Functionally and Temporally Correct Simulation of Cyber-Systems for Automotive Systems

Kyoung-Soo We, Seunggon Kim, Wonseok Lee, and Chang-Gun Lee
Department of Computer Science and Engineering, Seoul National University, Seoul, Korea
Email: {kswe, sgkim, wslee, cglee}@rubis.snu.ac.kr

***Abstract*—The current simulation tools used in the automotive industry do not correctly model timing behaviors of cyber-systems such as varying execution times and preemptions. Thus, they cannot correctly predict the real control performance. Motivated by this limitation, this paper proposes functionally and temporally correct simulation for the cyber-side of an automotive system. The key idea is to keep the data and time correctness only at physical interaction points and enjoy freedom of scheduling simulated jobs for all other cases. This way, the proposed approach significantly improves the real-time simulation capacity of the state-of-the-art simulation methods while keeping the functional and temporal correctness.**

I. Introduction

For developing an automotive system, it is essential to accurately predict the final performance at the design phase. For such prediction, simulators are commonly used like Simulink [1] and LabVIEW [2]. However, they do not consider timing behaviors of the cyber-system such as varying execution times and task preemptions. Thus, their control performance predictions are far different from the real performance. As an example, for LKAS (Lane Keeping Assistance System) that aims at keeping the vehicle at the center of the lane, Fig. 1 shows noticeable gap between the predicted performance by Simulink simulation and the real performance by the actually implemented cyber-system.

In this paper, we propose a new approach for simulating the cyber-side aiming at

- functionally and temporally correct simulation, that is, our simulation executes all the cyber-side tasks on the simulation PC accurately modeling the timing behavior that will happen on the actually implemented cyber-system and
- real-time simulation, that is, our simulation performs in real-time while interacting with the real working physical-side, i.e., a vehicle dynamics simulator like CarSim RT [3] or an actual vehicle.

The key idea of the proposed approach is to keep the data and time correctness only at the physical interaction points to maximally enjoy the freedom of scheduling simulated jobs. For this, we transform the simulation problem to a real-time job scheduling problem with precedence constraints necessary for the functional and temporal correctness. Then, we propose an efficient scheduling algorithm for the functionally and temporally correct real-time simulation. Our evaluation through both synthetic workload and actual implementation confirms both high accuracy and high efficiency of our approach compared with other state-of-the-art methods such as Simulink [1] and TrueTime [4].

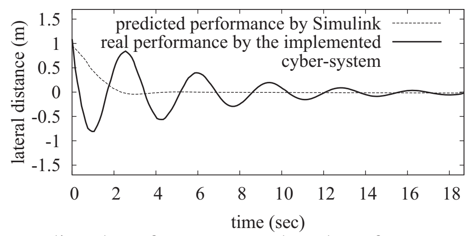


Fig. 1: Predicted performance and real performance of LKAS

The paper is organized as follows. In Section II, we formally describe our problem. Then, Section III explains our proposed approach. In Section IV, we report our experimental results through both synthetic workload and actual implementation. In Section V, we compare our approach with related works. Finally, Section VI concludes the paper.

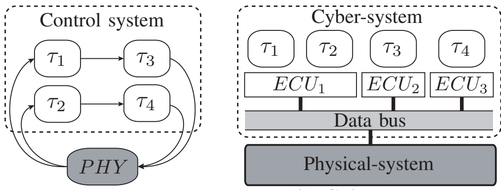
II. Problem Description

Control engineers design a control system as in Fig. 2(a). Then, it should be realized as a cyber-system as in Fig. 2(b). For this cyber-system realization, our goal is to provide a design time prediction on the control performance that the cyber-system will give to the physical-system. Thus, our problem is how to simulate the cyber-system keeping the functional and temporal correctness. In order to give the concept of functionally and temporally correct simulation, let us assume that the real cyber-system will give the job execution scenario as in Fig. 3(a). Our problem is to simulate jobs beforehand on the simulation PC. In the rest of this paper, by **simulating a job**, we mean **executing a job on the simulation PC**. If we simulate the jobs as in Fig. 3(b), it gives the same effect to the physical-system as the real cyber-system. This is because the simulated J_{31} produces the same output value as the real J_{31} since it executes the same function codes with the same data, i.e., the output of simulated J_{11} that also executes the same function codes of real J_{11} with the same physical data of real J_{11}. The output of the simulated J_{31} is given to the physical-system at 6 and hence the physical-system gets the functionally and temporally the same effect from the simulated J_{31} as the real J_{31}. Similarly, the simulated J_{41} gives the same output to the physical-system at the same time as the real J_{41}. Like this, the functionally and temporally correct simulation is to execute the jobs on the simulation PC such that it gives the same functional and temporal effects to the physical-system as if it is the real cyber-system.

