pi'} \mathcal{D}_{\Pi'}$.

IV. CONCLUSION

In this paper, we first formally model the challenge system as a cause-effect graph, and then formally define the problems. Two problems correspond to bounding the maximum reaction time and maximum data age which have been formally defined and studied in previous literature. To address the third problem, we extend the concept of immediate backward job chains proposed in [1] to immediate backward job graph, and formally define the source differ of an immediate backward job graph. Then the third problem in the challenge is defined as bounding the maximum source differ among all possible immediate backward job graphs. We present analysis techniques to bound the maximum source differ of immediate backward job graphs. The results show that the source differ may significantly grow with the propagation of data through different cause-effect chains. In particular, the source differ could also be significant even if the source data are produced by the same task. This suggests that data fusion after different cause-effect chains with long pipelines must be avoided to meet required timing constraints.

REFERENCES

[1] M. Dürr, G. V. D. Brüggen, K. Chen, and J. Chen, "End-to-end timing analysis of sporadic cause-effect chains in distributed systems," *ACM Transactions on Embedded Computing Systems (TECS)*, 2019.

[2] G. R.L., "Bounds on multiprocessing timing anomalies," *SIAM journal on Applied Mathematics*, 1969.

[3] A. Melani, M. Bertogna, and et.al, "Schedulability analysis of conditional parallel task graphs in multicore systems," *IEEE Trans on Computers*, 2017.

[4] X. Jiang, N. Guan, X. Long, and W. Yi, "Semi-federated scheduling of parallel real-time tasks on multiprocessors," in *2017 IEEE Real-Time Systems Symposium (RTSS)*, Dec 2017, pp. 80–91.

[5] 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 *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems, 2009.*

[6] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *JSA*, 2017.

[7] ——, "Analyzing end-to-end delays in automotive systems at various levels of timing information," *REACTION*, 2016.

[8] S. Schliecker and R. Ernst, "A recursive approach to end-to-end path latency computation in heterogeneous multiprocessor systems," in *CODES, 2009.*

[9] M. Günzel, K. Chen, N. Ueter, G. Brüggen, M. Dürr, and J. Chen, "Timing analysis of asynchronized distributed cause-effect chains," in *RTAS, 2021.*

[10] A. Kordon and N. Tang, "Evaluation of the age latency of a real-time communicating system using the let paradigm," in *ECRTS, 2020.*

Timing Analysis for Dependent Tasks in Distributed Embedded Systems

Fei Guan
Department of Computer Science and Technology
Northeast Forestry University
Harbin, China
fei.guan@nefu.edu.cn

***Abstract*—Given the problem proposed in RTSS 2021 Industry Challenge, we formally define a system model where dependent tasks are organized in a direct acyclic graph structure and communicate with an unblocking manner. For such a system, we present solutions to estimate three time latencies: maximum reaction time, maximum data age, and maximum difference among the timestamps of "source" sensor data. One major contribution made in this paper is that the clock shifts among different processing units have been taken into account during the estimations without a mandatory schedule simulation process. By doing this, fairly tight upper bounds can be calculated for all three latencies.**

***Index Terms*—distributed embedded system, real-time scheduling, heterogenous processing units**

I. Introduction

Based on the problem posted in RTSS 2021 Industry Challenge, we are considering a system with a set of data dependent tasks, where data dependencies among different tasks can be modeled as a direct acyclic graph (DAG) structure. Tasks communicate through buffers. More specifically, they read and write data from/to the buffer in a non-blocking manner. Each read or write is constrained to happen at the start or the complete time of a released task instance respectively.

According to the posted problem, the lengths of three time intervals are required to be analyzed. One is the maximum latency between the occurrence of an event and the time when the system gives the first control output that corresponds to the event. It is frequently referred to as the **"maximum reaction time"** [1]–[3] of a system. Another time interval is the maximum latency between a captured sensor data and the final output directly caused by the sensor data. In some papers, it is called **"maximum data age"** [1], [2]. It is worth noting that if under-sampling exists in any stage of the whole data processing progress, not every sensor data can be finally propagated to the output. The under-sampling could happen when a reader has lower reading frequency then its corresponding writer task. In which case, some of the data written to the buffer will not be read until the overwritten happens. The effect of under-sampling and over-sampling have been discussed in [3]. All "source" sensor data that do affect the control output are characterized with timestamps that correspond to the time when they are captured sensing tasks. The third interval needs calculating is the **time difference between the timestamps of "source" sensor data**.

In recent work [1]–[3], the timing analysis for loosely dependent tasks have already received some attention. In [3], a way to estimate the maximum reaction latency has been proposed for a cause-effect chain with non-blocking communication manner. In [1], [2], both maximum reaction time and maximum data age have been analyzed for a cause-effect chain. A major contribution made in [1] is that the clock shifts between any two different processing units have been considered carefully during the latency estimations. Their approach in [1] has greatly lowered the upper bounds of both maximum reaction time and maximum data age comparing with their previous work [2]. Although they have focused on a chain dependent structure, their work can be easily extended to support more complex structures, e.g. the DAG structure. However, several reasons hinders applying extended versions of [1]–[3] to the proposed problem. In [3] only DBP protocol is considered, which assumes that a reader task can read without waiting for the completion of its writer task. Also, tasks are required to start at the same time in [3]. It is not common to have this constraint satisfied in distributed embedded systems, since there is no easy way for different processing units to keep synchronous. In [2], the scheduling algorithm is limited to a fixed priority preemptive policy, but the proposed problem requires a non-preemptive scheduling policy. In addition, [2] computes the latencies by simulating the schedule, which requires a fixed execution time for each task. This requirement is a drawback when dealing with tasks with variable execution time. The system model in [2] aligns with what has been described in the proposed problem, but it assumes an arbitrary release time for all the jobs. This is not true for tasks that are strictly periodic, and thus causes pessimism in the estimations. All of these approaches have not provided estimation for the time difference between the timestamps of "source" sensor data. It is worth noting that subject to the space limitation, only several most related work have been discussed here. These publications have also shed light on our timing analysis.

In this paper, we present methods to tightly estimate the upper bounds of all three mentioned time latencies for a loosely dependent task set structured as a DAG. We take into

This work was supported by the Fundamental Research Funds for the Central Universities (2572019BH04).

account of the clock shifts among different processing units, and give a general solution for a task system where each task has its own phase. Our solution also avoids using schedule simulations to do the estimations, which allows variations in the length of the execution time for each task.

The remainder of this paper is organized as bellow. In Section II, we formally define the system model. In Section III, we describe how the timing analysis is conducted. In Section IV, we conclude the with a summary.

II. System Model

Based on the problem posted in RTSS 2021 Industry Challenge, we formally describe the system model and necessary notions as follows. We assume a system with multiple independent processing unites. A set of periodic tasks $\Gamma = \{\tau_1, ...\tau_i, ...\tau_n\}$ are scheduled in the system. A job is an instance of a task. The n^{th} job of τ_i is denoted as $J_{i,n}$. The release time, starting time and finishing time of $J_{i,n}$ are denoted as $a_{i,n}$, $s_{i,n}$ and $f_{i,n}$ respectively. Each task is mapped to a processing unit before runtime. On each processing unit, one or multiple tasks are executed. If multiple tasks are mapped to the same processing unit, one of the non-preemptive policy is used to schedule the tasks. Each task τ_i is characterized by the tuple $(C_i^{max}, C_i^{min}, T_i, \phi_i, P_i, ECU_i)$. The worst case execution time (WCET) and best case execution time (BCET) of τ_i are denoted as C_i^{max} and C_i^{min}, i.e. the longest and shortest time for τ_i to finish its execution without any preemption or interrupt on its assigned processing unit. The relationship $\infty > C_i^{max} \geq C_i^{min} > 0$ holds. In the case that $C_i^{max} = C_i^{min}$, τ_i has a fixed execution time on its assigned processing unit. The release period of τ_i is denoted as T_i. The processing unit that τ_i is assigned to is ECU_i. ϕ_i is the phasing of τ_i, meaning the time when τ_i releases its first job. In this article we limit ourselves to only consider the condition that each task is assigned a fixed priority. τ_i's priority is denoted as an integer P_i, where a higher value indicates a higher priority.

The worst case response time (WCRT) of τ_i, i.e. the longest time between arrival and finishing time of any job of τ_i, is denoted as R_i. For any τ_i that is assigned to a dedicated processing unit, it is obvious that $R_i = C_i^{max}$. If multiple tasks share the same processing unit, a method to calculate their WCRTs and verify their schedulability is needed. Under fixed-priority non-preemptive scheduling, one available method supporting tasks with arbitrary deadlines can be found in [4]. In this case, we assume R_i is a known value. B_i is used to denote the maximum blocking time of τ_i, i.e. the maximum time interval from the release to the starting of any job of τ_i. It is obvious that the equation $R_i = B_i + C_i^{max}$ holds. So, the value of B_i can be easily obtained when R_i and C_i^{max} are known. Given this knowledge, we assume B_i is also a known value.

The data dependencies among all the tasks in Γ is declared by a DAG structure. Each vertex of the DAG corresponds to a task. A directed edge between task τ_i to τ_j represents the fact that τ_i produces input data tokens for τ_j. In this paper, such a relationship is also described as $\tau_i \to \tau_j$ in text. Any path in the DAG with at least two tasks with interconnect edges is called a cause-effect chain [1]–[3]. According to the problem statement, one or multiple source vertexes are allowed in the DAG, but there is only one sink vertex. Each source vertex represents a "sensing task" and the sink vertex corresponds to the task that produces the final control output.

III. Timing Analysis

In this section we describe how the time latencies are bounded for the defined task system. Limited by the space, several concepts defined in [1], [2] are used directly without detailed explanation.

A. Maximum Reaction Time

First, we analysis the maximum reaction time of the system, i.e. the latency between the occurrence of an event and the time when the sink task gives the first control output that corresponds to the event. An integer arithmetic theorem that will be used in this section is stated as Theorem III.1.

Theorem III.1. *(Theorem 3.23 in [5]) Let a and b be integers and $gcd(a, b)$ be the greatest common divisor of a and b. The equation $ax + by = c$ has no integral solutions if $gcd(a, b) \nmid c$. If $gcd(a, b) \mid c$, then there are infinitely many integral solutions. Moreover, if $x = x_0$, $y = y_0$ is a particular solution of the equation, then all solutions are given by $x = x_0 + \frac{b}{gcd(a,b)} \cdot n$, $y = y_0 - \frac{a}{gcd(a,b)} \cdot n$, where n is an integer.*

Then, we prove several lemmas which will help the analysis later.

Lemma III.2. *Let $A_{a_x \to a_y}$ denote the lower bound for the distance between the release time of a job of τ_x and the release time of the nearest job of τ_y that is released after it. Then,*

$$A_{a_x \to a_y} = \begin{cases} gcd(T_x, T_y), & \text{if } gcd(T_x, T_y) \mid (\phi_y - \phi_x); \\ (\phi_y - \phi_x) + \lceil \frac{\phi_x - \phi_y}{gcd(T_x, T_y)} \rceil \cdot gcd(T_x, T_y), & \text{otherwise.} \end{cases}$$

Proof. Assume that job $J_{y,n}$ is released after job $J_{x,m}$ where $m, n \in \mathbb{N}^+$. Let d be the distance between release times of $J_{x,m}$ and $J_{y,n}$. Then, we have

$$d = (n \cdot T_y + \phi_y) - (m \cdot T_x + \phi_x)$$
$$\Rightarrow n \cdot T_y - m \cdot T_x = d - (\phi_y - \phi_x)$$

According to Theorem III.1, $d - (\phi_y - \phi_x)$ must be a value that equals $k \cdot gcd(T_x, T_y)$, where $k \in \mathbb{N}$. so

$$d = (\phi_y - \phi_x) + k \cdot gcd(T_x, T_y)$$

Because $d > 0$, if $gcd(T_x, T_y) \nmid (\phi_y - \phi_x)$, we have $k \geq \lceil \frac{\phi_x - \phi_y}{gcd(T_x, T_y)} \rceil$. We can find the smallest value of $d \in \mathbb{N}^+$ by letting $k = \lceil \frac{\phi_x - \phi_y}{gcd(T_x, T_y)} \rceil$. When $gcd(T_x, T_y) \mid (\phi_y - \phi_x)$, $k > \frac{\phi_x - \phi_y}{gcd(T_x, T_y)}$ should hold. Then, the smallest $d > 0$ can be reached by letting $k = \frac{\phi_x - \phi_y}{gcd(T_x, T_y)} + 1$. □

Lemma III.3. *Let $\Omega_{a_y \to s_x}$ denote the lower bound for the distance between the latest starting time of a job of τ_x and* *the release time of the nearest job of τ_y that is released before it. Then,*

$$\Omega_{a_y \to s_x} = \begin{cases} gcd(T_x, T_y), & \text{if } gcd(T_x, T_y) \mid (\phi_x + B_x - \phi_y); \\ (\phi_x + B_x - \phi_y) + \lceil \frac{-\phi_x - B_x + \phi_y}{gcd(T_x, T_y)} \rceil \cdot gcd(T_x, T_y), & \text{otherwise.} \end{cases}$$

Proof. Assume job $J_{y,n}$ is released before the latest starting time of job $J_{x,m}$ where $m, n \in \mathbb{N}^+$. Let d be the distance between the latest starting time of $J_{x,m}$ and the release time of $J_{y,n}$. Because $s_{x,m} \leq a_{x,m} + B_x$, we have

$$d = (m \cdot T_x + \phi_x + B_x) - (n \cdot T_y + \phi_y)$$
$$\Rightarrow m \cdot T_x - n \cdot T_y = d - (\phi_x + B_x - \phi_y)$$

According to Theorem III.1, $d - (\phi_x + B_x - \phi_y)$ must be a value that equals $k \cdot gcd(T_x, T_y)$, where $k \in \mathbb{N}$. so

$$d = (\phi_x + B_x - \phi_y) + k \cdot gcd(T_x, T_y)$$

The following proof is similar as the proof of Lemma III.2. Because $d > 0$, if $gcd(T_x, T_y) \nmid (\phi_x + B_x - \phi_y)$, we have $k \geq \lceil \frac{-\phi_x - B_x + \phi_y}{gcd(T_x, T_y)} \rceil$. We can find the smallest value of $d \in \mathbb{N}^+$ by letting $k = \lceil \frac{-\phi_x - B_x + \phi_y}{gcd(T_x, T_y)} \rceil$. When $gcd(T_x, T_y) \mid (\phi_x + B_x - \phi_y)$, $k > \frac{-\phi_x - B_x + \phi_y}{gcd(T_x, T_y)}$ should hold. Then, the smallest $d > 0$ can be reached by letting $k = \frac{-\phi_x - B_x + \phi_y}{gcd(T_x, T_y)} + 1$. □

Lemma III.4. *Let $\Theta_{a_y \to f_x}$ denote the lower bound for the distance between the latest finishing time of a job of τ_x and the release time of the nearest job of τ_y that is released before it. Then,*

$$\Theta_{a_y \to f_x} = \begin{cases} gcd(T_x, T_y), & \text{if } gcd(T_x, T_y) \mid (\phi_x + R_x - \phi_y); \\ (\phi_x + R_x - \phi_y) + \lceil \frac{-\phi_x - R_x + \phi_y}{gcd(T_x, T_y)} \rceil \cdot gcd(T_x, T_y), & \text{otherwise.} \end{cases}$$

Proof. Assume that job $J_{y,n}$ is released before the latest finishing time of job $J_{x,m}$ where $m, n \in \mathbb{N}^+$. Let d be the distance between the latest finishing time of $J_{x,m}$ and the release time of $J_{y,n}$. Because $f_{x,m} \leq a_{x,m} + R_x$, we have

$$d = (m \cdot T_x + \phi_x + R_x) - (n \cdot T_y + \phi_y)$$
$$\Rightarrow m \cdot T_x - n \cdot T_y = d - (\phi_x + R_x - \phi_y)$$

According to Theorem III.1, $d - (\phi_x + R_x - \phi_y)$ must be a value that equals $k \cdot gcd(T_x, T_y)$, where $k \in \mathbb{N}$. so

$$d = (\phi_x + R_x - \phi_y) + k \cdot gcd(T_x, T_y)$$

The following proof is similar as the proofs of Lemma III.2 and Lemma III.3. Because $d > 0$, if $gcd(T_x, T_y) \nmid (\phi_x + R_x - \phi_y)$, we have $k \geq \lceil \frac{-\phi_x - R_x + \phi_y}{gcd(T_x, T_y)} \rceil$. We can find the smallest value of $d \in \mathbb{N}^+$ by letting $k = \lceil \frac{-\phi_x - R_x + \phi_y}{gcd(T_x, T_y)} \rceil$. When $gcd(T_x, T_y) \mid (\phi_x + R_x - \phi_y)$, $k > \frac{-\phi_x - R_x + \phi_y}{gcd(T_x, T_y)}$ should hold. Then, the smallest $d > 0$ can be reached by letting $k = \frac{-\phi_x - R_x + \phi_y}{gcd(T_x, T_y)} + 1$. □

Now we consider the simplest case that there are only two tasks in a cause-effect chain.

Lemma III.5. *In a cause-effect chain $\tau_i \to \tau_j$, let $J_{i,p}$ and $J_{j,q}$ be the p^{th} and q^{th} job of τ_i and τ_j respectively. If $J_{j,q}$ is the first job of τ_j that executes after the finishing time of $J_{i,p}$. Let $\Delta_{a_i \to a_j}$ denote the upper bound of $a_{j,q} - a_{i,p}$. The following condition holds for any non-preemptive fixed-priority schedule*

$$\Delta_{a_i \to a_j} = \begin{cases} T_j + B_i - \Omega_{a_j \to s_i}, & ECU_i = ECU_j \text{ and } P_j > P_i; \\ T_j - A_{a_j \to a_i}, & ECU_i = ECU_j \text{ and } P_j < P_i; \\ T_j + R_i - \Theta_{a_j \to f_i}, & \text{otherwise, i.e. } ECU_i \neq ECU_j. \end{cases}$$

Proof. When τ_i and τ_j execute on the same ECU, two scenarios need to be considered:

1) $P_j > P_i$, i.e. τ_i has a lower priority than τ_j.
2) $P_j < P_i$, i.e. τ_i has a higher priority than τ_j.

If $P_j > P_i$, there must be $a_{j,q-1} < s_{i,p}$, meaning $J_{j,q-1}$ arrives no later than the starting time of $J_{i,p}$. Otherwise $J_{j,q-1}$ will be blocked until the finishing time of $J_{i,p}$ or arrive after the finishing time of $J_{i,p}$, which contradicts the assumption. Thus $a_{j,q-1} < a_{i,p} + B_i$. According to Lemma III.3, $a_{i,p} + B_i - a_{j,q-1} \geq \Omega_{a_j \to s_i}$. So $a_{j,q} = a_{j,q-1} + T_j \leq a_{i,p} + T_j + B_i - \Omega_{a_j \to s_i}$.

If $P_j < P_i$, there must be $a_{j,q-1} < a_{i,p}$. Otherwise $J_{j,q-1}$ will start its execution after the finishing time of $J_{i,p}$ and be the first job of τ_j that executes after the finishing time of $J_{i,p}$. According to Lemma III.2, $a_{i,p} - a_{j,q-1} \geq A_{a_j \to a_i}$. Thus $a_{j,q} = a_{j,q-1} + T_j \leq a_{i,p} + T_j - A_{a_j \to a_i}$.

When τ_i and τ_j are assigned to different ECUs, there should be $a_{j,q-1} < f_{i,p}$. It follows $a_{j,q-1} < a_{i,p} + R_i$ According to Lemma III.4, $a_{i,p} + R_i - a_{j,q-1} \geq \Theta_{a_j \to f_i}$. Then, we have $a_{j,q} = a_{j,q-1} + T_j \leq a_{i,p} + T_j + R_i - \Theta_{a_j \to f_i}$. □

Let E_j denote a cause-effect chain and K_j denote the number of periodic tasks in E_j. We denote $e_j(i)$ as the function that returns the index of the i^{th} task in E_j. Then, we have $E_j = \tau_{e_j(1)} \to \tau_{e_j(i)} \to ... \to \tau_{e_j(K_j)}$. Additionally, we use $J_{e_j(i), f_j(i)}$ and $J_{e_j(i), b_j(i)}$ to denote a job in the immediate forward and backward job chain respectively. Note that, the definition for immediate forward and backward job chains can be found in [2]. The function $f_j(i)$ and $b_j(i)$ return indexes denoting that $J_{e_j(i), f_j(i)}$ and $J_{e_j(i), b_j(i)}$ are the $f_j(i)^{th}$ and $b_j(i)^{th}$ job of $\tau_{e_j(i)}$. Now we can estimate the maximum reaction time of a cause-effect chain.

Theorem III.6. *The maximum reaction time of a cause-effect chain E_j is upper bounded by*

$$T_{e_j(K_j)} + R_{e_j(K_j)} + \sum_{i=1}^{K_j - 1} \Delta_{a_{e_j(i)} \to a_{e_j(i+1)}}$$

The proof of Theorem III.6 is omitted here, because it is the same as what has been elaborated in Theorem 5.3 and 5.4 in [2].

With Theorem III.6, the reaction time of all the cause-effect chains in a DAG that start from a event sensing task and end to the control output task can be upper bounded. Therefore, the maximum reaction time of the whole DAG task system is bounded.

B. Maximum Data Age

Then, we analyze the age of data for each sensing task in Γ, i.e. the interval between the time when a raw data is captured by a sensing task and when the final output directly caused by the data is produced by the sink task.

Lemma III.7. *In a cause-effect chain $\tau_i \rightarrow \tau_j$, let $J_{i,p}$ and $J_{j,q}$ be the p^{th} and q^{th} job of τ_i and τ_j respectively. If $J_{i,p}$ is the last job of τ_i that executes before the starting time of $J_{j,q}$. Let $\Lambda_{a_i \rightarrow a_j}$ denote the upper bound of $a_{j,q} - a_{i,p}$. Then, for any non-preemptive fixed-priority schedule, the following condition holds:*

$$\Lambda_{a_i \rightarrow a_j} \\ = \begin{cases} T_i + B_i - \Omega_{a_j \rightarrow s_i}, & \textit{ECU}_i = \textit{ECU}_j \textit{ and } P_j > P_i; \\ T_i - A_{a_j \rightarrow a_i}, & \textit{ECU}_i = \textit{ECU}_j \textit{ and } P_j < P_i; \\ T_i + R_i - \Theta_{a_j \rightarrow f_i}, & \textit{otherwise, i.e. } \textit{ECU}_i \neq \textit{ECU}_j. \end{cases}$$

Proof. When τ_i and τ_j are assigned to the same ECU, two scenarios need to be considered:

1) $P_j > P_i$, i.e. τ_i has a lower priority than τ_j.
2) $P_j < P_i$, i.e. τ_i has a higher priority than τ_j.

If $P_j > P_i$, because $J_{i,p+1}$ finishes after the starting time of $J_{j,q}$, it must be $a_{j,q} \leq s_{i,p+1}$. Hence, $a_{j,q} \leq a_{i,p+1} + B_i$. According to Lemma III.3, $a_{i,p+1} + B_i - a_{j,q} \geq \Omega_{a_j \rightarrow s_i}$. So $a_{j,q} \leq a_{i,p+1} + B_i - \Omega_{a_j \rightarrow s_i} = a_{i,p} + T_i + B_i - \Omega_{a_j \rightarrow s_i}$.

Similarly, if $P_j < P_i$, there must be $a_{j,q} < a_{i,p+1}$. Otherwise $J_{j,q}$ will start its execution after the finishing time of $J_{i,p+1}$. According to Lemma III.2, $a_{i,p+1} - a_{j,q} \geq A_{a_j \rightarrow a_i}$. Thus $a_{j,q} \leq a_{i,p+1} - A_{a_j \rightarrow a_i} = a_{i,p} + T_i - A_{a_j \rightarrow a_i}$.

When τ_i and τ_j are assigned to different ECUs, there should be $a_{j,q} < f_{i,p+1}$. It follows $a_{j,q} < a_{i,p+1} + R_i$ According to Lemma III.4, $a_{i,p+1} + R_i - a_{j,q} \geq \Theta_{a_j \rightarrow f_i}$. Then, we have $a_{j,q} \leq a_{i,p+1} + R_i - \Theta_{a_j \rightarrow f_i} = a_{i,p} + T_i + R_i - \Theta_{a_j \rightarrow f_i}$. □

Now the upper bound of the maximum data age for a cause-effect chain can be obtained by Theorem III.8.

Theorem III.8. *The maximum data age of a cause-effect chain E_j is upper bounded by*

$$R_{e_j(K_j)} + \sum_{i=1}^{K_j - 1} \Lambda_{a_{e_j(i)} \rightarrow a_{e_j(i+1)}}$$

The proof for Theorem III.8 is also omitted, because it is the same as what has been elaborated in the proofs of Theorem 5.3 and 5.4 in [2].

With Theorem III.8, the data age of all the cause-effect chains that start from a sensing task and end to the control output task can be upper bounded.

C. Maximum Difference of Data Age

Finally, we analyze the time difference between the timestamps of "source" sensor data.

Lemma III.9. *In a cause-effect chain $\tau_i \rightarrow \tau_j$, let $J_{i,p}$ and $J_{j,q}$ be the p^{th} and q^{th} job of τ_i and τ_j respectively. If $J_{i,p}$ is the last job of τ_i that executes before the starting time of $J_{j,q}$. Then, for any non-preemptive fixed-priority schedule, the following inequality holds:*

$$s_{j,q} - a_{i,p} \\ \geq \begin{cases} \min\{A_{a_i \rightarrow a_j}, a_{i,p} + C_i^{min}\}, & \textit{ECU}_i = \textit{ECU}_j \textit{ and } P_j > P_i; \\ a_{i,p} + C_i^{min}, & \textit{otherwise.} \end{cases}$$

Proof. When τ_i and τ_j are assigned to the same ECU, two scenarios need to be considered:

1) $P_j > P_i$, i.e. τ_i has a lower priority than τ_j.
2) $P_j < P_i$, i.e. τ_i has a higher priority than τ_j.

If $P_j > P_i$, it must be that $a_{j,q} > s_{i,p}$, otherwise τ_j will start and finish before τ_i. Then we have $a_{j,q} > a_{i,p}$. According to Lemma III.2, $a_{j,q} - a_{i,p} \geq A_{a_i \rightarrow a_j}$. Because $s_{j,q} \geq a_{j,q}$, we have $s_{j,q} - a_{i,p} \geq A_{a_i \rightarrow a_j}$. Meanwhile, by definition $s_{j,q} \geq f_{i,p}$ should hold. It follows that $s_{j,q} \geq a_{i,p} + C_i^{min}$. Thus, $s_{j,q} - a_{i,p} \geq \min\{A_{a_i \rightarrow a_j}, a_{i,p} + C_i^{min}\}$

If $P_j < P_i$, $s_{j,q} \geq f_{i,p} \geq a_{i,p} + C_i^{min}$. Similarly, when τ_i and τ_j are assigned to different ECUs, $s_{j,q} \geq a_{i,p} + C_i^{min}$ holds. □

Lemma III.10. *In a cause-effect chain $\tau_i \rightarrow \tau_j$, let $J_{i,p}$ and $J_{j,q}$ be the p^{th} and q^{th} job of τ_i and τ_j respectively. If $J_{i,p}$ is the last job of τ_i that executes before the starting time of $J_{j,q}$. Then, for any non-preemptive fixed-priority schedule, the following inequality holds:*

$$s_{j,q} - s_{i,p} \geq C_i^{min}$$

Proof. The minimum time interval between $s_{j,q}$ and $s_{i,p}$ is obtained when $J_{i,p}$ executes for its BCET (C_i^{min}) and $J_{j,q}$ starts immediately upon the completion of $J_{i,p}$. □

Theorem III.11. *The minimum data age of a cause-effect chain E_j is lower bounded by*

$$\begin{cases} \min\{A_{a_{e_j(1)} \rightarrow a_{e_j(2)}}, C_{e_j(1)}^{min}\} + \sum_{i=2}^{K_j} C_{e_j(i)}^{min}, & \textit{ECU}_{e_j(1)} = \textit{ECU}_{e_j(2)} \\ & \textit{and } P_{e_j(2)} > P_{e_j(1)}; \\ \sum_{i=1}^{K_j} C_{e_j(i)}^{min}, & \textit{otherwise.} \end{cases}$$

Proof. Without lose of generality, we assume the release time of $J_{e_j(1),b_j(1)}$ is 0, i.e. $a_{e_j(1),b_j(1)} = 0$. By definition of a valid immediate backward job chain [2], the length of the chain equals:

$$\begin{aligned} & (s_{e_j(2),b_j(2)} - a_{e_j(1),b_j(1)}) + \sum_{i=2}^{K_j - 1} (s_{e_j(i+1),b_j(i+1)} - s_{e_j(i),b_j(i)}) \\ & + (f_{e_j(K_j),b_j(K_j)} - s_{e_j(K_j),b_j(K_j)}) \\ \geq & (s_{e_j(2),b_j(2)} - a_{e_j(1),b_j(1)}) + \sum_{i=2}^{K_j - 1} (s_{e_j(i+1),b_j(i+1)} - s_{e_j(i),b_j(i)}) \\ & + C_{e_j(K_j)}^{min} \\ \geq & (s_{e_j(2),b_j(2)} - a_{e_j(1),b_j(1)}) + \sum_{i=2}^{K_j} C_{e_j(i)}^{min} \quad \text{(by Lemma III.10)} \\ \geq & \begin{cases} \min\{A_{a_{e_j(1)} \rightarrow a_{e_j(2)}}, C_{e_j(1)}^{min}\} + \sum_{i=2}^{K_j} C_{e_j(i)}^{min}, & \textit{ECU}_{e_j(1)} = \textit{ECU}_{e_j(2)} \\ & \text{and } P_{e_j(2)} > P_{e_j(1)}; \\ \sum_{i=1}^{K_j} C_{e_j(i)}^{min}, & \text{otherwise.} \end{cases} \\ & \text{(by Lemma III.9)} \end{aligned}$$

□

For a task set structured as a DAG, when Theorem III.11 is given, the data age of all the cause-effect chains that start from a sensing task and end to the control output task can be lower bounded.

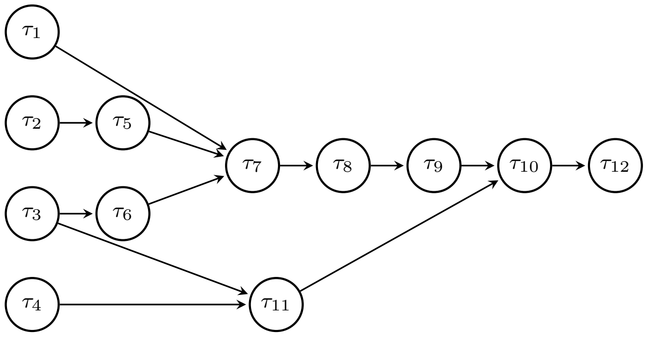


Fig. 1. An example of a DAG task set

Let $\Gamma_r = \{\tau_{r(1)}, ...\tau_{r(i)}, ...\tau_{r(m)}\}$ be the set of sensing tasks, i.e. the source vertexes in the DAG, where $r(i)$ is a function that returns the index of the i^{th} task in Γ_r. Assume that two cause-effect chains E_j and E_l follow the constraint: both of them end with the control output task; $\tau_{e_{j(1)}} = \tau_{r(p)}$, $\tau_{e_{l(1)}} = \tau_{r(q)}$ and $p \neq q$. One safe upper bound of the time difference between the timestamps of two "source" sensor data read in by $\tau_{r(p)}$ and $\tau_{r(q)}$ can be obtained by subtracting minimum data age of E_j (or E_l) from maximum data age of E_l (or E_j). But this approach is pessimistic in a way that it does not take into account the situation that E_j and E_l have crosspoint. In Figure 1, we provide an example task set. The dependent structure in this example is copied from the proposed problem. There are five cause-effect chains that start from a sensing task and end to the control output task: $E_1 = \tau_1 \to \tau_7 \to \tau_8 \to \tau_9 \to \tau_{10} \to \tau_{12}$; $E_2 = \tau_2 \to \tau_5 \to \tau_7 \to \tau_8 \to \tau_9 \to \tau_{10} \to \tau_{12}$; $E_3 = \tau_3 \to \tau_6 \to \tau_7 \to \tau_8 \to \tau_9 \to \tau_{10} \to \tau_{12}$; $E_4 = \tau_3 \to \tau_{11} \to \tau_{10} \to \tau_{12}$; $E_5 = \tau_4 \to \tau_{11} \to \tau_{10} \to \tau_{12}$. Take E_1 and E_2 for instance. These two chains converge from τ_7. Hance, one released job of τ_7 reads in the output of τ_1 and τ_5 simultaneously. From then on, it takes the same time for the source data tokens that read in by τ_1 and τ_2 to reach the output. Hance, the difference of timestamps between the source data tokens read in by τ_1 and τ_2 is determined by the parts of E_1 and E_2 that do not overlap. Given this fact, we propose Algorithm 1 to calculate the maximum time difference among the timestamps of "source" sensor data.

Algorithm 1 The maximum time difference

```
Input: $\Gamma$, $\Gamma_r$
Output: maximum time difference among the timestamps of
    "source" sensor data $DIFF$
 1: $\Gamma_{cross} \leftarrow$ all the tasks in $\Gamma$ that have indegree greater than
    1 as vertexes in the DAG structure
 2: $DIFF \leftarrow 0$, $\Gamma_{temp} \leftarrow \varnothing$
 3: for each task $\tau_m$ in $\Gamma_{cross}$ do
 4:     for each task $\tau_n$ in $\Gamma_r$ do
 5:         if exist a cause-effect chain $E_x$ starts from $\tau_n$ and
            ends to $\tau_m$ then
 6:             $\Gamma_{temp} \leftarrow \Gamma_{temp} \bigcup \{\tau_n\}$
 7:             $MAX_n \leftarrow$ the maximum data age of $E_x$ by
                Theorem III.8
 8:             $MIN_n \leftarrow$ the minimum data age of $E_x$ by Theo-
                rem III.11
 9:         end if
10:     end for
11:     for each task $\tau_n$ in $\Gamma_{temp}$ do
12:         $Y \leftarrow \arg\min_{\tau_m \in \Gamma_{temp} \bigwedge m \neq n} \{MIN_m\}$
13:         if $DIFF < MAX_n - Y$ then
14:             $DIFF \leftarrow MAX_n - Y$
15:         end if
16:     end for
17:     $\Gamma_{temp} \leftarrow \varnothing$
18: end for
```

IV. Summary

In this paper, we give a solution to estimate maximum reaction time, maximum data age, and maximum difference among the timestamps of "source" sensor data for a distributed embedded task system. We consider carefully about the effect that could be caused by clock shifts among different processing units and form a system model that allows each task to have its own first release time. Some other system models with tighter constraints can be seen as special cases under our assumption and adopt our solution directly. Examples are systems that restrict all the tasks to release their first job synchronously and systems that globally asynchronized and locally synchronized. Our solution also avoids using schedule simulations to do the estimations, which allows variations in the length of the execution time for each task. In the future, we would like to test our solution on a real hardware platform.

References

[1] M. Günzel, K.-H. Chen, N. Ueter, G. v. d. Brüggen, M. Dürr, and J.-J. Chen, "Timing analysis of asynchronized distributed cause-effect chains," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021, pp. 40–52.

[2] M. Dürr, G. V. D. Brüggen, K.-H. Chen, and J.-J. Chen, "End-to-end timing analysis of sporadic cause-effect chains in distributed systems," vol. 18, no. 5s, Oct. 2019.

[3] J. Abdullah, G. Dai, and W. Yi, "Worst-case cause-effect reaction latency in systems with non-blocking communication," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 1625–1630.

[4] R. J. Bril, J. J. Lukkien, and W. F. Verhaegh, "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption," *Real-time Systems*, vol. 42, no. 1, pp. 63–119, 2009.

[5] K. H. Rosen, *Elementary number theory*, 2005.

Toward Real-Time Guaranteed Scheduling for Autonomous Driving Systems

Jinghao Sun[1], Tianyi Wang[1], Kailu Duan[1], Bin Lu[2], Jiankang Ren[1], Zhishan Guo[3], Guozhen Tan[1]
1. *Dalian University of Technology*, Dalian, China
2. *Shanghai Glorysoft Co., Ltd.*, Shanghai, China
3. *University of Central Florida*, Orlando, United States

***Abstract*—We propose a SMT method to derive a real-time guaranteed schedule for the autonomous driving system described in the industry challenge. All timing constraints proposed as the industry challenge are considered in our SMT model.**

I. Problem Statement

We focus on an autonomous driving system with the processing graph as shown in Figure 1. Starting from the left side, the sensors, e.g., mmWave radar, LiDAR, Camera, and GNSS/IMU, produce raw data, which are processed by downstream components in the system. The sensors and processing components are all invoked periodically and at different frequencies. For example, the camera captures images at 30 Hz and the "tracking" task are invoked at 10 Hz as shown in Figure 1.

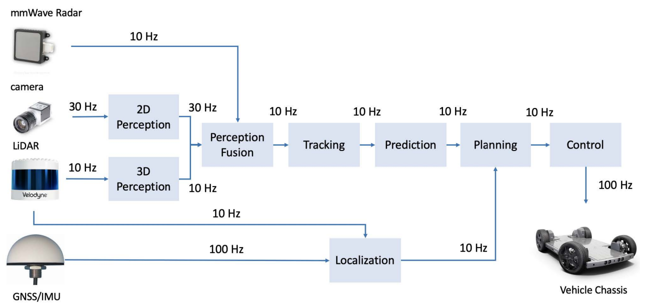


Fig. 1. Processing graph of an autonomous driving system.