A. Descriptions on the real cyber-system

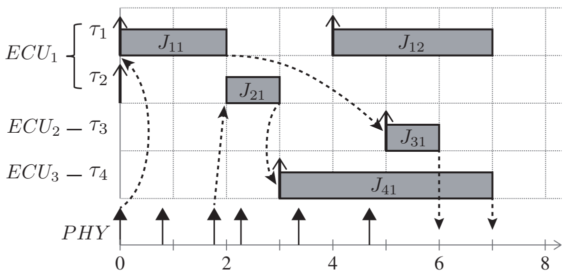
The control system designed by control engineers is given as a graph $G = (V, E)$ as in Fig. 2(a). V is the set of nodes where

1052-8725/17/31.00 ©2017 IEEE
DOI 10.1109/RTSS.2017.00014

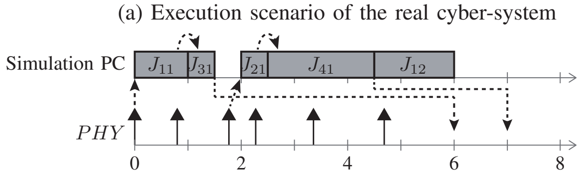


(a) Control system (b) Cyber-system

Fig. 2: Automotive system design



(a) Execution scenario of the real cyber-system



(b) Execution scenario of the simulated cyber-system

Fig. 3: Execution scenario

a node is either a control task denoted by τ_i or the physical-system denoted by PHY. Each control task τ_i executes a function F_i using input data from other tasks or PHY and produces output data to other tasks or PHY. E is the set of edges that represent such data producer/consumer relations among nodes.

A cyber-system realizing the given control system $G = (V, E)$ can be represented by m electronic control units, so called ECUs, i.e.,

$$\{ECU_1, ECU_2, \cdots, ECU_m\},$$

and tasks mapped to each ECU as in Fig. 2(b). We assume that each ECU is an embedded computer equipped with a single core processor like Infineon TC1797 [5], NXP MPC5566 [6] and executes its mapped tasks using a job-level static priority scheduling algorithm such as RM and EDF. The ECUs and PHY are connected through a TDMA (Time Division Multiple Access) bus such as TTCAN (Time Triggered Controller Area Network) [7] and FlexRay [8]. Tasks on the same ECU exchange data using a shared memory and hence the time for such data exchange is negligibly small. On the other hand, tasks on different ECUs exchange data through TDMA slots, which may take non-negligible time. For now, such data exchange time is also assumed to be zero for the simplicity of explanation. In Appendix B, we will address the TDMA bus delay.

Each control task τ_i in $G = (V, E)$ is realized as a periodic task in the cyber-system and represented as a five-tuple

$$\tau_i = (F_i, \Phi_i, P_i, C_i^{\mathsf{best}}, C_i^{\mathsf{worst}})$$

where F_i is the function that τ_i executes, Φ_i is the task offset, P_i is the period, and C_i^{best} and C_i^{worst} are the best and the worst case execution times, respectively. The j-th job of τ_i is denoted by J_{ij}. We assume that each job J_{ij}'s release time $t_{ij}^{\mathsf{R,real}}$ is deterministically $\Phi_i + (j-1)P_i$ without release jitter but its execution time varies within $[C_i^{\mathsf{best}}, C_i^{\mathsf{worst}}]$ depending on its input data.

For control tasks in automotive systems, the following properties generally hold:

- **Most Recent Data Use**: For each data content, there exists a single memory buffer. Thus, the memory buffer is overwritten by the most recently generated data. Therefore, the job that reads the memory buffer always uses the most recently generated data.
- **Entry Read and Exit Write**: Each job reads all the necessary data at the entry of its execution and writes all the output data at the exit of its execution. Both the entry read and the exit write are atomically performed without being preempted and their durations are negligibly short.

B. Description on the simulated cyber-system

For simulating jobs J_{ij}s on the simulation PC, we assume that the function F_i of each task τ_i is compilable not only for the ECU but also for the PC [1].

For executing jobs on the simulation PC, we use a single core in the PC, which is much faster than the ECU processor, e.g., Core i5-7600 [9] in PC vs. TC1797 [5] in ECU. Thus, for a job J_{ij}, its execution time on PC is much shorter than that on ECU. Although the execution times on PC and ECU vary depending on the input data, we assume that there is a strong correlation between the execution time on PC, i.e., e_{ij}^{sim}, and that on ECU, i.e., e_{ij}^{real}, which will be validated in Section IV. Thus, we assume the following mapping function:

$$e_{ij}^{\mathsf{real}} = M_i(e_{ij}^{\mathsf{sim}}).$$

Thus, once we know e_{ij}^{sim} after finishing J_{ij} on the simulation PC, we can estimate the real execution time e_{ij}^{real}.

Also, on the simulation PC, the following data read and write mechanisms are possible:

- **Tagged Data Read**: Unlike the real cyber-system that keeps only the most recent data, the simulation PC can log all the data history with time-tags or producer-tags. Thus, the simulation PC can execute a job with any specific tagged data in the data log.
- **Delayed Data Write**: Unlike the real cyber-system that writes the data immediately after a job finishes, the simulation PC can hold the output data of a job and intentionally delay its actual write to a specific time.

C. Formal definition of the simulation problem

We first define the following notations:

- $t_{ij}^{\mathsf{S,sim}}$, $t_{ij}^{\mathsf{F,sim}}$: The start time and finish time, respectively, of J_{ij} on the simulation PC.
- $t_{ij}^{\mathsf{S,real}}$, $t_{ij}^{\mathsf{F,real}}$: The start time and finish time, respectively, of J_{ij} on the real cyber-system.

[1]This is a valid assumption since control engineers make MATLAB codes or C codes of their control tasks and they can be cross-compiled for the simulation PC and the ECU. The compiler ensures that the computational results on both platforms are the same. The precision error due to the arithmetic difference is negligible and beyond the scope of this paper.