Sensing Data. There are two types of sensor data: event sensing data and status sensing data. Status sensing data are used to report the status. Event sensing data are used to detect events. Without loss of generality, we let the data from GNSS/IMU be the status sensing data, and let the data from camera, mmWave radars and LiDAR be the event sensing data. When an event happens, (some of) the corresponding sensors are capable to capturing this event.

Task Processing. There is an unit-size buffer between each pair of adjacent components (e.g., the "tracking" component and the "prediction" component). The upstream ("tracking") component writes data to the buffer via an output port. The downstream ("prediction") component reads data from the buffer via an input port. After a task is ready for execution, it first reads data from one or multiple input ports. When the task finishes, it produces output data to its output ports. The old data in the buffer is over-written by the new data produced by the upstream component. The data writing and reading are applied in a non-blocking manner. We assume that there is no data communication delay between tasks.

Computation Platform. For simplicity, we consider the platform with three computation resources, i.e., a DSP, a GPU and a CPU.[1] The processing procedures of sensor data (from mmWave Radar, camera, LiDAR, and GNSS/IMU) are executed on the DSP. The perception (including "2D perception", "3D perception" and "perception fusion") and localization are deployed on the GPU. Other procedures (including "tracking", "prediction", "planing" and "control") are deployed on the CPU. The scheduling on each computation resource is non-preemptive.

Timing Constraints. We aim to schedule tasks in the autonomous driving system meeting the following three main timing constraints.

- When an event occurs, its related data must be perceived, processed and eventually used to generate control commands within a certain time limit.
- The control command must be performed based on the status information generated sufficiently fresh.
- The difference among the timestamps of the corresponding raw data read by a task from its input ports must be no longer than a pre-defined threshold.

II. System Model

A. Multi-Rate DAG Task

We use a multi-rate directed acyclic graph (DAG) model $G = (V, E)$ to represent an autonomous system, where V is the set of vertices, E is the set of edges between vertices. Each vertex v_i of V represents the task τ_i with the execution time e_i, the period P_i, and the relative deadline D_i. The task τ_i releases jobs periodically. For the sake of convenience, we assume that each task τ_i releases its x-th job $J_{i,x}$ at time $r_{i,x} = (x-1)P_i$, and the absolute deadline of $J_{i,x}$ is $(x-1)P_i + D_i$. For simplicity, we consider that each task τ_i has the implicit deadline , i.e., $D_i = P_i$. A job is *ready* to be executed once it is released, and must be completed before its absolute deadline. Each edge (v_i, v_j) of E indicates that there is a buffer B_{ij} with size 1 between the tasks τ_i and τ_j. When

[1]Our method can also be extended to the computation platform with multiple DSPs, GPUs, and multi-core CPUs.

a job $J_{j,x}$ of τ_j is ready to executed, it first reads from the buffer B_{ij} which are associated with the edge (v_i, v_j) ending at v_j. When the job $J_{j,x}$ finishes, it writes to the buffer B_{ji} which are associated with the edge (v_j, v_i) beginning with v_j. The reading and writing time are both assumed to be 0. The vertex with no incoming edges is called *source* vertex. The vertex with no outgoing edges is called *sink* vertex. There may be multiple source vertices and sink vertices in G.

Example 1. *The corresponding multi-rate DAG G of the autonomous driving system in Figure 1 is shown in Figure 2. The DAG G has 12 vertices and 11 edges. Each vertex is labeled with its period. There are three types of periods, i.e., 10 ms, 33 ms and 100 ms. The buffer between two vertices is labeled on their associated edge. There are four source vertices, i.e., v_0, v_1, v_2 and v_3, and a single sink vertex v_{11}. The task of each vertex is also listed in the above of Figure 2.*

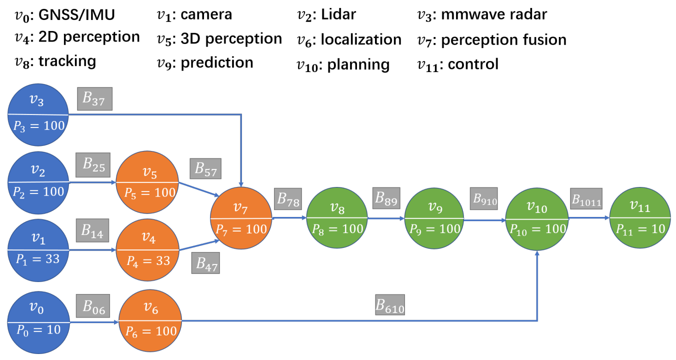


Fig. 2. Multi-rate DAG G of the autonomous driving system given in Figure 1.

B. Scheduling Model

We schedule the tasks of G on a heterogeneous platform including three types of processors: DSP, GPU and CPU. We divide the vertex set V into three disjoint subsets: $V = V_1 \cup V_2 \cup V_3$. The vertices in V_1 can only be executed on the DSP. The vertices in V_2 can only be executed on the GPU. The vertices in V_3 can only be executed on the CPU. For example, as shown in Figure 2, the vertices marked blue are contained in V_1, i.e., $V_1 = \{v_0, v_1, v_2, v_3\}$. The vertices marked orange are contained in V_2, i.e., $V_2 = \{v_4, v_5, v_6, v_7\}$. The vertices marked green are contained in V_3, i.e., $V_3 = \{v_8, v_9, v_{10}, v_{11}\}$. The scheduling strategy on each processor is non-preemptive.

Scheduling Constraints

In the following, we formally describe the timing constraints that a feasible schedule of G should satisfy. Before going into details, we first introduce some useful notations.

Definition 1 (Path). *A path π of G is a sequence of vertices $\pi = (v_1, v_2, \cdots, v_k)$ such that each pair of adjacent vertices v_i and v_{i+1} corresponds to an edge (v_i, v_{i+1}) of G.*

For example, as shown in Figure 2, $\pi = \{v_1, v_4, v_7, v_8, v_9, v_{10}, v_{11}\}$ is a path of G. For any two vertices v_i and v_j of G, we say v_j is *reachable* from v_i if there is a path from v_i to v_j. For any vertex v_i of G, we use $R_{src}(v_i)$ to denote the set of source vertices that reach v_i. For example, as shown in Figure 2, $R_{src}(v_8) = \{v_1, v_2, v_3\}$. As we know that each source vertex of G corresponds to a sensor, and each job $J_{i,x}$ of v_i should deal with the data flows coming from the source vertices in $R_{src}(v_i)$. For simplicity, we give the following assumption.

Assumption 1. *For any two vertices v_i and v_j of G, $R_{src}(v_i) \cap R_{src}(v_j) = \emptyset$ if v_i and v_j can be executed in parallel.*

Clearly, the DAG model of the autonomous driving system as shown in Figure 2 satisfies the above assumption since it has a tree structure.

Definition 2 (Cause-effect chain). *The cause-effect chain c that corresponds to a path π is a sequence of jobs such that*

- *The i-th $J_{i,x}$ job of chain c is released by the task of the i-th vertex v_i of π.*
- *For any two adjacent jobs $J_{i,x}$ and $J_{i+1,y}$, the job $J_{i,x}$ has the maximum finishing time among all jobs that are released by τ_i and finish before the starting of Job $J_{i+1,y}$.*

The latency of c is the difference between the finishing time of the last job of c and the release time of the first job of c.

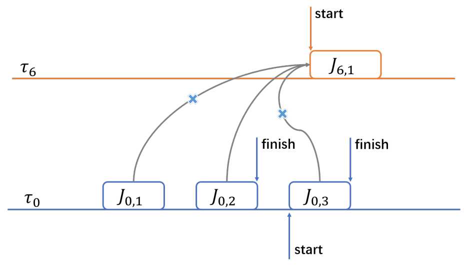


Fig. 3. Illustration for cause-effect chain

For example, we consider a path $\pi = \{v_0, v_6\}$, and one of its possible cause-effect chains $c = \{J_{0,2}, J_{6,1}\}$ is shown in Figure 3. It should emphasize that the job sequence $(J_{0,1}, J_{6,1})$ is not a cause-effect chain since $J_{0,1}$ is not the last job that finishes before $J_{6,1}$. Moreover, job sequence $(J_{0,3}, J_{6,1})$ is not a cause-effect chain since $J_{0,3}$ is not finished before the start of $J_{6,1}$.

We distinguish two types of source vertices: *event* source vertex and *status* source vertex. As shown in Figure 2, there is only one status source vertex v_0 and three event source vertices v_1, v_2 and v_3. We say a path π is a *complete path* of G if π starts with an event source vertex and ends at a sink vertex. For example, three *complete paths* in Figure 2 are $\pi_1 = \{v_1, v_4, v_7, v_8, v_9, v_{10}, v_{11}\}$, $\pi_2 = \{v_2, v_5, v_7, v_8, v_9, v_{10}, v_{11}\}$, and $\pi_3 = \{v_3, v_7, v_8, v_9, v_{10}, v_{11}\}$.

Definition 3 (Reactive time). *The reactive time of G is the maximum latency among all the cause-effect chains corresponding to the complete paths of G.*

Definition 4 (Time stamp). *For any job $J_{i,x}$, its time stamp is a set of integers $\{s^j_{i,x} | v_j \in R_{src}(v_i)\}$, where $s^j_{i,x}$ is a release* *time of a job $J_{j,y}$ released by the source vertex v_j such that there is a cause-effect chain from $J_{j,y}$ to $J_{i,x}$.*

Figure 4 gives a (part of) possible schedule of the jobs in G. The time stamp of job $J_{7,1}$ is $\{s^1_{7,1}, s^2_{7,1}, s^3_{7,1}\}$. The time stamp $s^1_{7,1}$ equals to the release time of $J_{1,2}$ as there is a cause-effect chain from $J_{1,2}$ to $J_{7,1}$, i.e., $(J_{1,2}, J_{4,2}, J_{7,1})$ according to the above definition. Similarly, time stamp $s^2_{7,1}$ and $s^3_{7,1}$ equal to the release time of $J_{2,1}$ and $J_{3,1}$, respectively.

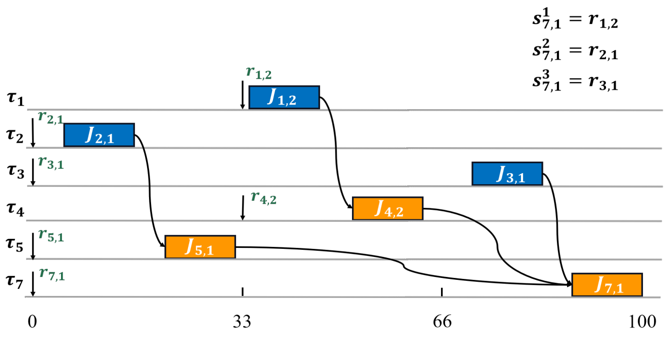


Fig. 4. Illustration for time stamp

According to Assumption 1, for any job $J_{i,x}$ and for any source vertex v_j, there is only one cause-effect chain from a job of v_j to $J_{i,x}$. According to Definition 4, each time stamp $s^j_{i,x}$ of $J_{i,x}$ has an unique value, which can be calculated as follows.

- If v_i is a source vertex, then
$$s^j_{i,x} = r_{i,x} \tag{1}$$
- Otherwise, there is a cause-effect chain $(J_{j,z}, \cdots, J_{l,y}, J_{i,x})$. The time stamp $s^j_{i,x}$ is calculated as
$$s^j_{i,x} = s^j_{l,y} \tag{2}$$

Based on the above concepts, the timing constraints of a feasible schedule is formulated as follows.

- The reactive time of G is bounded by Δ.
- For simplicity, we consider the multi-rate DAG as shown in Figure 2, there is a single status sensing task τ_1 and a single control task τ_{11}. These two tasks have the same period. In each period, the control job released by τ_{11} should start after the completion of job released by task τ_1.
- For each job $J_{i,x}$, and for any two time stamps $s^j_{i,x}$ and $s^l_{i,x}$ of $J_{i,x}$, the difference between $s^j_{i,x}$ and $s^l_{i,x}$ must be less than Δ', i.e., $|s^j_{i,x} - s^l_{i,x}| \le \Delta'$.

III. Real-Time Guaranteed Scheduling

In this section, we aim to derive a static periodic schedule of G that satisfies all timing constraints described in the above section. To this end, we first transform a multi-rate DAG G into its equivalent single-rate DAG G'. Then we solve a feasible schedule of G by developing a SMT model for the scheduling problem defined on G'.

A. Single-Rate DAG Transformation

The transformation from a multi-rate DAG G to its corresponding single-rate DAG G' is applied in Algorithm 1. Similar transformation process was also proposed in [1].

Algorithm 1: Transformation from G to G'.

```
1  generate a virtual source vertex $v_{src}$ with execution time 0
2  generate a virtual sink vertex $v_{snk}$ with execution time 0
3  for each vertex $v_i$ of $G$ do
4      let $k := \frac{HP}{P_i}$
5      generate $k$ job vertices $v_{i,1}, \cdots, v_{i,k}$, each with
         execution time $e_i$ and relative deadline $D_i$
6      if $k = 1$ then
7          add an edge from $v_{src}$ to the first job vertex $v_{i,1}$
8          add an edge from the last job vertex $v_{i,k}$ to $v_{snk}$
9      else
10         generate $k$ dummy vertices $\delta_{i,1}, \cdots, \delta_{i,k}$
11         generate $k-1$ synchronous vertices $\sigma_{i,1}, \cdots, \sigma_{i,k-1}$
12         let $\sigma_{i,0} = v_{src}$ and $\delta_{i,k} = v_{snk}$
13         for each $j = 1, \cdots, k$ do
14             add an edge from $v_{i,j}$ to $\sigma_{i,j}$
15             add an edge from $\delta_{i,j}$ to $\sigma_{i,j}$
16             add an edge from $\sigma_{i,j-1}$ to $v_{i,j}$
17             add an edge from $\sigma_{i,j-1}$ to $\delta_{i,j}$
```

We first denote the hyper period of G as the least common multiple of period of tasks involved in G, i.e., $HP = lcm_{\forall v_i \in V}\{P_i\}$. For example, the hyper period of G is $HP = lcm\{10 \text{ ms}, 33 \text{ ms}, 100 \text{ ms}\} = 100$ ms. Algorithm 1 aims to obtain a transformed DAG G' with a single period HP, and thus, all jobs of the original DAG G that are released during a hyper period HP are revealed in the single-rate DAG G'. As shown in Lines 1 and 2, we generate a virtual source vertex v_{src} and a virtual sink vertex v_{snk}. For each vertex v_i of G, we generate a sequence of job vertices $(v_{i,1}, \cdots, v_{i,k})$, where $k = \frac{HP}{P_i}$ is the number of jobs released by τ_i during a single hyper period HP (see Lines 4 an 5). We consider the following two cases. If there is only a single job of τ_i released in a hyper period, we connect the virtual source vertex v_{src} to the first job vertex $v_{i,1}$, and connect the last job vertex $v_{i,k}$ to the virtual sink vertex v_{snk}. Otherwise, there are multiple jobs of τ_i released during a hyper period HP, i.e., $k > 1$. For each job vertex $v_{i,x}$, we add two auxiliary vertices: a dummy vertex $\delta_{i,x}$ with execution time e_i and a synchronous vertex $\sigma_{i,x}$ with zero execution time. For the sake of convenience, we let $\sigma_{i,0} = v_{src}$ and $\sigma_{i,k} = v_{snk}$. The job vertex $v_{i,x}$ and dummy vertex $\delta_{i,x}$ both join to $\sigma_{i,x}$ for synchronization, and meanwhile, $v_{i,x}$ and $\delta_{i,x}$ are both forked from the vertex $\sigma_{i,x-1}$.

Figure 5 gives the equivalent single-rate DAG G' of G that is generated by Algorithm 1. All dummy vertices are labeled with their execution time. The execution of G' starts from v_{src} and ends at v_{snk}. A vertex in G' is eligible to execute only if all its predecessors are finished. For example, the job vertex $v_{i,x}$ can only start when the synchronous vertex $\sigma_{i,x-1}$ is finished. It indicates that $v_{i,x}$ releases its job at time $(x-1)P_i$. Moreover, the job of $v_{i,x}$ must finish before xP_i due to

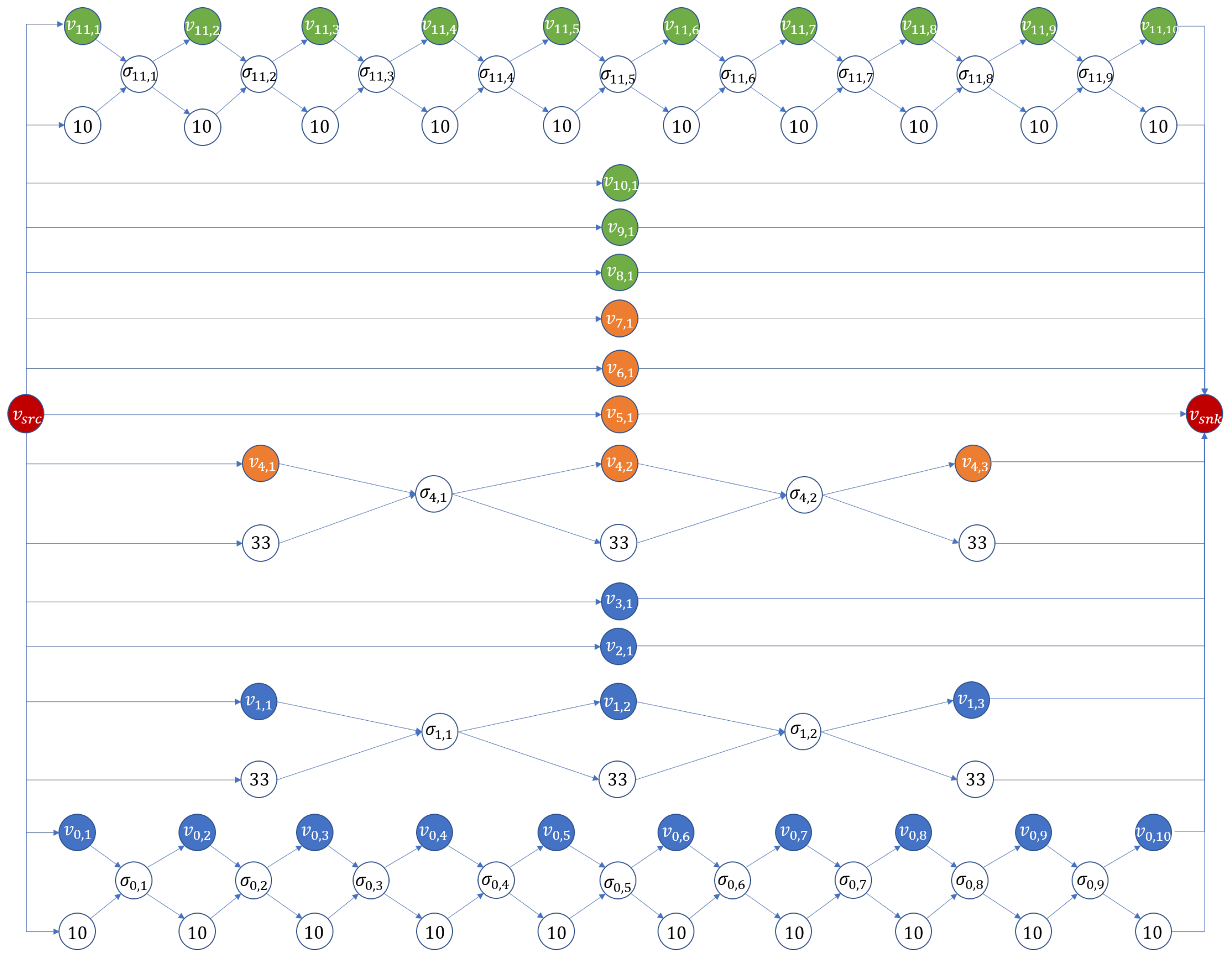


Fig. 5. Single-rate DAG G' transformed from G in Figure 2.

its deadline requirement. The job of $v_{i,x}$ is executed during its own period.

B. SMT-based Schedule Generation

We aim to derive a schedule of the single-rate DAG G' in Figure 5 (during a hyper period HP). Such a schedule is static and is periodically repeated each hyper period. The scheduling problem of G' is formulated as follows. The constants and variables are listed in Table I and Table II, respectively.

TABLE I
CONSTANTS INVOLVED IN THE SMT FORMULATION

constant	description
e_i	the execution time of each job vertex $v_{i,x}$
$r_{i,x}$	the release time of job $J_{i,x}$, i.e., $r_{i,x} = (x-1)P_i$
$d_{i,x}$	the absolute deadline of job $J_{i,x}$, i.e., $d_{i,x} = xP_i$
V_1	the set of vertices that are executed on DSP
V_2	the set of vertices that are executed on GPU
V_3	the set of vertices that are executed on CPU

TABLE II
VARIABLES INVOLVED IN THE SMT FORMULATION

variable	type	description
$b_{i,x}$	integer	the beginning time of the job vertex $v_{i,x}$
$f_{i,x}$	integer	the finishing time of the job vertex $v_{i,x}$
$s^j_{i,x}$	integer	the value of time stamp of job $J_{i,x}$ related to sensor v_j

Constraints.

Non-preemptive. The execution of each job $J_{i,x}$ is non-preemptive, i.e.,

$$f_{i,x} = b_{i,x} + e_i \tag{3}$$

Implicit deadline. Each job $J_{i,x}$ must start after its release time $r_{i,x}$ and finish before its absolute deadline $d_{i,x}$, i.e.,

$$b_{i,x} \geq r_{i,x} \wedge f_{i,x} \leq d_{i,x} \tag{4}$$

Exclusive execution. Any two jobs that need to be executed on the same computation resource should be executed sequentially, i.e., for any vertex subset V_k $(k = 1, 2, 3)$ as defined in

Section II-B and for any two vertices v_i and v_j of V_k, the jobs $J_{i,x}$ and $J_{j,y}$ separately released by τ_i and τ_j must satisfy the following constraint:

$$b_{i,x} \geq f_{j,y} \vee f_{i,x} \leq b_{j,y} \tag{5}$$

Time stamp computation. For any source vertex v_i of G ($i = 0, \cdots, 3$), and for any job vertex $J_{i,x}$ in G', according to (1), there is a single time stamp $s^i_{i,x}$ of $J_{i,x}$, which is calculated as follows.

$$s^i_{i,x} = r_{i,x} \tag{6}$$

For any task τ_i, we let $J_{i,0}$ be the last job of τ_i released in the previous hyper period. For each vertex $v_j \in R_{src}(v_i)$, the time stamp $s^j_{i,0}$ of $J_{i,0}$ is calculated as follows. We let $k = \frac{HP}{P_i}$,

$$s^j_{i,0} = s^j_{i,k} - HP \tag{7}$$

Figure 6 illustrates the above formula.

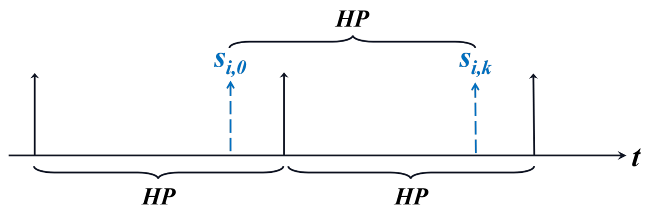


Fig. 6. The difference between the time stamps of $J_{i,0}$ and $J_{i,k}$ is HP

For any edge (v_i, v_j) of G, and for any two job vertices $J_{i,x}$ and $J_{j,y}$ (here $x \geq 0$ and $y \geq 1$), if $J_{i,x}$ is the last job finished before $J_{j,y}$ starts, the time stamp of $J_{j,y}$ equals to that of $J_{i,x}$, i.e., for each $v_l \in R_{src}(v_i)$,

$$\bigwedge_{z>x} (f_{i,z} \geq b_{j,y}) \wedge (f_{i,x} < b_{j,y}) \wedge (s^l_{i,x} = s^l_{j,y}) \tag{8}$$

The first item and the second item ensure that $J_{i,x}$ is the last job finished before the start of $J_{j,y}$, and according to Definition 2, $(J_{i,x}, J_{j,y})$ is a cause-effect chain. According to Definition 4 and by (2), the time stamp of $J^l_{j,y}$ is calculated as the third item of (8).

In the following, we formulate three timing constrains as described in Section II-B.

Reaction time. The reaction time of G is less than a predefined value Δ.

$$f_{11,x} - s^i_{11,x} \leq \Delta \quad v_i \in R_{src}(v_{11}) \tag{9}$$

Data freshness. The status sensing data released from GNSS/IMU sensor must sufficiently fresh when it is used by control modular. Noting that the GNSS/IMU task τ_0 and the control task τ_{11} have the same period, i.e., $P_0 = 10$ms and $P_{11} = 10$ms, we enforce that for each $x = 1, \cdots, k$, the job $J_{0,x}$ of τ_0 must be finished before the job $J_{11,x}$ of τ_{11} starts, i.e.,

$$b_{11,x} > f_{0,x} \tag{10}$$

Difference of time stamps. For each job $J_{i,x}$, the difference of any two of its timestamps is upper-bounded by a pre-defined value Δ', i.e.,

$$|s^j_{i,x} - s^l_{i,x}| \leq \Delta' \quad \forall v_j, v_l \in R_{src}(v_i) \tag{11}$$

By using the above SMT formulation, we can compute a feasible schedule of G', which can be periodically used as a static schedule of G for each hyper period. The SMT model for the autonomous driving system in Figure 1 has 56 variables and 388 constraints.

References

[1] Micaela Verucchi, Mirco Theile, Marco Caccamo, and Marko Bertogna. Latency-aware generation of single-rate dags from multi-rate task sets. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 226–238. IEEE, 2020.

A linear programming solution to real-time scheduling in autonomous driving system

Chu-ge Wu, Automatic School, Beijing Institute of Technology, wucg@bit.edu.cn;
Shaopeng Yin, SenseTime, yinshaopeng@senseauto.com.

Considering the instance of autonomous driving system mentioned in RTSS 2021 Industry Challenge, a mixed integer linear programming (MILP) model is built to formulate this specific instance. Based on the specific instance, system and scheduling models are proposed correspondingly.

System and application model:

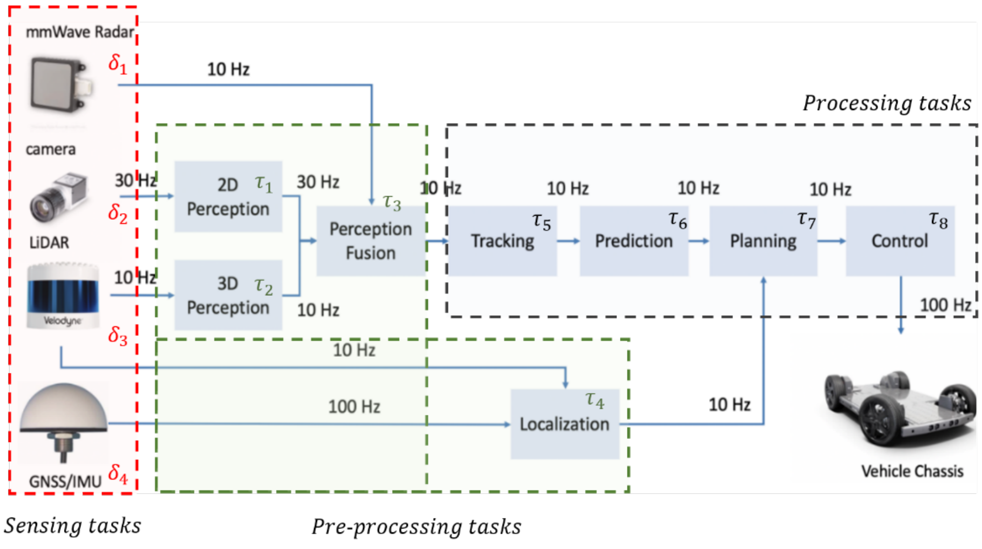
The processing units are modelled as a system consists of m heterogeneous processors (e.g., CPU, GPU or DSP). The data communication delay between different processors is zero.

Row data produced by sensors including “mmWave Radar”, “camera”, “LiDAR” and “GNSS/IMU”, are released periodically. A set of periodic **sensing tasks** $\delta_{i,j} = (ps_i, rs_{i,j})$ is used to model the source data, where $\delta_{i,j}$ represents its j-th job of task δ_i with its period ps_i and release time $rs_{i,j}$. It is assumed that the release time is known in advance as (1):

$$rs_{i,j+1} = rs_{i,j} + ps_i, \qquad j = 0,1,2 \dots \tag{1}$$

For the processing components, $\tau_{l,k} = (p_l, \boldsymbol{w_{l,k}})$ is used to model these processing tasks, where $\tau_{l,k}$ represents the k-th job of task τ_l and $\boldsymbol{w_{l,k}}$ is an m-dimension vector consisting of the worst-case execution time (WCET) on the corresponding processors. We divide the tasks into “**pre-processing tasks**” and “**processing tasks**”, where pre-processing tasks refer to the tasks reading data directly, such as “2D Perception”, “Perception Fusion” and “Location” and processing tasks refer to the other ones.

In addition, the **hybrid period (HP)** is calculated by the least common multiple of all the periods of sensing and processing tasks.



Scheduling model

Based on the application and system models, some decision variables are defined and added to build a MILP model to formulate the timing constraints and the whole procedure. In this work, we consider all the sensing tasks released in **one HP**. All of the pre-processing and processing tasks read and use the sensing data are scheduled while the tasks might be processed after the first HP ends because of the lack of computing capability.

Table 1. Notations for the real-time DAG scheduling problem

Notation		Implication
Instance Data	n	number of tasks in the DAG;
	m	number of processors;
	$w_{l,k}(i)$	WCET of the k-th job of task l on processor i;
	ps_i	period of sensing task i;
	p_l	period of processing task l;
	HP	hyper period of the DAG;
Others	M	a very large constant number;
	$t_1 \rightarrow t_2$	task t_1 is precedent of task t_2, i.e., a direct edge exists between node t_1 and t_2 in DAG.
Decision variables	$x_{l,k,i,r}$	$x_{l,k,i,r} \in \{0,1\}$, $x_{l,k,i,r} = 1$ denotes that the k-th job of task l is the r-th job processed on processor i during the hybrid period, otherwise, $x_{l,k,i,r} = 0$.
	$s_{l,k}$	$s_{l,k} \geq 0$, the starting time of the k-th job of task l;
	$c_{l,k}$	$c_{l,k} \geq 0$, the completion time of the k-th job of task l.
	$qs(i, l)$	$qs(i,l) \in \boldsymbol{N}$, the order of sensing job of task i used by pre-processing task l;
	$q(l, k)$	$q(i,l) \in \boldsymbol{N}$, the order of job of task l used by processing task k;
	$ls(i, l)$	$ls(i,l) \in \boldsymbol{N}$, the number of HP between sensing task i and its pre-processing task l;
	l_{lk}	$l_{lk} \in \boldsymbol{N}$, the number of HP between task l and k where task k uses the data of task l.

Locate the source data token for each task:

As the buffer is over-written when a new token is produced, only the latest released data can be received by its following task. For each pair of tasks in DAG where exists a path between them, its real data transmission path needs to be determined and the transmission time can be calculated with $q(i, l)$ and l_{lk} defined in Table 1. The DAG path and real data transmission path are illustrated in Figure. 2.

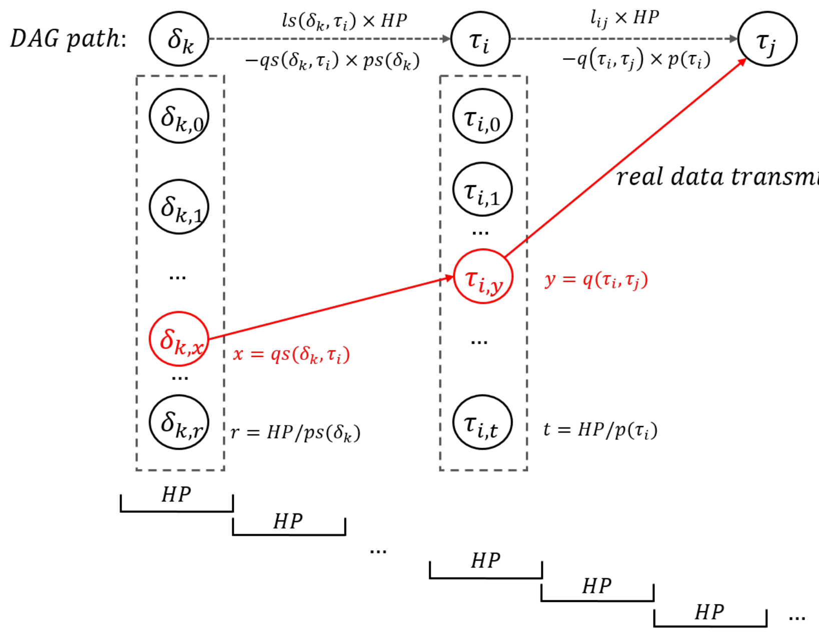


Figure. 2 A simple example of DAG path and its corresponding data transmission path

It can be seen from Figure 2, for sensing task δ_k, $r = HP/ps(\delta_k)$ jobs are released during the HP. For its pre-processing task τ_i, to determine which job is received (x in Figure 2), the relationship between the starting time of pre-processing task and the release time of sensing data is considered.

It is obviously that if we have:

$$rs_{k,x} + ls(k,i) \times HP \leq s_{i,y} \leq rs_{k,x+1} + ls(k,i) \times HP \tag{2}$$

Then, based on (1), we have (3)-(6):

$$rs_{i,0} + qs(k,i) \cdot ps_i + ls(k,i) \times HP \leq s_{l,t} \tag{3}$$

$$s_{l,t} \leq rs_{i,0} + (qs(k,i)+1) \cdot ps_i + ls(k,i) \times HP, \;\; y = 0,1,\dots,\frac{HP}{p_i} - 1 \tag{4}$$

$$0 \leq qs(k,i) \leq \frac{p_i}{ps_k} - 1 \tag{5}$$

$$ls(k,i) \in \boldsymbol{N} \tag{6}$$

In this way, the data transmission delay (d_{ki}) between sensing task δ_k and its pre-processing task τ_i can be calculated as:

$$ds_{ki} = ls(k,i) \times HP - qs(k,i) \times ps_k \tag{7}$$

Similarly, considering the processing task τ_i and τ_j with different frequency, it takes some efforts to determine which job of τ_i is received by task τ_j (y in Figure 2) as the starting and completion time of pre-processing tasks are not determined in the heterogeneous processing system. Take task pair (τ_1, τ_3) in Figure. 1 as an example, where $c'_{1,0}$ represents the completion time of $\tau_{1,0}$ in the next hybrid period:

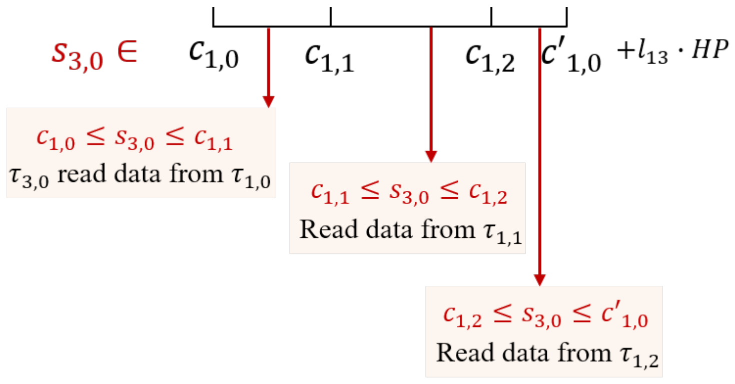

$s_{3,0} \in$
$c_{1,0}$
$c_{1,1}$
$c_{1,2}$
$c'_{1,0}$
$+l_{13} \cdot HP$
$c_{1,0} \leq s_{3,0} \leq c_{1,1}$
$\tau_{3,0}$ read data from $\tau_{1,0}$
$c_{1,1} \leq s_{3,0} \leq c_{1,2}$
Read data from $\tau_{1,1}$
$c_{1,2} \leq s_{3,0} \leq c'_{1,0}$
Read data from $\tau_{1,2}$

Figure. 3 An illustration of how to locate the source data

To determine which interval starting time of the following job belongs to, some inter-variables (a_k and z_l) are introduced into our model:

$$s_{3,0} = a_0 \cdot c_{1,0} + a_1 \cdot c_{1,1} + a_2 \cdot c_{1,2} + a_3 \cdot c'_{1,0} + l_{13} \cdot HP \tag{8}$$

$$0 \leq a_0 \leq z_0 \tag{9-1}$$

$$0 \leq a_1 \leq z_0 + z_1 \tag{9-2}$$

$$0 \leq a_2 \leq z_1 + z_2 \tag{9-3}$$

$$0 \leq a_3 \leq z_2 \tag{9-4}$$

$$z_0 + z_1 + z_2 = 1, \qquad z_0, z_1, z_3 \in \{0,1\} \tag{10}$$

$$q(1,3) = z_1 + 2 \cdot z_2 \tag{11}$$

In this way, $q(1,3)$ can be deduced by the starting time and completion time of its precedent jobs. Thus, the data transmission delay (d_2) between task pair (τ_i, τ_j) directly connect with an edge can be calculated as:

$$d_{ij} = l_{ij} \times HP - q(k,i) \times p_k \tag{12}$$

Correlation constraints

time-skew between its input source sensing tasks should be limited, i.e. $\boldsymbol{C(Y|X_1, X_2, \dots, X_i) = \varepsilon_c}$ where ε_c denotes the pre-defined upper bound.