A simple minded solution for the functionally and temporally correct cyber-system simulation is that the simulation PC starts and finishes each job J_{ij} at the same times as the real start and finish times, i.e., $t_{ij}^{\mathsf{S,sim}} = t_{ij}^{\mathsf{S,real}}$ and $t_{ij}^{\mathsf{F,sim}} = t_{ij}^{\mathsf{F,real}}$ $\forall J_{ij}$. However, such a solution is not practical since it too much restricts the scheduling freedom of simulated jobs.

Overcoming this restriction, our key idea is to maximally enjoy the freedom of scheduling simulated jobs by keeping the data and time correctness only at the physical interaction points. That is, if the simulation PC gives **the same data to the physical-system at the same time as the real cyber-system**, there is no difference from the physical-system's view point. Inspired by this, we formally define our simulation problem as a job scheduling problem on the simulation PC with only the following constraints:

(1) **Physical-read constraint**: For any job J_{ij} that reads data from the physical-system, the simulation PC should schedule it later than its start time on the real cyber-system, i.e.,

$$t_{ij}^{\mathsf{S,sim}} \geq t_{ij}^{\mathsf{S,real}}. \tag{1}$$

If $t_{ij}^{\mathsf{S,sim}} \geq t_{ij}^{\mathsf{S,real}}$, at $t_{ij}^{\mathsf{S,sim}}$, the physical data read at $t_{ij}^{\mathsf{S,real}}$ is already logged in the simulation PC. Thus, the simulation PC can start J_{ij} at $t_{ij}^{\mathsf{S,sim}}$ with the same physical data used by the real J_{ij} at $t_{ij}^{\mathsf{S,real}}$, due to "Tagged Data Read".

(2) **Physical-write constraint**: For any job J_{ij} that writes data to the physical-system, the simulation PC should finish J_{ij}'s execution before its finish time on the real cyber-system, i.e.,

$$t_{ij}^{\mathsf{F,sim}} \leq t_{ij}^{\mathsf{F,real}}. \tag{2}$$

If $t_{ij}^{\mathsf{F,sim}} \leq t_{ij}^{\mathsf{F,real}}$, the simulation PC can hold the output data of the simulated J_{ij} and send it out to the physical-system at the same time as the real cyber-system, i.e., at $t_{ij}^{\mathsf{F,real}}$, due to "Delayed Data Write".

(3) **Producer-consumer constraint**: For any pair of jobs, $J_{i'j'}$ and J_{ij}, if $J_{i'j'}$ becomes a data producer of J_{ij} on the real cyber-system according to the "Most Recent Data Use" and "Entry Read and Exit Write", the simulation PC should finish $J_{i'j'}$ before starting J_{ij}, i.e.,

$$t_{i'j'}^{\mathsf{F,sim}} \leq t_{ij}^{\mathsf{S,sim}}. \tag{3}$$

If $t_{i'j'}^{\mathsf{F,sim}} \leq t_{ij}^{\mathsf{S,sim}}$, the simulation PC can execute J_{ij} with the output data from $J_{i'j'}$ due to "Tagged Data Read". This ensures that the simulated J_{ij} uses the same data as the real J_{ij}.

If the simulation PC can schedule the simulated jobs meeting all the above constraints, all the simulated jobs can be executed with the same data as the real jobs and eventually can write the same physical data at the same time as the real cyber-system.

III. Proposed Approach

For the simulation PC to schedule jobs meeting the above three types of constraints, one challenge is that $t_{ij}^{\mathsf{S,real}}$ and $t_{ij}^{\mathsf{F,real}}$ are non-deterministic due to varying execution times of jobs. Thus, the producer-consumer relations among jobs are also non-deterministic.

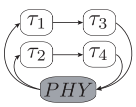
To tackle this challenge, we take a two-step approach: (1) in the offline phase, we construct a job-level precedence graph, so called an *offline guider*, which represents the aforementioned constraints in non-deterministic forms and (2) in the online phase, we schedule jobs on the simulation PC guided by the offline guider while resolving the non-determinism as we progress the scheduling, which we call *online progressive scheduling*.

A. Overview of our two-step approach

Offline guider: Let us use the example in Fig. 4(a) assuming the RM scheduling policy for each ECU. For all the jobs during one hyperperiod of the four tasks as in Fig. 4(b), we can compute their start time and finish time ranges by considering the RM scheduling with the best and worst case execution times. In Fig. 4(b), each up-arrow represents the release time of each job and each box represents the possible execution window of each job. Each job's start time and finish time ranges are represented by the double-headed arrows at the upper left corner and at the lower right corner of each box, respectively.

Considering these start time and finish time ranges of all the jobs, we construct the offline guider by adding precedence edges among them. In the initial state of Fig. 4(c), we have only one deterministic precedence edge from J_{11} to J_{12} since they are consecutive jobs of the same task τ_1. Then, we consider each job one by one to see whether it has the physical-read, physical-write, and producer-consumer constraints. For example, J_{21} has only a physical-read constraint as marked by R. For satisfying its physical-read constraint in Eq. (1), we have to know $t_{21}^{\mathsf{S,real}}$ to ensure that the simulation PC starts J_{21} later than $t_{21}^{\mathsf{S,real}}$. For this, we have to know the real execution time of J_{11} because J_{11} is a higher priority job on the same ECU. Thus, we set J_{11} as a deterministic predecessor of J_{21} as shown Fig. 4(c). This can guide the simulation PC to simulate J_{11} prior to J_{21} so that it can know e_{11}^{sim}, $e_{11}^{\mathsf{real}} = M_1(e_{11}^{\mathsf{sim}})$, and in turn $t_{21}^{\mathsf{S,real}}$.