For this specific DAG, we have:

$$C(\tau_3|\delta_1, \delta_2, \delta_3) = \varepsilon_c \tag{13}$$

$$C(\tau_4|\delta_3, \delta_4) = \varepsilon_c \tag{14}$$

$$C(\tau_7|\delta_1, \delta_2, \delta_3, \delta_4) = \varepsilon_c \tag{15}$$

Specifically, considering $C(\tau_3|\delta_1, \delta_2, \delta_3) = \varepsilon_c$, we have:

$$C(\tau_3|\delta_1, \delta_2) = \varepsilon_c \rightarrow |ds_{13} - ds_{23}| \le \varepsilon_c$$

$$(ls(1,3) \times HP - qs(1,3) \times ps_3) - (ls(2,3) \times HP - qs(2,3) \times ps_3) \le \varepsilon_c$$

$$(ls(2,3) \times HP - qs(2,3) \times ps_3) - (ls(1,3) \times HP - qs(1,3) \times ps_3) \le \varepsilon_c$$

$$C(\tau_3|\delta_1, \delta_3) = \varepsilon_c \rightarrow |ds_{13} - ds_{33}| \le \varepsilon_c$$

$$(ls(1,3) \times HP - qs(1,3) \times ps_3) - (ls(3,3) \times HP - qs(3,3) \times ps_3) \le \varepsilon_c$$

$$(ls(3,3) \times HP - qs(3,3) \times ps_3) - (ls(1,3) \times HP - qs(1,3) \times ps_3) \le \varepsilon_c$$

$$C(\tau_3|\delta_2, \delta_3) = \varepsilon_c \rightarrow |ds_{23} - ds_{33}| \le \varepsilon_c$$

$$(ls(2,3) \times HP - qs(2,3) \times ps_3) - (ls(3,3) \times HP - qs(3,3) \times ps_3) \le \varepsilon_c$$

$$(ls(3,3) \times HP - qs(3,3) \times ps_3) - (ls(2,3) \times HP - qs(2,3) \times ps_3) \le \varepsilon_c$$

In this way, for this instance, $A_3^2 + A_2^2 + A_4^2 = 6 + 2 + 12 = 20$ constraints are introduced to our model. In addition, the correction constraints of τ_5 and τ_6 are guaranteed by τ_3. Similarly, the time-skew between source data of τ_8 is constrained by the constraints of τ_7.

Freshness constraints

To meet the first timing constraint in the RTSS challenge: "Let *a* be a sensor data token and *b* be the final data output indirectly caused by *a*. If *b* is produced at *t*, then *a* must be produced no earlier than a pre-defined value before *t*", i.e., the freshness constraints mentioned in [1].

For this specific DAG, we have:

$$F(Y|\delta_i) = \varepsilon_f, \forall i. \tag{16}$$

where ε_f denotes the pre-defined upper bound.

Specifically, considering $F(Y|\delta_3) = \varepsilon_f$, we have:

$$|ds_{34} + d_{47} + d_{78}| \le \varepsilon_f$$

$$|ds_{32} + d_{23} + d_{35} + d_{56} + d_{67} + d_{78}| \le \varepsilon_f$$

where the delay can be calculated according to (7) and (12).

Scheduling constraints

In spite of the data transmission path, feasible schedule in the heterogeneous system should be guaranteed. Figure 4 presents a gantt paint for the instance in Figure 2. It can be seen that:

$$s_{i,0} = rs_{k,0};\ s_{j,0} \ge c_{i,0}, s_{j,0} < c_{i,1}: \delta_{k.0} \rightarrow \tau_{i,0} \rightarrow \tau_{j,0}$$

$$rs_{k,x-1} < s_{i,3} < rs_{k,x};\ s_{j,1} \ge c_{i,3}, s_{j,1} < c_{i,4}: \delta_{k.x-1} \rightarrow \tau_{i,3} \rightarrow \tau_{j,1}.$$

It is obvious that the data transmission path can be deduced according to the starting time and completion time of the corresponding jobs. The values of starting and completion time can be calculated through the inherent constraints mentioned next.

For the jobs in task pair linked with edge in the DAG, constraints as (6) are able to guarantee the precedence constraint because of $a_i \ge 0, \forall i$. On the other hand, there exist precedence constraints between the jobs of one task. Besides, the processor is able to compute at most one job during a certain time period while the tasks are non-preemptive. Based on our previous work [2], the assumptions are modelled as follows:

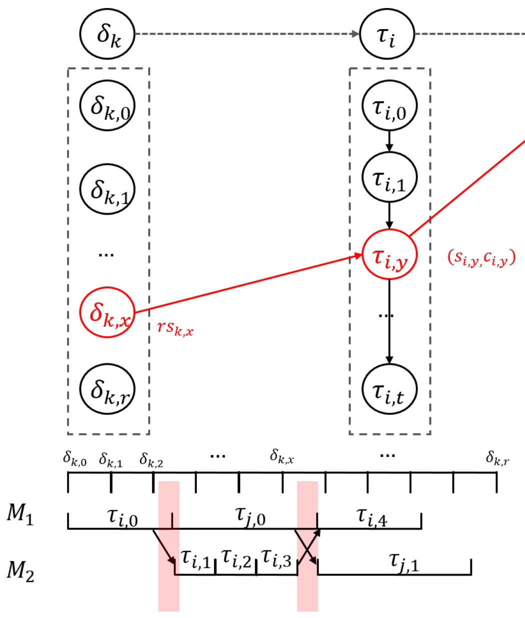


Figure. 4 A simple example of DAG path and its corresponding gantt paint on two processors

$$\sum_{r=1}^{n}\sum_{i=1}^{m} x_{l,k,i,r} = 1, \forall l, k \tag{17}$$

$$\sum_{l=1}^{n}\sum_{k} x_{l,k,i,r} \leq 1, \forall i, r \tag{18}$$

$$\sum_{l_1=1}^{n}\sum_{k} x_{l_1,k,i,r-1} \geq \sum_{l_2=1}^{n}\sum_{k} x_{l_2,k,i,r}, \forall r \geq 1, i \tag{19}$$

$$s_{l_1,k_1} - c_{l_2,k_2} + M\cdot\left(2 - x_{l_1,k_1,i,r} - x_{l_2,k_2,i,r-1}\right) \geq 0, \forall l_1, l_2, k_1, k_2, r \geq 1, i \tag{20}$$

$$c_{l,k} \geq s_{l,k} + \sum_i w_{l,k}(i)\cdot\sum_r x_{l,k,i,r}, \forall l, k \tag{21}$$

$$s_{l,k} \geq c_{l,k-1}, \forall l, k \geq 1 \tag{22}$$

where (17) ensures that each job can be processed once and only once; (18) ensures that for a given position of a certain processor, no more than one job can be processed in case of avoiding the time conflict; To strictly meet the task queue, (19) means that *r*-th job does not exist in the queue if there is no the (*r*-1)-th task on the certain processor. For the jobs processed on the same processor, (20) ensures that the *r*-th job cannot be started before the completion of the (*r*-1)-th job.

In addition, (21) guarantees that the completion time each job is not less than the sum of its starting time and the WCET on the certain processor. (22) ensures that the jobs of one task must be processed in order.

Optimization objective

Combine constraints (3)-(6), general forms of (8-11), (7) and (12), general forms of (13-15) and (16), as well as (17-22), the MILP model is built. To improve the efficiency of the system, an objective is designed as follows:

$$\min\left(\sum_{t_1\to t_2}(l_{t_1t_2} + q(t_1,t_2)) + \sum_{\delta_k\to t}(ls(k,t) + qs(k,t)\right)$$

For each instance, such a MILP model can be built and used to formulate the problem. If the scale of the instance is small, math solver, such as Gurobi, can be adopted to solve the model and a scheduling solution is presented according to the decision variables solved by Gurobi.

MILP based heuristic

If the scale of instance is too large for the solver, a MILP based heuristic might be adopted inspired by [3] where the 0-1 variables are slacked to make the model easier to solve. In this work, the decision variables are set as non-negative real numbers rather than 0-1 or non-negative integers. Then, the slacked model is solved by Gurobi and a set of $\{c^*_{l,k}\}$ is calculated. Obviously, compared to the feasible $c_{l,k}$, $c^*_{l,k} \leq c_{l,k}, \forall l, k$.

Set $\{c^*_{l,k}\}$ as the deadlines of $\{\tau_{l,k}\}$ and schedule the tasks according to Earliest Deadline First (EDF). The timing constraints formulated in (13-15) and (16) might be met.

[1] Gerber R, Hong S, Saksena M. Guaranteeing End-to-End Timing Constraints by Calibrating Intermediate Processes[C]// Real-time Systems Symposium. IEEE, 1994.

[2] Wu C G, Wang L, Wang J J, A path relinking enhanced estimation of distribution algorithm for direct acyclic graph task scheduling problem[J]. Knowledge-Based Systems, 2021:107255.

[3] Sundar S, Liang B, Offloading dependent tasks with communication delay and deadline constraint[C]// IEEE Conference on Computer Communications, INFOCOM 2018, IEEE, 2018, 37-45.

S-Bottleneck Scheduling with Safety-Performance Trade-offs in Stochastic Conditional DAG Models

Ashrarul H. Sifat*, Xuanliang Deng*, Shao-Yu Huang, Burhanuddin Bharmal, Sen Wang, Ryan K. Williams, Haibo Zeng, Changhee Jung, Jia-bin Huang
Virginia Tech, Purdue University, University of Maryland

***Abstract*—In this paper, we propose a general solution to the problem of scheduling real-time applications on heterogeneous hardware platforms. To fully utilize the computing capacity of heterogeneous processing units, we model the real-time application as a heterogeneous Directed Acyclic Graph (DAG) which specifies the types of processors (CPU, GPU, etc.) where each task should run. In this well-known DAG context, we propose a novel extension aimed at safety-critical systems that operate in unpredictable environments: the coupling of conditional DAG nodes with *stochasticity*. Specifically, conditional DAG nodes enable the modeling of systems that execute computational pipelines based on *environmental context*, while stochasticity of DAG edges captures the uncertain nature of a system's environment or the reliability of its hardware. Furthermore, considering the pessimism of deterministic worst-case execution time (WCET) in scheduling processes, we model execution times of tasks (DAG nodes) as *probability distributions* which yields a novel *stochastic conditional DAG* model. Coupled with a novel S-bottleneck heuristic and safety-performance (SP) metric, our proposed framework allows for efficient online scheduling in complex computational pipelines, with more flexible representation of timing constraints, and ultimately, safety-performance trade offs.**

I. System model

A. Stochastic Heterogeneous Conditional DAGs

In the challenge problem model, there are precedence constraints among different computational tasks, i.e., tasks are connected with input/output ports following a specific execution order. To capture this nature, we propose a new DAG task model, the *Stochastic Heterogeneous Conditional Directed Acyclic Graph (StochHC-DAG)*, which incorporates both the timing and resource constraints for safety-critical autonomous systems. Our core concept is to model computational pipelines that execute conditionally under some uncertainty, recognizing that not all outcomes can be perfectly predicted in real-world applications (e.g., extreme events). In practice, the execution times of tasks are not perfectly known before run-time and may vary *online* due to environmental dynamism and conditions of the hardware platform. Thus, we propose a generalization of the challenge problem model by assuming the execution times of tasks follow probability distributions rather than WCET to reduce schedule pessimism. To fully utilize this probabilistic information and respect timing constraints, we additionally propose new DAG node structures: (1) stochastic conditional nodes for computational path selection; and (2) sensor/synchronization nodes for controlling the difference of timestamps among data streams. A real-time application is then represented by a *StochHC-DAG*, $G = (V, E, C, Type, Tag)$, described by [1] [2]:

- **V** $= \{v_1, v_2, \cdots, v_n\}$ is the set of IDs for all computational tasks in the application.
- **E** $\subseteq V \times V$ is the set of edges among tasks that indicates the data dependencies, with associated probabilities indicating the likelihood of edge traversal during execution.
- **C** $= \{C_1, C_2, \cdots, C_n\}$ is the set of probability distributions of execution times for all tasks.
- **Type** $= \{type_1, type_2, \cdots, type_n\}$ is the set of types of all tasks. A node in *StochHC-DAG* has one of the following types {Computing, Conditional, Sensor, Sync}
- **Tag** $= \{tag_1, tag_2, \cdots, tag_n\}$ indicates the type of processing units that each task should run onto (e.g. CPU, GPU, DLA etc.).

An example of the proposed DAG framework for the challenge problem is given in Figure 4.

B. Stochastic Conditional Nodes in DAG Models

To capture the stochastic nature of real-time applications, a new stochastic conditional node structure is proposed in *StochHC-DAG*. This node allows for the selection of computational paths in a DAG based on events that occur under uncertainty. For example, consider the event that an autonomous vehicle is in a good environment for object detection/tracking vs. a bad environment. In any given window of time, the outcome of this event is uncertain as detecting/predicting environment conditions in practice is imperfect. Thus, if varying computational pipelines are necessary based on environment conditions, it is critical to model the *distribution* of possible execution times. Figure 1 illustrates the structure of our stochastic conditional node which allows for such a modeling. Nodes v_1 and v_2 represent two different computational paths selected according to environment type. The outgoing edges of the conditional node are associated with probabilities since the detection of environment type is uncertain, allowing us to define execution time *distributions* based on conditional uncertainty.

C. Sensor and Synchronization Nodes in DAG Models

As our stochastic conditional node is based on the concept of environmental events, we propose two new DAG nodes for sensing, data synchronization, and event generation: (1) a

This work was supported by the National Science Foundation under grant CNS-1932074.

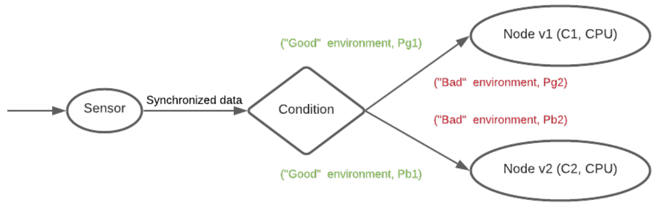


Fig. 1: Structure of stochastic conditional node.

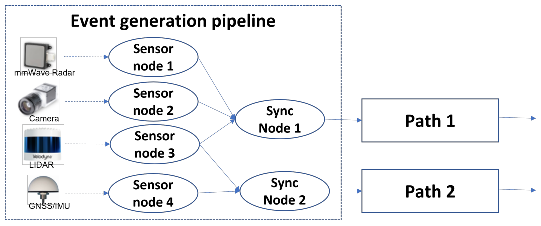


Fig. 2: Event generation pipeline depicting sensor and sync nodes.

sensor node for modeling sensor lag and data post-processing; and (2) a *synchronization node* for fusing sensor data with different frequencies. We then define an *event generation pipeline* comprising these nodes which acts as an event source for our DAG task model (Figure 2). The *sensor node* defines the distribution of lag between the occurrence of a physical event (e.g., nearby obstacle) and the availability of raw sensor data representing the event. Modeling this information in a DAG allows our scheduling algorithm to account for sensor characteristics and satisfy data-based timing constraints. The *synchronization node* then collects data streams from our sensor nodes and outputs a fused data stream with a given user-defined frequency, guaranteeing a bounded difference of data stream timestamps. To implement our synchronization node, we propose a smart ring buffer which utilizes data age and period as illustrated in Figure 3. At each timestep, any new data are written into the smart ring buffer head. The sensor timestamp difference as well as the individual period and age requirements are then verified and data not satisfying these criteria are dropped from the tail. The wide availability of DDS middleware make the information for the synchronicity check readily available for most autonomous systems [3].

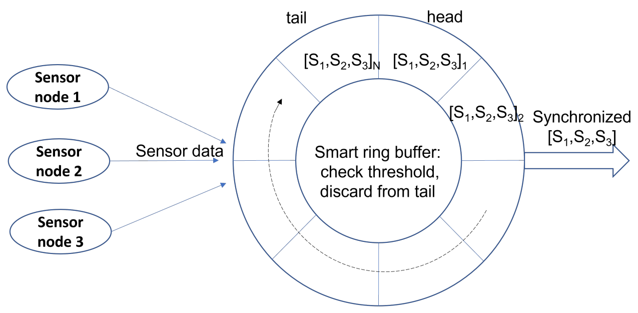


Fig. 3: Smart ring buffer for sensor synchronization in sync node.

II. Shifting Bottleneck Scheduling Algorithm

We propose a new scheduling algorithm which utilizes our proposed DAG model and shifting bottleneck heuristics [4] [5]. Baruah et al. has proposed an exact method to solve the DAG scheduling problem by solving it as an Integer Linear Programming (ILP) problem [6]. However, it only works with the simplest DAG model and he proves in his later work that it is unlikely to write ILP solver for conditional DAG in polynomial time [7]. Therefore, considering the safety-critical requirements and dynamism of autonomous systems, we need an efficient heuristic instead of an exact method.

A detailed explanation and preliminary implementation of our algorithm is provided on https://github.com/Xuanliang-Deng/RTSS2021_Industry_Submission. The process of the algorithm is briefly summarized below.

- **Partition the DAG nodes**: We consider the heterogeneous platform which consists of different types of processing units (e.g., CPU, GPU, DSP etc.). In *StochHC-DAG*, each node is associated with a tag which indicates the processing unit where the node should run. Each node is statically mapped to a processing unit and the mapping is fixed a priori, thereby partitioning the DAG nodes according to their tags and allocating them to the corresponding processing unit.
- **Select Bottleneck Processor**: The starting makespan of the DAG is determined by the maximal finish time (FT) of all nodes on the set of processing units. To select the bottleneck processor, we first assume that there are no resource conflicts and each schedulable task originates at a source node and finishes in a sink node of the DAG. The potential starting time (ST), where a node v_i can start its execution, is the maximal finish time among all its predecessors,

$$ST_i = \max_{k \in pred(i)} FT_k \tag{1}$$

The execution time is denoted as ET_i, which follows a probability distribution, yielding the finish time of node v_i as:

$$FT_i = ST_i + ET_i(C_i) \tag{2}$$

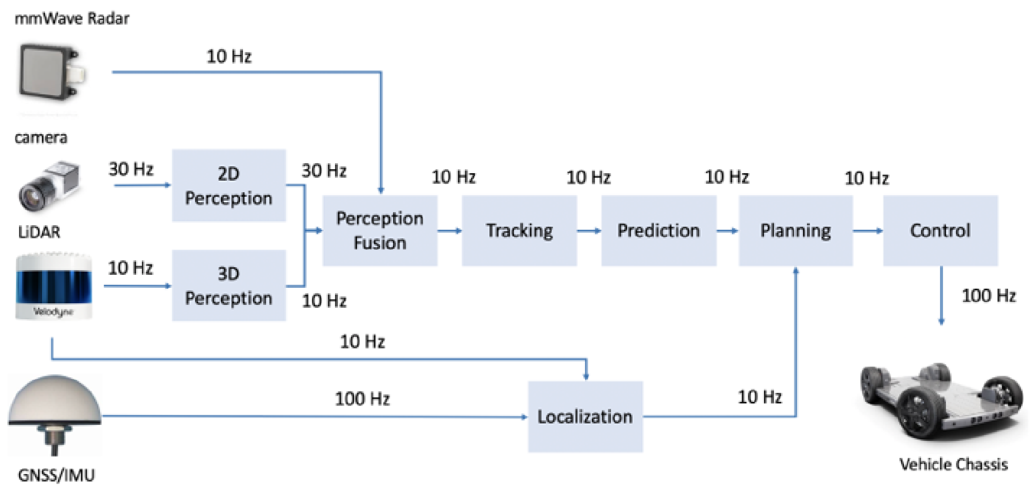
The starting makespan MK_k of processing unit k is,

$$MK_k = \max_{node\ v_i \in proc(k)} FT_i \tag{3}$$

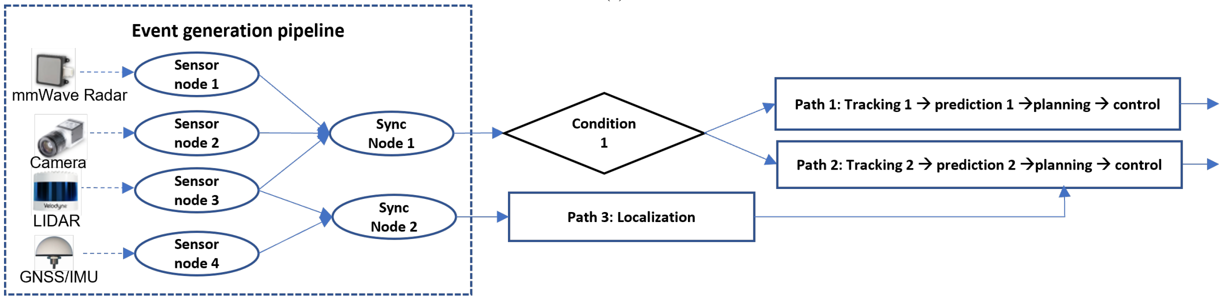
Finally, the starting bottleneck processor is selected by,

$$\max_{k \in 1,2,\ldots,K} MK_k \tag{4}$$

- **Find optimal schedule with Branch and Bound (BnB)**: For selected bottleneck processor, we apply a single-processor analysis by searching with BnB to determine the optimal schedule. This search is different from typical BnB techniques as we utilize the precedence constraints and criticality of nodes in *StochHC-DAG* (see next Section) to greatly reduce the search. Specifically, any potential schedule which violates the precedence constraints



(a)



(b)

Fig. 4: (a) Challenge model of an autonomous vehicle computational system; (b) Proposed Stochastic HPC-DAG framework for the problem statement.

in *StochHC-DAG* will be infeasible. This branch will be cut directly in the search. In addition, nodes with higher criticality are expected to be scheduled ahead of normal DAG nodes on the same processor. The remaining feasible schedules with greatest objective function value (see next Section) will be selected as the optimal one.

- **Shift Bottleneck Processor** Once the optimal schedule of bottleneck processor is determined, we shift the bottleneck to the next processing unit which has maximal value of MK_k in the remaining processing units. The whole process is terminated when all the processors are traversed.

III. A Safety-Performance Metric

Our proposed scheduling approach requires an objective function to optimize when selecting appropriate schedules. While a typical function can be used, such as makespan, we propose a metric that recognizes that safety can live on a spectrum and, when appropriate, safety can be traded off with system performance. Importantly, we do not suggest that safety requirements are ignored, instead we propose to identify safety-critical nodes and paths in our *StochHC-DAG*, allowing our scheduler to ensure safety where necessary and then exploit remaining timing "headroom" to maximize performance. Specifically, we propose a novel *safety-performance metric* by defining a series of penalties/rewards based on violating/satisfying timing constraints in a *StochHC-DAG*. We start with the concept that our metric should penalize when safety-critical paths and/or critical nodes violate timing constraints based on a particular schedule. This implies that a system designer must label all paths and nodes in our *StochHC-DAG* as either safety-critical or non-safety-critical based on the application (e.g., a computational path for pedestrian detection would certainly have a safety-critical label). With this in mind, we define the first term of our metric which penalizes unsafe critical paths:

$$f^{\mathrm{p}}_{\mathrm{cp}}(\mathcal{S}) = \sum_{\ell_i \in \mathcal{C}_{\mathrm{us}}} p_{\mathrm{cp}}(P(R_{\ell_i} > \tau_{\ell_i}) - \lambda_{\ell_i}) \qquad (5)$$

In the above term, we define $\mathcal{C}_{\mathrm{us}}$ as the set of safety-critical DAG paths that violate a *probabilistic* timing constraint, that is, $P(R_{\ell_i} > \tau_{\ell_i}) > \lambda_{\ell_i}$ where R_{ℓ_i} is the random variable describing the uncertain response time of critical path ℓ_i, τ_{ℓ_i} is the *minimally safe* response time for path ℓ_i, and λ_{ℓ_i} is the probabilistic timing constraint for ℓ_i. With these definitions, and noting that $f^{\mathrm{p}}_{\mathrm{cp}}(\mathcal{S})$ represents a penalty term (p) for critical path violations (cp) based on schedule $\mathcal{S}$ with a generic penalty function $p_{\mathrm{cp}}(\cdot)$, equation (5) can be interpreted as penalizing based on the *deviation* of every violating critical path from its probabilistic timing constraint. Thus, if there are no safety-critical paths that violate their timing constraints based on schedule $\mathcal{S}$ then $\mathcal{C}_{\text{us}} = \emptyset$ and $f^{\text{p}}_{\text{cp}}(\mathcal{S}) = 0$ yielding no penalty. Otherwise, the severity of constraint violation dictates the penalty, driving our schedule optimization to improve critical path timing. It is important to note for the above term and all terms defined below, that if a hard timing constraint is desired, one can simply set $\lambda_{\ell_i} = 0$ which enforces sureness of satisfying $R_{\ell_i} > \tau_{\ell_i}$.

Next, we define a similar term for penalizing unsafe critical nodes in a DAG:

$$f^{\text{p}}_{\text{cn}}(\mathcal{S}) = \sum_{v_i \in \mathcal{V}_{\text{us}}} p_{\text{cn}}(P(R_{v_i} > \tau_{v_i}) - \lambda_{v_i}) \tag{6}$$

where $\mathcal{V}_{\text{us}}$ is the set of safety-critical DAG nodes that violate a *probabilistic* timing constraint, that is, $P(R_{v_i} > \tau_{v_i}) > \lambda_{v_i}$ where R_{v_i} is the random variable describing the uncertain response time of critical node v_i, τ_{v_i} is the minimally safe response time for node v_i, and λ_{v_i} is the probabilistic timing constraint for node v_i. Importantly, we model specific terms for critical nodes as there may be instances where a timing constraint for a critical path is satisfied but the system remains unsafe. For example, if a localization and mapping node is too slow, even if the computational path it lies on meets a timing constraint, the staleness of the map may endanger the system or bystanders.

With the penalties for our metric defined, we now describe rewards gained when timing constraints for safety-critical paths are satisfied. Critically, the following reward terms are non-zero *only* when there exists no critical path or node constraints that are violated. In this way, a system will focus purely on safety when required, only balancing safety and performance when all critical constraints are satisfied. The reward terms for our metric are now:

$$\begin{aligned} f^{\text{r}}_{\text{path}}(\mathcal{S}) = \sum_{\ell_i \in \mathcal{P}} & \alpha_{\ell_i} r^{\text{s}}_{\text{path}}(\lambda_{\ell_i} - P(R_{\ell_i} > \tau_{\ell_i})) \\ & + (1 - \alpha_{\ell_i}) r^{\text{p}}_{\text{path}}(P(R_{\ell_i})) \end{aligned} \tag{7}$$

and

$$\begin{aligned} f^{\text{r}}_{\text{node}}(\mathcal{S}) = \sum_{v_i \in \mathcal{V}} & \alpha_{v_i} r^{\text{s}}_{\text{node}}(\lambda_{v_i} - P(R_{v_i} > \tau_{v_i})) \\ & + (1 - \alpha_{v_i}) r^{\text{p}}_{\text{node}}(P(R_{v_i})) \end{aligned} \tag{8}$$

In the above, $f^{\text{r}}_{\text{path}}(\mathcal{S})$ and $f^{\text{r}}_{\text{node}}(\mathcal{S})$ represent a reward term (r) for *every* path and node based on schedule $\mathcal{S}$, respectively, with generic reward functions $r^{\text{s}}_{\text{path}}(\cdot), r^{\text{p}}_{\text{path}}(\cdot), r^{\text{s}}_{\text{node}}(\cdot), r^{\text{p}}_{\text{node}}(\cdot)$ that separately reward safety margins (s) and system performance based on timing (p). Then, with safety-performance balancing parameters $\alpha_{\ell_i}, \alpha_{v_i}$, equations (7) and (8) can be interpreted as rewarding for each DAG path and node, a balance of exceeding timing constraints (safety margin) and system performance related to improved response time ($P(R_{\ell_i})$ and $P(R_{v_i})$). Finally, with all terms defined our scheduler can optimize our safety-performance metric defined as a weighted sum of terms (5)-(8), yielding efficiently computable schedules that trade off safety and system performance relative to probabilistic timing constraints.

References

[1] Zahaf Houssam-Eddine, Nicola Capodieci, Roberto Cavicchioli, Giuseppe Lipari, and Marko Bertogna. The hpc-dag task model for heterogeneous real-time systems. *IEEE Transactions on Computers*, 2020.

[2] Donald W Gillies and Jane W-S Liu. Scheduling tasks with and/or precedence constraints. *SIAM Journal on Computing*, 24(4):797–810, 1995.

[3] Real-Time Innovations (RTI). DDS in Autonomous Car Design.

[4] Subhash C Sarin, Balaji Nagarajan, and Lingrui Liao. *Stochastic scheduling: expectation-variance analysis of a schedule*. Cambridge university press, 2010.

[5] Thomas Morton and David W Pentico. *Heuristic scheduling systems: with applications to production systems and project management*, volume 3. John Wiley & Sons, 1993.

[6] Sanjoy Baruah. Scheduling dags when processor assignments are specified. In *Proceedings of the 28th International Conference on Real-Time Networks and Systems*, RTNS 2020, page 111–116, New York, NY, USA, 2020. Association for Computing Machinery.

[7] Sanjoy Baruah and Alberto Marchetti-Spaccamela. Feasibility Analysis of Conditional DAG Tasks. In Björn B. Brandenburg, editor, *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*, volume 196 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

A General Scheduling Framework for Multi-objective Real-time Systems

Sen Wang[1], Ashrarul Haq Sifat[1], Xuanliang Deng[1], Shao-Yu Huang[2], Changhee Jung[2], Jia-Bin Huang[3], Ryan Williams[1], and Haibo Zeng[1]

[1]Virginia Polytechnic Institute and State University
[2]Purdue University
[3]The University of Maryland, College Park

***Abstract*—We regard the industry challenge as a multi-objective real-time scheduling problem. Different from traditional methods, we model it as a nonlinear program (NLP) and use gradient-based methods for efficient solutions. Different requirements can be freely added as constraints, which gives the proposed method effective for a large range of problems. Furthermore, the proposed NLP scheduling method can utilize available scheduling algorithm as initialization method, and improve their performance even further. Preliminary evaluation shows advantages against simple heuristic methods in complicated problems.**

I. Introduction

In this brief paper, based on the scheduling problem posed by RTSS 2021 industry challenge [1], we model it as a multi-objective scheduling problem and propose a general scheduling method. Specifically, given a computing system and multiple scheduling requirements, our method returns an efficient scheduling algorithm that satisfies all the requirements at the worst cost of pseudo-polynomial complexity. We envision that our approach can address the scheduling problem in a broad range of real-time systems. In the following, we use the original industry challenge as an example to illustrate our methods. Necessary notations are introduced in this section.

We consider a single directed acyclic graph (DAG) model $\mathcal{G} = (V, E)$ to describe computation works. Each node v_i has their own period T_i, worst-case execution time c_i, relative deadline d_i. An edge E_{ij}, (v_i, v_j), goes from node v_i to node V_j means v_j's input depends on v_i's output. The overall DAG graph $\mathcal{G}$ is not necessarily fully connected. The hyper-period, the least common multiple of periods for all nodes in $\mathcal{G}$, is denoted as H. Within a hyper-period, the k-th instance of node v_i starts execution at time s_{ki} *non-preemptively* (required by the industry challenge, but may be relaxed in our method), and finishes at $f_{ki} = s_{ki} + c_i$. For a node v_i with precedence constraints, we denote all its source tasks as $pre(v_i)$, and all its successor as $suc(v_i)$.

A path in DAG is described by a node sequence $\lambda_{be} = \{v_b, ..., v_e\}$, which starts at node v_b and ends at node v_e, and is connected in sequence, i.e., $(v_i, v_{i+1}) \in E$.

Any questions about this paper can be sent to Sen Wang: swang666@vt.edu, Haibo Zeng: hbzeng@vt.edu

There are multiple requirements posed to the scheduling system:

- **Schedulability**. All the nodes should be schedulable, i.e.,
$$r_i \leq d_i \tag{1}$$
In this paper, we consider implicit deadline for simplicity, i.e., $d_i = T_i$.
- **Computation resource bounds**. All the computation resources R_i (e.g., CPU, GPU) are not overloaded for any time interval from t_i to t_j. The mathematical description is given by demand bound function (DBF) [2] as follows
$$\forall R_i, \text{DBF}(t_i, t_j) \leq t_j - t_i \tag{2}$$
We will give a more detailed description for this requirement in the methodology section.
- **Sensor bound**. If we use v_i to denote a node under consideration, and $\{v_l | v_l \in pre(v_i)\}$ represents all its predecessor nodes, then for any instance v_{ik} with start time at s_{ik}, the time difference of all the source data tokens v_{lj} must be smaller than Θ_s.
$$\forall s_{ik}, \forall s_{lj} \in \{0 \leq l \leq \frac{H}{T_l} | s_{lj} \leq s_{ik} < s_{l(j+1)}\} : \tag{3}$$
$$\max_l s_{lj} - \min_l s_{lj} \leq \Theta_s \tag{4}$$
- **Event chain**. For all the event chains λ_{be}, the response time from its start at s_{bi} to its end at s_{ej} should be bounded:
$$\forall \in \lambda_{be}, s_{ej} + c_e - s_{bi} \leq \Theta_e \tag{5}$$
where the instances of $v_{ik} \in \lambda_{be}$ are matched by the following constraints:
$$\forall b \leq i \leq e, \ s_{(i-1)l} \leq s_{ik} \leq s_{(i-1)(l+1)} \tag{6}$$
- **Period constraints**. All the instances of each task cannot start earlier than the beginning of their periods.
$$\forall i, k, \ s_{ik} \geq T_i(k-1) \tag{7}$$
- **DAG dependency**. Each node cannot start until all its dependency tasks have finished.
$$\forall v_i, \forall l \in pre(v_i), s_{l0} + c_l \leq s_{i0} \tag{8}$$

Unlike traditional DAG systems where all the nodes have same frequency, in our model where different nodes have different frequency, over-sampling or under-sampling [3] would be unavoidable.

II. Methodology

We model the scheduling problem as an optimization problem, and propose to use gradient-based methods to solve it for efficiency. The main framework is shown in Fig. 1.

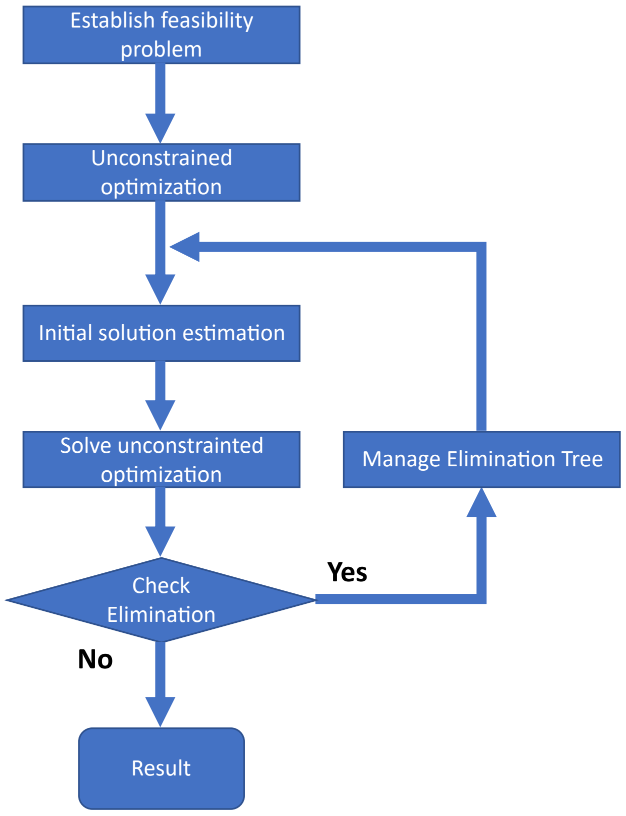

Establish feasibility problem
Unconstrained optimization
Initial solution estimation
Solve unconstrained optimization
Manage Elimination Tree
Check Elimination
Yes
No
Result

Figure 1: Main optimization framework

A. Schedulability analysis

We propose a new method to analyze schedulability for non-preemptive situations based on the ILP formulation proposed by Baruah [4]. The proposed algorithm is proved to yield the same results as [4], but with worst computation cost of only $O(N^2)$ (in average $O(N\log(N))$) as opposed to $O(N^3)$ in [4]. Formal description and proof are skipped because of word limits, but the basic ideas are described as follows:

In non-preemptive situations, an equivalent schedulability analysis can be obtained by calculating "interval" overlapping, where an "interval" is defined as a task instance's execution interval. For example, job v_{ik}'s execution interval starts at s_{ik}, and ends at $s_{ik} + c_i$. If all the intervals are not overlapping with each other, then the system is schedulable.

B. Optimization problem formulation

In this paper, we decide to formulate the scheduling problem as an optimization problem. Inspired from operation research [4], the variables are the start time of all the node instances in the DAG graph $\mathcal{G}$ within a hyper-period. Such modeling method is very flexible and can easily describe all the constraints without pessimistic assumptions. The constraints are described in the previous section, as specified by Equation eqs. (1) to (8). The optimization problem that we are considering is general, in the sense that different constraints can be added or removed freely if only they can be expressed mathematically.

The objective of the optimization is to find a set of start variables for each task instance such that all the constraints are satisfied.