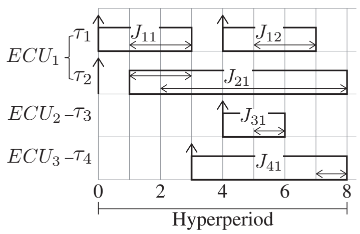
For another example, J_{41} has a physical-write constraint as marked by W and a producer-consumer constraint. Regarding the physical-write constraint in Eq. (2), we have to know $t_{41}^{\mathsf{F,real}}$. For this, it is enough to simulate just J_{41} and hence no precedence edge is added. However, regarding the producer-consumer constraint, it is not clear which job of τ_2 will be the data producer job of J_{41} since J_{21}'s finish time range overlaps J_{41}'s start time 3 in Fig. 4(b). If $t_{21}^{\mathsf{F,real}} \leq 3$, J_{21} will be the data producer job of J_{41}. Otherwise, J_{21}'s previous job (not shown in this simplified figure) will be the data producer. Thus, we have to know $t_{21}^{\mathsf{F,real}}$ before simulating J_{41}. Thus, jobs that can affect $t_{21}^{\mathsf{F,real}}$, i.e., J_{11}, J_{12}, and J_{21} in the example, become predecessors of J_{41} in Fig. 4(c). However, J_{12} and J_{21} become non-deterministic predecessors of J_{41}, i.e., dotted arrows, due to the following reason; After simulating only J_{11}, if $e_{11}^{\mathsf{real}} = M_1(e_{11}^{\mathsf{sim}})$ turns out to be 3, from Fig. 4(b), it is already clear that J_{21} cannot be the data producer of J_{41}. In that case, we do not have to simulate J_{12} and J_{21} prior to J_{41}.



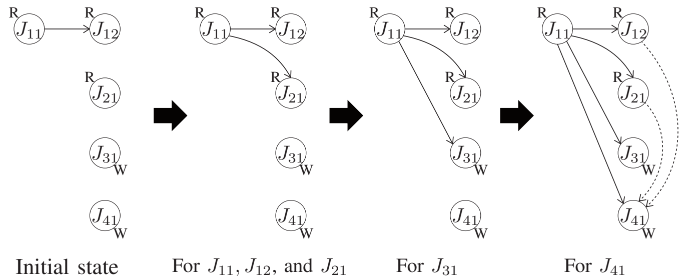
Task parameters

		Φ_i	P_i	C_i^{best}	C_i^{worst}
ECU_1	τ_1	0	4	1	3
	τ_2	0	8	1	2
ECU_2	τ_3	4	8	1	2
ECU_3	τ_4	3	8	4	5

(a) Control system and task parameters

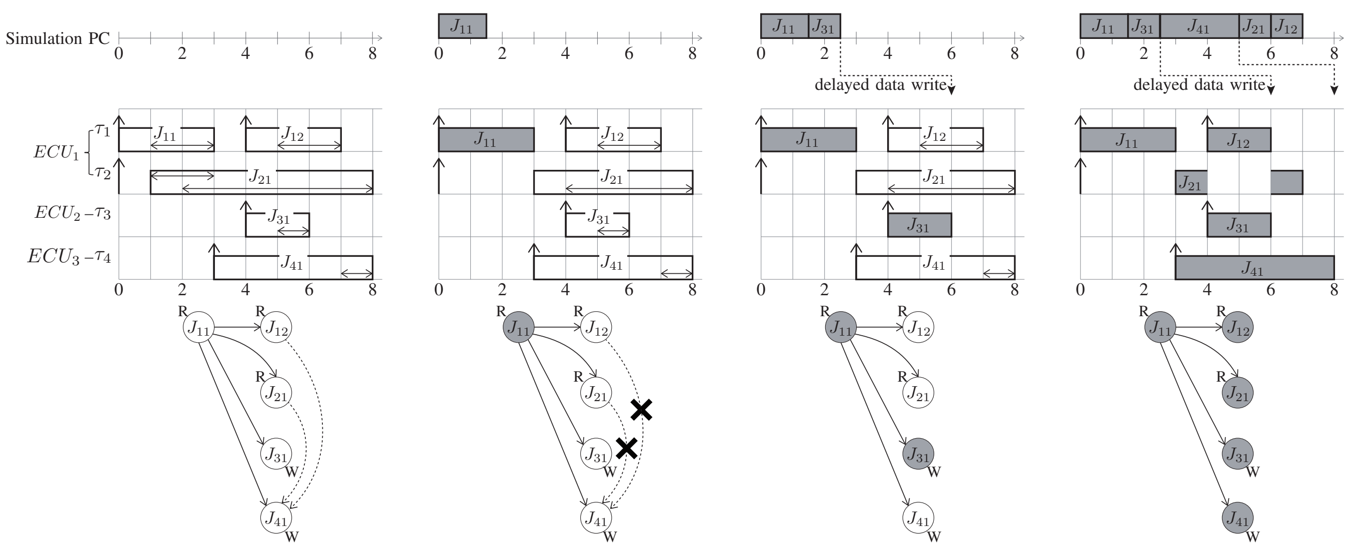


(b) Execution windows



(c) Construction of the offline guider

Fig. 4: Offline guider



(a) Initial state (b) After simulating J_{11} (c) After simulating J_{31} (d) Progress of simulation