C. Approximated unconstrained optimization

Depending on task periods, the optimization problem proposed could have a large number of variables and constraints. As such, integer linear programming proposed in Baruah's formulation [4] is not appropriate in our case. However, efficient algorithms exist by noticing that most constraints as described above are only concerned with a few variables. Such sparsity is well exploited in many robotics and optimization problems such as simulated localization and mapping (SLAM) [5], [6], motion planning [7], where thousands or even millions of variables are optimized together with fast speed.

To exploit such sparsity with nonlinear optimizer, we transform the feasibility problem above into an unconstrained optimization problem with barrier function [8]. Given a constraint such as

$$f(x) \leq 0 \tag{9}$$

It is transformed into an objective function

$$\text{Barrier}(f(x)) = \begin{cases} 0, & f(x) \leq 0 \\ g(x), & \text{otherwise} \end{cases} \tag{10}$$

where the $g(x) > 0$ is a punishment function for violated constraints.

Since most constraints are linear with respect to their variables, the punishment function is usually straightforward to design. For example, the schedulability constraints

$$r_i(\mathbf{s}) - d_i \leq 0 \tag{11}$$

is transformed to

$$\text{Barrier}(r_i(\mathbf{s}) - d_i) = \begin{cases} 0, & r_i(\mathbf{s}) - d_i \leq 0 \\ d_i - r_i(\mathbf{s}), & \text{otherwise} \end{cases} \tag{12}$$

After transforming constraints into objective function with the barrier method, we formulate a least-square minimization problem as follows:

$$\min_{\mathbf{s}} \sum_{m=1}^{M} \text{Barrier}^2(f_i(\mathbf{s})) \tag{13}$$

Since the Barrier function always gives a positive error if some constraints are violated, objective function 13 establishes a *necessary and sufficient* condition for schedulability analysis: **s** is schedulable if and only if:

$$\sum_{m=1}^{M} \text{Barrier}^2(f_i(\mathbf{s})) = \mathbf{0} \tag{14}$$

D. Gradient-based optimization method

After formulating an optimization problem, we use gradient-based trust-region methods [7], [9] to solve it.

1) Numerical Jacobian evaluation: Since many constraints are not differentiable, numerical methods are used to estimate Jacobian matrix in these situations:

$$\frac{\partial f}{\partial x_i} = \frac{f(x_1, ..., x_i + h, ..., x_N) - f(x_1, ..., x_i - h, ..., x_N)}{2h} \tag{15}$$

The h parameter above should be *reasonably* small to properly estimate Jacobian matrix. Although numerical Jacobian is simple, analytic Jacobian should always be preferred whenever possible because it would save lots of computation cost.

2) Vanishing gradient problem: During optimization, some points have 0 gradient with non-zero error, as shown in Fig. 2. To handle this issue, a simple idea would be increasing the granularity, i.e., h in Equation 15, until the gradient is not zero if the interval overlap error is not 0. More effective ideas such as random walk will be exploited in the future.

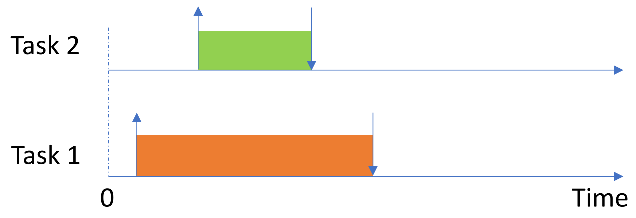


Figure 2: When task 1's *interval*(see section II-A) fully covers task 2's interval, gradient for both the start time of task 1 and task 2 would become 0, even though the system is not schedulable.

E. Managing elimination forest

In this section, we propose a new algorithm named *elimination forest* to bring more efficient and effective optimization. Let's consider an unconstrained optimization problem:

$$\min_x f(x) \tag{16}$$

where $f(x) : R^n \to R^m$ is a discrete function. Furthermore, we assume this optimization problem is solved by gradient-based methods as mentioned above. Two definitions are given there for the ease of presentation.

Definition II.1 (Local optimal point)**.** *A point x_0 is a local optimal point if*

$$\forall \Delta \in \{R^n |\ |\Delta| = 1\},\ \delta \to 0 :$$
$$f(x_0) \leq f(x_0 + \Delta\delta) \tag{17}$$

Definition II.2 (Pseudo-local optimal point (PSOP))**.** *A point x_0 for an objective function $f(x)$ is called pseudo-local optimal point if*

- *It is judged as a local optimal point from Jacobian J, Hessian H or their variants:*

$$\delta \to 0 : f(x_0) < f(x_0 + \Delta(J, H)) \tag{18}$$

where Δ is given by the common gradient based methods such as steepest descent, Gauss-Newton, Levenberg-Marquardt (LM) [10], [11], Dogleg(DL) [12], etc.

- *It is not a local optimal point, i.e.,*

$$\exists \Delta \in \{R^n |\ |\Delta| = 1\},\ \delta \to 0 :$$
$$f(x_0 + \Delta\delta) \leq f(x_0) \tag{19}$$

1) Leaving PSOC by elimination: In our experience, it is very common that the optimization algorithm is stuck at a local optimal or a pseudo-local optimal point. Two possible reasons are summarized:

- Numerical Jacobian at a discrete point is not estimated appropriately such that the Jacobian matrix becomes very large towards one direction. As a result, optimizer tends to go along the opposite direction, which may be misleading.
- Some constraints are very tight such that slightly adjusting some variables to optimize the objective function in one direction will cause other constraints to be violated.

Both two situations suggest that we may be able to leave this situation if we can leave some dimension of objective function out of consideration. For example, if the optimization algorithm terminates at some point $\mathbf{x}_0$ and $\mathbf{f}(\mathbf{x}_0) > 0$, we would go through each dimension $\mathbf{f}_i(\mathbf{x})_0$, if we find that

$$f_i(\mathbf{x}_0) = \theta \tag{20}$$

where $\theta \geq 0$ is smaller than a very small pre-defined threshold, then we stop optimizing toward this criteria, and eliminate this dimension from objective function. To keep variables remain at $f_i(\mathbf{x}_0)$ in the future, we transform it into a constraint added to the variables in such a form

$$x_k = g(\mathbf{x}_0, \theta) \tag{21}$$

In other words, x_k depends on other variables and can be derived from Equation 21. In the following optimization loops, x_k will always be replaced with its dependency variables based on Equation 21.

2) Building elimination forest: As the iteration loop continues, more and more variables may be eliminated and could bring confliction. Inspired from Bayes tree proposed by Kaess *et al.* [13], we propose elimination forest, an efficient algorithm for managing elimination.

Building an elimination forest is simple. First create a node (root for a tree) for each variables. Each time a variable is eliminated, add dependency edges from the eliminated variables to the dependency variables. Since all the nodes in a tree have fixed relative value (given by equation 21), adding a

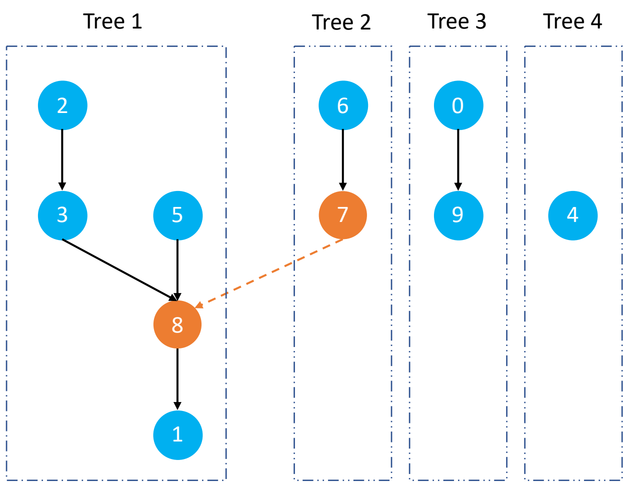


Figure 3: Managing elimination forest: After adding an edge from node 7 to node 8, all the nodes in tree 1 and 2 have fixed relative value. Careful attention must be paid to prevent related nodes, such as node 6 and node 2, violate some constraints.

new dependency may indirectly make some nodes in those two trees violate some constraints. In that case, confliction check must be performed to all the nodes in the two related trees. An example is given in Fig. 3.

Building elimination forest is also helpful in improving algorithm efficiency because all the nodes inside an elimination tree are already checked to be confliction-free. When two nodes are considered to be added together, we only need to check confliction between the two trees. It will save more time and effort when performing optimization with respect to a large number of variables.

F. Initial solution estimation

Flexible initialization policy makes NLP compatible with most available scheduling algorithms to achieve further performance boost. If only one scheduling algorithm could satisfy parts of requirements, then it can be used as initialization and the proposed algorithm is expected to work better. For example, Rate Monotonic is used in our experiments.

III. Preliminary results

Because of time limits, we only finished partial, slow implementation of the proposed ideas. The algorithm is tested in 500 random task sets, whose periods are limited to a small range $\{100, 200, 300, 400, 500\}$ for fast evaluation. The results are summarized in Table tables I to V. The first row reports initialization method's performance, second row reports baseline optimization method simulated annealing (SA), third row are the proposed NLP algorithm. All the experiments within one table have the same initialization, and all the experiments are evaluated based on the same task sets.

Several preliminary conclusions can be drawn from these experiments:

Algorithm 1: Managing elimination forest inside one iteration

```
Input: variables x after performing unconstrained
       optimization, elimination graph $G$
Output: elimination graph $G$
1  if $G$ is empty then
2      Add x as independent nodes to $G$
3  else
4  end
   // iterate over objective function
5  for $(i = 0;\ i < m;\ i = i + 1)$ do
6      if $f_i(\mathbf{x}) \leq \theta$ then
7          $x_0$ = Eliminated variable
8          $X$ = Dependency variables
9          Trees = {}
10         for $(x_j \in \{x_0, X\})$ do
11             Trees.add( ExtractSubGraph($G$, $x_j$))
12         end
13         if $CheckConfliction(Trees)$ then
14             $G.\ AddEdge(x_0, X)$
15         end
16     end
17 end
18 return G
```

- The proposed algorithm improves around 50% to 1000% acceptance rate in the experiments, which proves the validity and effectiveness of the algorithm.
- The proposed algorithm can easily handle systems with different constraints and requirements.
- Although relying on a reasonable initialization method, the proposed algorithm does not require a very good initialization, which is critical in practice.
- It takes a longer time than the baseline methods, which we believe is mostly an implementation issue and can be improved.

In the near future, we will perform more extensive experimental evaluation on larger random task sets. Another major goal is to look at ways to improve the runtime efficiency of the proposed methods.

References

[1] PerceptIn, "2021 rtss industry challenge." http://2021.rtss.org/industry-session/, 2021.

[2] S. Baruah, M. Bertogna, and G. Buttazzo, "Multiprocessor scheduling for real-time systems," in *Embedded Systems*, 2015.

[3] J. Abdullah, G. Dai, and W. Yi, "Worst-case cause-effect reaction latency in systems with non-blocking communication," *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1625–1630, 2019.

[4] S. Baruah, "Scheduling dags when processor assignments are specified," *Proceedings of the 28th International Conference on Real-Time Networks and Systems*, 2020.

[5] F. Dellaert and M. Kaess, "Factor graphs for robot perception," *Found. Trends Robotics*, vol. 6, pp. 1–139, 2017.

[6] F. Dellaert and M. Kaess, "Square root sam: Simultaneous localization and mapping via square root information smoothing," *The International Journal of Robotics Research*, vol. 25, pp. 1181 – 1203, 2006.

Table I: Random task sets subject to DBF, DAG constraints, single processor

Algorithm	Accept rate (Error<1)	Accept rate (Error<0.1)	Average Initial Error (time units)	Average Optimized Error (time units)	Average time (seconds)
RM&DAG_Initialize	54.4%	54.4%	54.87	-	2e-4
SA_RM_Optimize	54.4%	54.4%	54.87	54.87	0.446
NLP_RM_Optimize	72%	66.6%	54.87	3.43	0.165

Table II: Random task sets subject to DBF, DAG, SensorBound constraints, single processor

Algorithm	Accept rate (Error<1)	Accept rate (Error<0.1)	Average Initial Error (time units)	Average Optimized Error (time units)	Average time (seconds)
RM&DAG_Initialize	31.4%	31.4%	158.29	-	2e-4
SA_RM_Optimize	31.4%	31.4%	158.29	158.29	0.443
NLP_RM_Optimize	58%	53.6%	158.29	31.12	1.58

Table III: Random task sets subject to DBF, DAG, SensorBound, two processors

Algorithm	Accept rate (Error<1)	Accept rate (Error<0.1)	Average Initial Error (time units)	Average Optimized Error (time units)	Average time (seconds)
RM&DAG_Initialize	40.2%	38.4%	115.04	-	2e-4
SA_RM_Optimize	40.2%	38.4%	115.04	115.04	0.471
NLP_RM_Optimize	66.2%	63.4%	115.04	22.28	1.65

Table IV: Random task sets subject to DBF, DAG constraints, two processors

Algorithm	Accept rate (Error<1)	Accept rate (Error<0.1)	Average Initial Error (time units)	Average Optimized Error (time units)	Average time (seconds)
RM_Initialize	7.8%	7.8%	91.45	-	3e-4
SA_RM_Optimize	7.8%	7.8%	91.45	91.45	0.453
NLP_RM_Optimize	**81.6%**	78.8%	91.45	2.63	0.22

Table V: Random task sets subject to DBF, DAG, SensorBound, two processors

Algorithm	Accept rate (Error<1)	Accept rate (Error<0.1)	Average Initial Error (time units)	Average Optimized Error (time units)	Average time (seconds)
RM_Initialize	6.8%	6.8%	168.94	-	3e-4
SA_RM_Optimize	6.8%	6.8%	168.94	168.94	0.451
NLP_RM_Optimize	**66.2%**	64.8%	168.94	24.50	2.13

[7] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time gaussian process motion planning via probabilistic inference," *The International Journal of Robotics Research*, vol. 37, pp. 1319 – 1340, 2018.

[8] S. P. Boyd and L. Vandenberghe, "Convex optimization," *IEEE Transactions on Automatic Control*, vol. 51, pp. 1859–1859, 2006.

[9] S. Wang, J. Chen, X. Deng, S. Hutchinson, and F. Dellaert, "Robot calligraphy using pseudospectral optimal control in conjunction with a novel dynamic brush model," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6696–6703, 2020.

[10] K. Levenberg, "A method for the solution of certain non – linear problems in least squares," *Quarterly of Applied Mathematics*, vol. 2, pp. 164–168, 1944.

[11] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of The Society for Industrial and Applied Mathematics*, vol. 11, pp. 431–441, 1963.

[12] M. Powell, "A new algorithm for unconstrained optimization," 1970.

[13] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, "isam2: Incremental smoothing and mapping using the bayes tree," *The International Journal of Robotics Research*, vol. 31, pp. 216 – 235, 2012.

Achieving Time Determinism using System-Level LET

Robin Hapka, Sebastian Abel, Kai-Björn Gemlau, Mischa Möstl, Rolf Ernst
Institute of Computer and Network Engineering
TU Braunschweig
Braunschweig, Germany
hapka|sebastiana|gemlau|moestl|ernst@ida.ing.tu-bs.de

***Abstract*—As a feasible solution for the RTSS 2021 industry challenge, we propose System-Level Logical Execution Time (SL-LET) in combination with a modified middleware layer.**

***Index Terms*—real-time, industry challenge, time determinism, logical execution time, middleware**

I. Introduction

The RTSS 2021 industry challenge deals with real-time constraints in deep processing pipelines with strong dependencies among different stages. Such pipelines are used for autonomous vehicles and an example is illustrated in the challenge's announcement. At first, we want to clarify the concept of an ”event”, we define it as the arrival of an input data token at the first stage of the pipeline. The challenge asks for strategies to guarantee the following three timing-constraints:

Requirement 1: Reaction time

For any event, the end-to-end latency from a source to a destination is bounded by a predefined value.

Requirement 2: Data age

For a data token b produced at a specific final output, the causing token must not be older than a predefined threshold.

Requirement 3: Data age difference

For each task, the age difference among all its input data is bounded by a predefined value.

Fig. 1 shows the periodical activation of the two tasks τ_1 and τ_2 with a read/write dependence. In addition, it illustrates our understanding of reaction time and data age. Thereby, the reaction time is the delay from the causing event to the first output data token considering this event. The data age is the elapsed time beginning from the output of a certain task instance to the past event, which caused it. Typically, the considered task instance is the final instance i.e. the last one processing the event's input data token. Reaction time and data age are therefore very similar and can be regarded as forward and backward definition of end-to-end latency of data token flow [1].

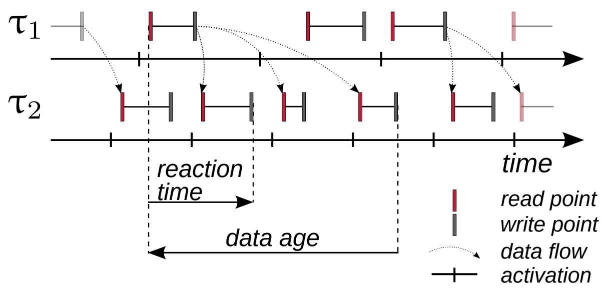


Fig. 1. Data propagation in pipeline with read/write jitter

We propose System-Level LET (SL-LET) as a design model and programming paradigm for the system type envisioned by the challenge's “problem model”. SL-LET is an extension of the well known Logical Execution Time (LET) programming paradigm [2], where each task has a fixed logical execution time. The reading and writing of input/output data tokens occurs instantaneously at the beginning and end of the LET interval. Because the SL-LET concept enforces the fixed logical execution time, it allows to predetermine when data tokens are written and read by a task. Thus, SL-LET achieves deterministic timing of the data token flow. As a result, system designers can easily compute these read/write time instants and derive end-to-end timing bounds for the data token flow along a processing pipeline [3]. This is in harsh contrast to an execution model where outputs of a task instance are written immediately after the instance's completion (see Fig. 1). This leads to many possible scenarios for data-age and reaction time.

Furthermore, we propose the modification of an existing middleware layer to deploy SL-LET and time determinism as easily and transparent as possible.

In the following we will explain SL-LET in more depth, discuss how it fulfills the given requirements, where it offers more generality and where it requires stricter constraints.

II. System-Level LET

SL-LET [4] is an extension to the LET design paradigm. Therefore, the main objectives of LET will be explained first [2].

The time an instance j of task τ_i takes from activation until the instance is completed is called the response time $R_{i,j}$. Thereby, the maximum possible response time is called the worst-case response time (WCRT) and is denoted as R_i. In the real-time domain it is often demanded that a task always completes before or at least at its deadline d_i i.e. $R_{i,j} \leq d_i \; \forall j$. But the response time $R_{i,j}$ varies depending on numerous factors from run to run. Such a variation is commonly known as (response time) jitter. Control systems require time-deterministic input/output behavior to reach the specified level of stability and a certain degree of robustness. But jitter negatively affects determinism and is therefore unwanted or even unacceptable for such systems.

LET enables a jitter-free communication by masking the physical execution time with a fixed logical execution time. Instead of $R_{i,j} \leq d_i \; \forall j$, LET enforces $R_{i,j} = LET_i \; \forall j$ i.e. the response time for every instance is supposed to equal a constant value LET_i which is known as logical execution time. However, to meet the deadline, it is necessary to set $LET_i \leq d_i$. Since LET changes the requirement of the response time, it is also considered as a design model.

To be able to abstract from physical execution time, the implementation needs to enforce the logical execution time. To achieve this, LET requires read/write operations in zero time and an instantaneous write propagation to all readers. This is a strong assumption, but it can be fulfilled for inter-process communication within a non-distributed platform featuring a shared memory architecture. Since automotive software components are typically distributed among different hardware units and exhibit significant communication delays, we aim to provide a general solution to cope with such complex problems. In consequence, we assume that data propagation is not instantaneous, but can be bounded by a constant value. To be able to deploy LET despite communication delay e.g. between different processing platforms, SL-LET was introduced.

SL-LET preserves the properties of LET for communication within a single platform and exclusively deals with data transfer in case of communication delay. Similar to the physical response time of a task executing on a non-distributed hardware platform, the physical communication time, i.e. the exchange of data tokens between distributed hardware platforms, takes a certain amount of time and exhibits a jitter, therefore affecting time determinism. Hence, the solution is the same as the one applied to the physical response time: masking it. The transfer time is hidden in the response time of a so called SL-LET interconnect task. Thereby, the task's only objective is to copy data from one platform to another time-deterministically. This is achieved by reading at the beginning and writing at the end of the logical execution time interval LET_i. The time instants of reading and writing are fixed and predetermined based on the platforms local clocks. As a result, it is necessary to synchronize the local clocks among each other, while a small error of ϵ is tolerated [5]. Note, that synchronized clocks are quite common in today's distributed networks.

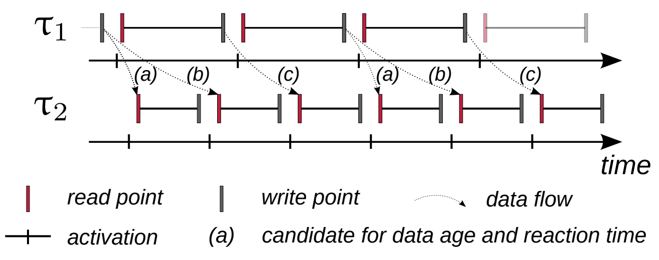


Fig. 2. Data propagation in pipeline without read/write jitter

III. Solving the problem model

As already mentioned, the read and write points in the SL-LET paradigm are performed at fixed time instants and are not affected by jitter or delays. Subsequently, tasks implemented with the SL-LET paradigm are *time-deterministic* i.e. a task produces a sequence of outputs with the exact same timing behavior at any iteration. Due to that benefit, engineers can easily calculate upper-bounds for the end-to-end latency. In contrast to that, in a traditional execution model, tasks would write their outputs immediately after an instance has finished. Hence, the resulting output jitter of output data tokens would be between best-case response time (BCRT) and WCRT. It would further be amplified by the fact, that it is not guaranteed for tasks to read their inputs at the point of their activation, as they may experience blocking due to scheduling. Note that the combined effects of output and input jitter accumulate along a pipeline. This would create a large continuum of possible reaction times, data ages and also data age difference. Such a behavior is illustrated in Fig. 1, where two tasks are activated with a certain period and write their results at the moment the calculation is completed. Since the jitter is hardly predictable, the reaction time and data age of an event is unknown and can only be upper and lower bounded [6].

In contrast to that, the outstanding property of SL-LET limits the computed reaction time and data age to a few possible candidates rather than (continuous) intervals between a minimum and a maximum. The same example as in Fig. 1 is shown in Fig. 2, but with SL-LET and therefore without jitter. It can be seen, that the given example has only three possible candidates for data age and reaction time (these are denoted in Fig. 2 as "(a)", "(b)", and "(c)"). This simplifies the calculation of upper-bounds for reaction time as well as for data age compared to traditional execution models. Same holds true for the data age difference, since there is only a limited number of possible candidates [7]. SL-LET therefore not only upper bounds the reaction time, data age and data age difference, but these values can be computed at design-time, because the candidates occur in a deterministic order.

Another useful benefit of SL-LET is its *composability* with respect to the LET_i [5]. That means, as long as the logical execution time interval LET_i is neither shortened nor extended, the task itself can be completely rewritten, updated or even moved to another processing unit without changing the timing behavior of the data token flow. This is a key property to update (autonomous) vehicles without necessarily repeating timing verification processes for every update and the whole system. Note, in systems with traditional execution models, the update of a single task would change the timing behavior completely. This would not only include tasks in a read/write relation to the updated one, but any task in the system.

The challenge's problem model states that the worst-case execution time (WCET) of each task is known in advance and the scheduling is performed non-preemptively. Despite the fact, that SL-LET supports preemptive scheduling algorithms as well, these assumptions simplify the determination of logical execution time intervals LET_i. Without activation jitter (this is enforced by SL-LET) and preemption the WCRT can easily be calculated from the WCET and thus LET_i can be set to the task specific WCRT. In fact, SL-LET guarantees the required timing-constraints of the challenge solely, due to a time-deterministic communication, based on the enforcement of jitter-free communication.

Though SL-LET offers numerous benefits, it may seem challenging to adopt an existing system of applications to the SL-LET programming paradigm. Especially, if applications are used, which are developed by suppliers or communities unfamiliar with real-time constraints. We are aware of such reservations and subsequently implemented SL-LET in the widespread middleware framework Data Distribution Service (DDS) [8]. Namely, we extended *eProsima's Fast-DDS* implementation[1]. Since many applications in automotive (e.g. AUTOSAR [9]) or in the robotic domain (e.g. ROS2) support or even completely rely on DDS for communication, it enables the transparent usage of SL-LET without further effort, except providing a task specific LET_i.

IV. Conclusion

We described the main properties and benefits of SL-LET and illustrated that a time-deterministic communication is sufficient to fulfill the RTSS 2021 industry challenge's timing constraints. Time determinism enables the system designer to derive upper-bounds for reaction time, data age and data age difference while the SL-LET implementation guarantees these upper-bounds. Furthermore, a middleware, namely DDS, can be deployed to switch comfortably from a traditional execution model to SL-LET.

References

[1] N. Feiertag, K. Richter, J. E. Nordlander, and J. Å. Jönsson, "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics," in *RTSS 2009*, 2008.

[2] C. M. Kirsch and A. Sokolova, "The logical execution time paradigm," in *Advances in Real-Time Systems*, 2012.

[3] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Analyzing end-to-end delays in automotive systems at various levels of timing information," *SIGBED Rev.*, vol. 14, no. 4, p. 8–13, Jan. 2018. [Online]. Available: https://doi.org/10.1145/3177803.3177805

[4] K.-B. Gemlau, L. Köhler, and R. Ernst, "System-Level Logical Execution Time: Augmenting the Logical Execution Time Paradigm for Distributed Real-Time Automotive Software," *ACM Transactions on Cyber-Physical Systems*, Jan. 2020.

[5] R. Ernst, L. Ahrendts, and K.-B. Gemlau, "System level let: Mastering cause-effect chains in distributed systems," in *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*, 2018, pp. 4084–4089.

[6] K.-B. Gemlau, J. Schlatow, M. Möstl, and R. Ernst, "Compositional Analysis for the WATERS Industrial Challenge 2017," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, Dubrovnik, Croatia, Jun. 2017.

[7] M. Möstl, "On timing in technical safety requirements for mixed-critical designs," Ph.D. dissertation, TU Braunschweig, 2021, note: Dissertation has been accepted, will be published soon. [Online]. Available: https://doi.org/10.24355/dbbs.084-202109130759-0

[8] G. Pardo-Castellote, "Omg data-distribution service: architectural overview," in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, 2003, pp. 200–206.

[9] "Specification of communication management," AUTOSAR, Tech. Rep. R20-11, Nov 2020.

[1] https://github.com/eProsima/Fast-DDS

RTSS Industrial Challenge: Towards I/O-Predictable Real-Time Autonomous Systems

Ze Jiang[†§], Xiaotian Dai[†], Neil Audsley[‡]
[†]University of York, UK, [§]ARM Ltd, UK, [‡]City, University of London, UK,

I. Introduction and Research Challenge

Input/Output (*I/O*) peripherals are a vital part in modern embedded real-time systems, as the I/Os often interface with physical sensors and actuators that need to either sense a potential hazard in time or make a manoeuvre to avoid danger. In autonomous driving system [1], the planning module makes decisions based on perception of external environments using different input I/O data, *e.g.*, images, point clouds, and localisation information provided by a camera, a Lidar, and a GNSS, respectively. With the determined driving routine, the vehicle can control controlled correspondingly using different I/Os, *e.g.*, motor controls in chassis.

To ensure the correctness and effectiveness of the safety-critical modules, I/O operations usually require restricted demands on both *timing-predictability* — having an analytical bound for the worst-case, and *timing-accuracy* — getting executed at exact time instants (or at least within a small time range) [2], [3]. Considering the aforementioned example, when the vehicle's location is altered from the expected control time points, the input I/O data can be invalid, and the output I/O control may even cause a mistake.

The I/O timing requirements were implicit in a conventional real-time system, as the systems usually had relatively lower system complexity and fewer I/O peripherals, where a timing-predicable/accurate I/O operation can be achieved based on the interrupt of a high-resolution timer (*e.g.*, the micro-second timer provided by an RTOS [4]–[6]). However, with the growing system complexity and number of I/Os, the *complex I/O access paths* and *complicated resource management* lead timing-predictable and timing-accurate I/O operations to be challenging to achieve. For instance, to access an I/O device in a Network-on-Chip (NoC) based many-core system (see Fig. 1), I/O requests must pass through the OS kernel, I/O manager and different low-level drivers at the software; when it comes to the hardware, the I/O requests are also needed to be transmitted through multiple routers/arbiter and an I/O controller. After processing, corresponding results (*i.e.*, I/O responses) are transmitted back to a processor or a memory module using a similar routine as the I/O requests. Such complex paths introduce significant communication latency and timing uncertainty to the I/O operations. Moreover, along the transaction paths, potential resource contentions occur frequently, which involve additional resource management throughout the entire system. The resource management magnifies the difficulty of satisfying the real-time requirements of I/O operations [3].

II. A Prototype Framework for I/O-Predcitable Real-time Systems

In order to guarantee the timing-predictability and timing-accuracy of I/O operations in a NoC-based many-core system, I/O requests and responses are both required to be taken into considerations. With this in mind, we present

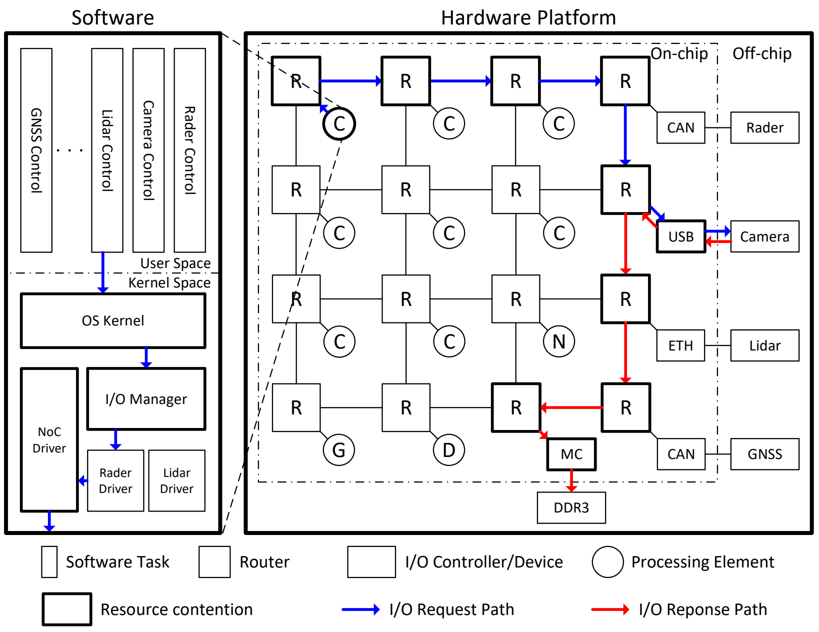


Fig. 1. I/O operations in conventional many-core heterogeneous systems involve complex I/O access paths and resource management at all system levels, leading to challenges in guaranteeing its real-time performance (C: processor core; N: neural engine; G: GPU; D: DSP; MC: memory controller).

- A programmable real-time I/O controller (named *I/O-NoC-RT*) to replace the conventional I/O controllers (*e.g.*, CAN controller in Fig. 1). The *I/O-NoC-RT* can operate periodic I/O requests at exact time points and execute sporadic I/O requests with an analytical time bound.
- An allocation algorithm to allocate *I/O-NoC-RT* to the routers, optimising the routines of I/O responses and reduce contentions on the NoC.

A. *I/O-NoC-RT: A Programmable Real-time I/O Controller*

Typically, I/O tasks in a system can either be *periodic* or *sporadic*. The periodic I/O tasks are usually determined *before* system execution, *e.g.*, periodic sensor read; and the sporadic I/O tasks are often generated *during* system execution, e.g., sporadic body control.

At system initialisation, *I/O-NoC-RT* pre-loads periodic I/O tasks and records their timing information (*e.g.*, starting time points, time budgets, *etc.*) using a *time slot table*. In the time slot table, *I/O-NoC-RT* also reserves certain time slots for the hard real-time (HRT) sporadic I/O tasks in a periodic manner. During system execution, *I/O-NoC-RT* runs the periodic tasks at their specified times without involving the complicated I/O request path and complex resource management (described in Fig. 1), which guarantees their timing-accuracy and timing-predictability. At the same time, *I/O-NoC-RT* also receives and buffers the run-time sporadic I/O tasks issued by the processors, and executes them when the periodic tasks are not occupying the I/O. With the reserved time slots, *I/O-NoC-RT* provides analytical bounds for the HRT sporadic I/O tasks.

***I/O-NoC-RT* design.** Fig. 2 illustrates the micro-architecture of *I/O-NoC-RT*, mainly containing a request channel and a response channel.

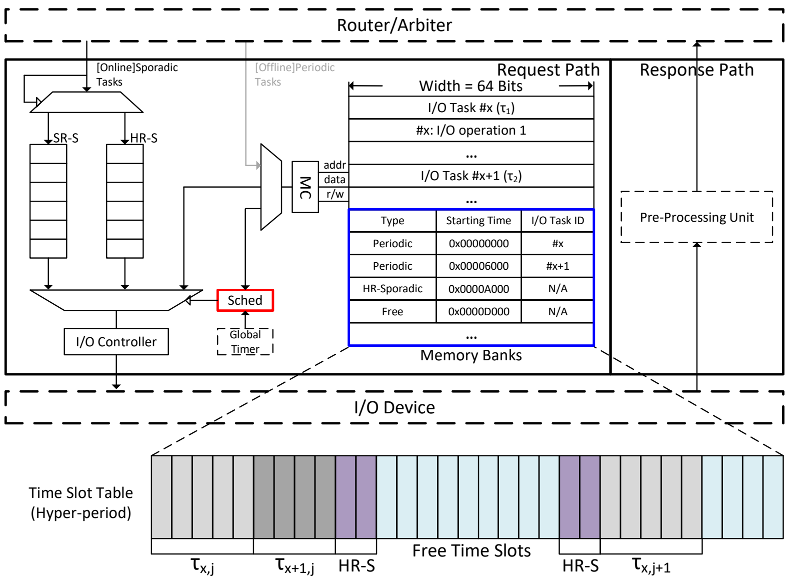


Fig. 2. Micro-architecture of real-time I/O controller (SRT-S: software real-time sporadic tasks; HRT-S: hard real-time sporadic tasks).

Request channel. In request channel, we deploy a memory module to store the periodic I/O tasks and the time slot table, which are loaded during system initialisation. Also, we employ two FIFO queues to buffer the HRT and soft real-time (SRT) sporadic I/O tasks sent by the processors at run-time. Between the memory module and the FIFO queues, we introduce a scheduler, which synchronises with a global timer and compares the synchronised results with the time slot table. Once the system executes at a starting time point of a periodic (HRT sporadic) I/O task, the scheduler loads the task from the memory unit (FIFO queue) to the connected I/O controller for execution. If no task is required to be executed at a time point (*i.e.*, the time slots are free), the scheduler executes the sporadic (either HRT or SRT) I/O tasks.

Response channel. The response channel is pass-through, since the processing speed of the processors is usually hundreds of times faster than the I/O peripherals. Alternatively, a pre-processing unit (*e.g.*, checking the validness of a response) can be deployed in the response channel to reduce data workload of I/O responses. This is implemented with a generalised pre-processing unit that can be loaded with user-defined code. Examples includes (1) implementing a CRC check for most communication protocols, e.g., CAN and Ethernet; (2) define data pre-processing procedure such as dividing images or remove anomaly data using DSP. Note there is only limited computation power in this unit and thus it should be size-bounded and, to guarantee timing, time-bounded.

Router optimization. Additionally, as packets are transmitted non-preemptively on the on-chip network, route optimization can be performed to determine which node to be allocated at which network location, in a way that the interference and contention on the inter-connections can be minimized. The process includes modelling of the I/O streams spatially and temporally, by giving their transmission time and data workload.

B. Summary

Modern real-time embedded systems are integrating increasingly more I/O peripherals, driven by the diverse functionalities required by modern embedded computing and the rapid evolution of manufacturing processes in the semiconductor industry. In modern real-time systems, conventional methods, e.g., relying on high-resolution timers in RTOS, are facing challenges to guarantee the restricted timing demands of I/O operations due to complicated resource management and complex I/O access paths. In this work, we identified that achieving real-time I/O requires considerations of both I/O requests and responses. With this in mind, we proposed (i) a novel I/O controller which can operate periodic I/O requests at exact time points and execute sporadic I/O requests with an analytical time bound; (ii) an allocation algorithm to allocate the new I/O controller to the routers, optimising the routines of I/O responses and reduce contentions on the NoC. Due to the page limits, we only give the top-level overview of the work and leave detailed implementation and evaluation in future work.

REFERENCES

[1] RTSS, "RTSS industrial session," http://2021.rtss.org/industry-session/.

[2] S. Zhao, Z. Jiang, X. Dai, I. Bate, I. Habli, and W. Chang, "Timing-accurate general-purpose i/o for multi-and many-core systems: scheduling and hardware support," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[3] Z. Jiang and N. C. Audsley, "Gpiocp: Timing-accurate general purpose I/O controller for many-core real-time systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 806–811.

[4] FreeRTOS, "FreeRTOS official website," http://www.freertos.org/.

[5] μCos, "μCos official website," https://weston-embedded.com/micrium/overview.

[6] RTEMSWeb, "Rtsms official website," https://www.rtems.org/.