Fig. 5: Online progressive scheduling

Online progressive scheduling: We overview our online progressive scheduling with an example in Fig. 5 assuming $e_{ij}^{\text{real}} = 2 \cdot e_{ij}^{\text{sim}}$. In the beginning, the simulation PC starts with the offline guider as in Fig. 5(a). At time 0, since J_{11} is the only job with no deterministic predecessor and its physical-read constraint is met, the simulation PC simulates J_{11} as in Fig. 5(b). Let us assume that $e_{11}^{\text{sim}} = 1.5$ and hence $e_{11}^{\text{real}} = 3$. Using $e_{11}^{\text{real}} = 3$, it can recalculate J_{21}'s finish time range as in Fig. 5(b). Then, it clearly knows J_{21} cannot be the data producer of J_{41}. Thus, it deletes the non-deterministic precedence edges from J_{12} and J_{21} to J_{41}. At time 1.5 on the simulation PC, there are four jobs with no unexecuted deterministic predecessors, i.e., J_{12}, J_{21}, J_{31}, and J_{41}. However, J_{12} and J_{21} do not meet the physical-read constraint yet. Thus, only J_{31} and J_{41} are ready for simulation. Out of them, J_{31} has an earlier deadline than J_{41}, i.e., $t_{31}^{\text{F,real}} < t_{41}^{\text{F,real}}$, due to the physical-write constraints. Thus, the simulation PC selects J_{31} as the next simulation job as in Fig. 5(c). After completing J_{31} at 2.5, the simulation PC knows $e_{31}^{\text{sim}} = 1$ and hence $e_{31}^{\text{real}} = 2$. Using $e_{31}^{\text{real}} = 2$, it computes the schedule on ECU_2 and knows the real finish time of J_{31} is 6 as in Fig. 5(c). Thus, it plans the delayed data write operation that will happen at 6. Like this, it progresses simulating jobs while resolving non-determinism guided by the offline guider.

B. Construction of the offline guider with non-determinism

For constructing the offline guider, we consider only jobs in one hyperperiod, i.e., HP. During one HP, a task τ_i has $n_i = HP/P_i$ jobs. For those jobs in our interested HP, we index them as $J_{i1}, J_{i2}, \cdots, J_{in_i}$ while we index those in the previous HP as $J_{i(-(n_i-1))}, \cdots, J_{i(-1)}, J_{i0}$ and those in the next HP as $J_{i(n_i+1)}, J_{i(n_i+2)}, \cdots, J_{i(2n_i)}$, and so on. The offline guider designates predecessors only for $J_{i1}, J_{i2}, \cdots, J_{in_i}$. This is enough for simulating jobs across multiple HPs, since the proper job index can simply be computed by the modulo n_i operation.

Here, without loss of generality, we use a single job $J_{ij} \in \{J_{i1}, J_{i2}, \cdots, J_{in_i}\}$ to explain how to determine its predecessors. For J_{ij}, we first add its previous job $J_{i(j-1)}$ as its deterministic predecessor, since control task usually uses the context of its previous instance for the computation of the next instance and hence the simulation PC also has to finish $J_{i(j-1)}$ before starting J_{ij}.

Then, we check what type of constraints J_{ij} has. If J_{ij} has a physical-read constraint, i.e., Eq. (1), we have to know $t_{ij}^{\text{S,real}}$ in order to ensure that the simulation PC starts J_{ij} after $t_{ij}^{\text{S,real}}$, i.e., $t_{ij}^{\text{S,sim}} \geq t_{ij}^{\text{S,real}}$. For this, we have to identify jobs that can affect the start of J_{ij} on the real cyber-system. Those jobs are the higher priority jobs on the same ECU that are released during the last busy period of J_{ij} on the real cyber-system. The last busy period is defined as the last time duration before J_{ij}'s release for which the processor is executing J_{ij} or its higher priority jobs [10]. For this, we compute the last worst case busy period $WCBP(J_{ij}) = [WCBP_{ij}^{\text{start, real}}, WCBP_{ij}^{\text{end, real}}]$ of J_{ij} using the worst case execution times for J_{ij} and all its higher priority jobs. In addition, we also compute the start time range $[\min(t_{ij}^{\text{S,real}}), \max(t_{ij}^{\text{S,real}})]$ of J_{ij} on the real cyber-system, where $\min(t_{ij}^{\text{S,real}})$ and $\max(t_{ij}^{\text{S,real}})$ can be computed using the best and worst case execution times, respectively, for all J_{ij}'s higher priority jobs. Similarly, we can compute the start time ranges of other jobs as well.

From the last worst case busy period $WCBP(J_{ij})$ and the start time range $[\min(t_{ij}^{\text{S,real}}), \max(t_{ij}^{\text{S,real}})]$ of J_{ij}, we can conservatively compute the set of jobs that can possibly affect $t_{ij}^{\text{S,real}}$ as follows:

$$\begin{aligned}\mathbb{J}^{\text{S}}(J_{ij}) =& \{J_{kl} | J_{kl} \in \mathbb{J}^{\text{high}}(J_{ij}),\\ & WCBP_{ij}^{\text{start,real}} \leq t_{kl}^{\text{R,real}} < \max(t_{ij}^{\text{S,real}})\}\end{aligned} \tag{4}$$

where $\mathbb{J}^{\text{high}}(J_{ij})$ denotes a set of higher priority jobs of J_{ij} on the same ECU. This equation means that any higher priority job J_{kl} whose release time $t_{kl}^{\text{R,real}}$ is after $WCBP_{ij}^{\text{start,real}}$ but before J_{ij}'s latest start time $\max(t_{ij}^{\text{S,real}})$ has potential to affect the start of J_{ij}. Thus, the set of such jobs, i.e., $\mathbb{J}^{\text{S}}(J_{ij})$, is called the "start time set" of J_{ij}.

Out of the jobs in $\mathbb{J}^{\text{S}}(J_{ij})$, the jobs whose latest start times are before the earliest start time of J_{ij} definitely affect the start of J_{ij} in the real cyber-system. Thus, the simulation PC should definitely execute them before J_{ij} to know their simulated execution times, their mapped real execution times, and in turn $t_{ij}^{\text{S,real}}$. Therefore, the jobs in the following set

$$\mathbb{J}^{\text{S-det}}(J_{ij}) = \{J_{kl} | J_{kl} \in \mathbb{J}^{\text{S}}(J_{ij}), \max(t_{kl}^{\text{S,real}}) < \min(t_{ij}^{\text{S,real}})\} \tag{5}$$

are designated as deterministic predecessors of J_{ij}.

On the other hand, other jobs in $\mathbb{J}^{\text{S}}(J_{ij})$ may or may not actually affect $t_{ij}^{\text{S,real}}$ in the real cyber-system depending on the real execution times of jobs in $\mathbb{J}^{\text{S-det}}(J_{ij})$. Thus, the jobs in the following set

$$\mathbb{J}^{\text{S-nodet}}(J_{ij}) = \mathbb{J}^{\text{S}}(J_{ij}) - \mathbb{J}^{\text{S-det}}(J_{ij}) \tag{6}$$

are designated as non-deterministic predecessors of J_{ij}.

If J_{ij} has a physical-write constraint, i.e., Eq. (2), we have to know $t_{ij}^{\text{F,real}}$ in order to ensure that the simulation PC finishes J_{ij} before $t_{ij}^{\text{F,real}}$, i.e., $t_{ij}^{\text{F,sim}} \leq t_{ij}^{\text{F,real}}$. Similarly to the case of physical-read constraint, we can compute the finish time range $[\min(t_{ij}^{\text{F,real}}), \max(t_{ij}^{\text{F,real}})]$ of J_{ij}, where $\min(t_{ij}^{\text{F,real}})$ and $\max(t_{ij}^{\text{F,real}})$ can be computed using the best and worst case execution times, respectively, for J_{ij} and all its higher priority jobs. Note that any higher priority job J_{kl} with release time $t_{kl}^{\text{R,real}}$ after $WCBP_{ij}^{\text{start,real}}$ but before J_{ij}'s latest finish time $\max(t_{ij}^{\text{F,real}})$ has potential to affect the finish time $t_{ij}^{\text{F,real}}$ of J_{ij}. Also, J_{ij} itself affects $t_{ij}^{\text{F,real}}$. Thus, a conservative set of jobs to be executed by the simulation PC to know $t_{ij}^{\text{F,real}}$ is given as follows:

$$\begin{aligned}\mathbb{J}^{\text{F}}(J_{ij}) =& \{J_{kl} | J_{kl} \in \mathbb{J}^{\text{high}}(J_{ij}),\\ & WCBP_{ij}^{\text{start,real}} \leq t_{kl}^{\text{R,real}} < \max(t_{ij}^{\text{F,real}})\} \cup \{J_{ij}\}.\end{aligned} \tag{7}$$

This set is called "finish time set" of J_{ij}. Unlike the case of physical-read constraint, the simulation PC can start J_{ij} without knowing $t_{ij}^{\text{F,real}}$ as long as it can finish J_{ij} before $t_{ij}^{\text{F,real}}$. Thus, the jobs in $\mathbb{J}^{\text{F}}(J_{ij})$ do not need to be predecessors of J_{ij}. Instead, we introduce J_{ij}'s terminal node denoted by $\hat{J}_{ij}$ (not shown in the simplified figure of Fig. 4(c)) with zero execution time and designate all the jobs in $\mathbb{J}^{\text{F}}(J_{ij})$ as predecessors of $\hat{J}_{ij}$. This way, it is enough to know $t_{ij}^{\text{F,real}}$ before starting zero execution time job $\hat{J}_{ij}$. If the simulation PC can start and also finish $\hat{J}_{ij}$ before $t_{ij}^{\text{F,real}}$, it means that the simulation PC finishes J_{ij} before $t_{ij}^{\text{F,real}}$ meeting J_{ij}'s physical-write constraint.

Out of the jobs in $\mathbb{J}^{\text{F}}(J_{ij})$, the jobs whose latest start times are before the earliest finish time of J_{ij} definitely need to be executed to know $t_{ij}^{\text{F,real}}$. Therefore, the jobs in the following set

$$\mathbb{J}^{\text{F-det}}(\hat{J}_{ij}) = \{J_{kl} | J_{kl} \in \mathbb{J}^{\text{F}}(J_{ij}), \max(t_{kl}^{\text{S,real}}) < \min(t_{ij}^{\text{F,real}})\} \tag{8}$$

are designated as deterministic predecessors of J_{ij}'s terminal node $\hat{J}_{ij}$.

On the other hand, other jobs in $\mathbb{J}^{\text{F}}(J_{ij})$ may or may not actually affect $t_{ij}^{\text{F,real}}$ depending on the real execution times of jobs in $\mathbb{J}^{\text{F-det}}(\hat{J}_{ij})$. Thus, the jobs in the following set

$$\mathbb{J}^{\text{F-nodet}}(\hat{J}_{ij}) = \mathbb{J}^{\text{F}}(J_{ij}) - \mathbb{J}^{\text{F-det}}(\hat{J}_{ij}) \tag{9}$$

are designated as non-deterministic predecessors of J_{ij}'s terminal node $\hat{J}_{ij}$.

Lastly, if J_{ij} has a producer-consumer constraint, i.e, Eq. (3), due to the data producer-consumer relation, i.e., $\tau_{i'} \rightarrow \tau_i$, we have to know which job $J_{i'j'}$ of $\tau_{i'}$ becomes the data producer job of J_{ij} in order to ensure that the simulation PC finishes $J_{i'j'}$ before starting J_{ij}, i.e., $t_{i'j'}^{\text{F,sim}} \leq t_{ij}^{\text{S,sim}}$. However, we cannot deterministically determine the data producer job due to the non-determinism of $J_{i'j'}$'s finish time and J_{ij}'s start time. For example, consider two tasks $\tau_{i'}$ and τ_i in Fig. 6 where $\tau_{i'} \rightarrow \tau_i$. The figure shows the start time and finish time ranges of three jobs $J_{i'(j'-1)}$, $J_{i'j'}$, $J_{i'(j'+1)}$ of $\tau_{i'}$, and our target job J_{ij} of τ_i. Note that the finish time ranges of $J_{i'j'}$ and $J_{i'(j'+1)}$ overlap with J_{ij}'s start time range. Those jobs are called "potential producers" of J_{ij}. If the real finish times of $J_{i'j'}$, $J_{i'(j'+1)}$ and the start time of J_{ij} are as marked by "1", $J_{i(j'-1)}$ is the producer job of J_{ij} according to the most recent data use property. On the other hand, if they are as marked by "2" or "3", $J_{i'j'}$ or $J_{i'(j'+1)}$ become the producer job, respectively.

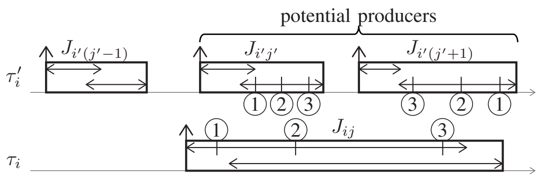


Fig. 6: Potential producers

Thus, in order to determine J_{ij}'s producer job, we have to know the real finish times of potential producers and the real start time of J_{ij}. Therefore, a conservative set of jobs to be simulated prior to J_{ij} is the union of finish time sets of the potential producers and the start time set of J_{ij} as follows:

$$\mathbb{J}^{\mathsf{P}}(J_{ij}) = \left(\bigcup_{\forall \text{potential producers } J_{i'j'}\text{s}} \mathbb{J}^{\mathsf{F}}(J_{i'j'}) \right) \cup \mathbb{J}^{\mathsf{S}}(J_{ij}) \tag{10}$$

Out of the jobs in $\mathbb{J}^{\mathsf{P}}(J_{ij})$, the jobs whose latest start times are before J_{ij}'s earliest start time definitely need to be executed by the simulation PC to determine the producer job of J_{ij}. Thus, they are designated as deterministic predecessors of J_{ij} as follows:

$$\begin{aligned} \mathbb{J}^{\mathsf{P\text{-}det}}(J_{ij}) =& \{J_{kl} | J_{kl} \in \mathbb{J}^{\mathsf{P}}(J_{ij}), \max(t_{kl}^{\mathsf{S,real}}) < \min(t_{ij}^{\mathsf{S,real}})\} \\ & \cup \{J_{i'(j'-1)}\} \\ \text{where } & \max(t_{i'(j'-1)}^{\mathsf{F,real}}) \le \min(t_{ij}^{\mathsf{S,real}}) < \max(t_{i'j'}^{\mathsf{F,real}}). \end{aligned} \tag{11}$$

In this equation, we designate $J_{i'(j'-1)}$ as a deterministic predecessor of J_{ij}. This is because $J_{i'(j'-1)}$ is the last job just before the potential producers and hence it becomes J_{ij}'s producer job if all potential producers' real finish times turn out to be later than J_{ij}'s real start time.

Other jobs in $\mathbb{J}^{\mathsf{P}}(J_{ij})$ may or may not need to be executed by the simulation PC to determine the producer job of J_{ij}. Thus, they are designated as non-deterministic predecessors of J_{ij} as follows:

$$\mathbb{J}^{\mathsf{P\text{-}nodet}}(J_{ij}) = \mathbb{J}^{\mathsf{P}}(J_{ij}) - \mathbb{J}^{\mathsf{P\text{-}det}}(J_{ij}). \tag{12}$$

C. Online progressive scheduling of simulated jobs

To schedule simulated jobs, our online algorithm dynamically manages an online job-level precedence graph, called *OJPG*, guided by the offline guider. In the beginning, we first initialize *OJPG* as follows. From the offline guider, all the jobs with positive job indexes are copied to *OJPG*. The copied jobs are those in the first HP, i.e., $J_{i1}, J_{i2}, \cdots, J_{in_i}$ for each τ_i. Also, all the associated deterministic and non-deterministic precedence edges in the offline guider are copied to *OJPG*. We also compute the start time range $[\min(t_{ij}^{\mathsf{S,real}}), \max(t_{ij}^{\mathsf{S,real}})]$ and finish time range $[\min(t_{ij}^{\mathsf{F,real}}), \max(t_{ij}^{\mathsf{F,real}})]$ of every job $J_{ij} \in OJPG$ considering the best case and the worst case execution times of all the jobs. All the data contents are initialized as their default values of the real-cyber system for the first executing jobs that use the data contents.

With such initialized *OJPG*, our online algorithm dynamically manages *OJPG* and schedules simulated jobs as follows. Our online algorithm considers a job J_{ij} in *OJPG* with no uncompleted deterministic predecessors. If J_{ij} does not have a physical-read constraint, we add it into the simulation ready queue. If it does have a physical-read constraint, we consider J_{ij}'s real start time, i.e, $t_{ij}^{\mathsf{S,real}}$. Since we can know $t_{ij}^{\mathsf{S,real}}$ for any job J_{ij} with no uncompleted deterministic predecessors (Lemma 2 in Appendix A), we add J_{ij} to the simulation ready queue at $t_{ij}^{\mathsf{S,real}}$, so that the simulation PC can start J_{ij} after $t_{ij}^{\mathsf{S,real}}$ satisfying the physical-read constraint, i.e., $t_{ij}^{\mathsf{S,sim}} \ge t_{ij}^{\mathsf{S,real}}$ in Eq. (1).

Out of the jobs in the simulation ready queue, the simulation PC schedules one of them based on the preemptive EDF policy. Note that the preemptive EDF scheduling of jobs based on their "effective deadlines" is an optimal scheduling approach for scheduling jobs with precedence constraints on a single processor [11]. Thus, we assign the effective deadline $t_{ij}^{\mathsf{D,sim}}$ to each job $J_{ij} \in OJPG$ as follows:

$$t_{ij}^{\mathsf{D,sim}} = \begin{cases} \min(t_{ij}^{\mathsf{F,real}}) & \text{for } \hat{J}_{ij} \\ \infty & \text{otherwise} \end{cases}, \tag{13}$$

$$t_{ij}^{\mathsf{D,sim}} = \min\left(t_{ij}^{\mathsf{D,sim}}, \min_{\forall J_{kl} \in \mathbb{J}^{\mathsf{succ\text{-}det}}(J_{ij})} \left(t_{kl}^{\mathsf{D,sim}} \right) \right) \tag{14}$$

where $\mathbb{J}^{\mathsf{succ\text{-}det}}(J_{ij})$ is the set of deterministic successors of J_{ij}. In Eq. (13), we first initialize the deadline of each job in *OJPG*. For a terminal node $\hat{J}_{ij}$ of J_{ij} with a physical-write constraint, we initialize its deadline as $\min(t_{ij}^{\mathsf{F,real}})$ because the simulation PC has to finish J_{ij} before $t_{ij}^{\mathsf{F,real}}$ to meet its physical-write constraint. For other jobs, we initialize their deadlines as ∞. Then, in Eq. (14), we backtrace jobs along the deterministic precedence edges and set each job J_{ij}'s deadline $t_{ij}^{\mathsf{D,sim}}$ as the minimum of its deterministic successors' deadlines.

At the time when the simulation PC is about to start a job J_{ij} with a physical-read constraint, we already know $t_{ij}^{\mathsf{S,real}}$ (Lemma 2 in Appendix A) and the current time is already later than $t_{ij}^{\mathsf{S,real}}$. Thus, the simulation PC starts J_{ij} with the proper time-tagged physical data. Also, at that time, all the data producer jobs $J_{i'j'}$s of J_{ij} are determined and they have finished (Lemma 4 in Appendix A). Thus, the simulation PC starts J_{ij} with the proper producer-tagged data if it is a data consumer job.

After finishing a job J_{ij} on the simulation PC, we add to *OJPG* J_{ij}'s corresponding new job $J_{i(j+n_i)}$ in the next HP together with deterministic and non-deterministic precedence edges from other jobs in *OJPG* guided by the offline guider. This makes the simulation continue across multiple HPs.

In addition, we now know J_{ij}'s execution time e_{ij}^{sim} on the simulation PC and hence we can estimate its real execution time e_{ij}^{real} on the ECU. Using e_{ij}^{real}, for every job J_{ab} in *OJPG* whose start and finish times are affected by J_{ij}'s execution time, we can update its start time range and finish time range. Specifically, for J_{ab}, by replacing C_i^{best} and C_i^{worst} used in the previous computation with e_{ij}^{real}, we can compute a narrowed start time range $[\min(t_{ab}^{\mathsf{S,real}}), \max(t_{ab}^{\mathsf{S,real}})]$ and a narrowed finish time range $[\min(t_{ab}^{\mathsf{F,real}}), \max(t_{ab}^{\mathsf{F,real}})]$.

Using these updated ranges, our online algorithm resolves the non-deterministic precedence edges in the *OJPG*. For the case where an edge from J_{kl} to J_{ij} in *OJPG* is declared non-deterministic since we were not sure whether $t_{kl}^{\mathsf{S,real}}$ is earlier than $t_{ij}^{\mathsf{S,real}}$, that is, the condition $\max(t_{kl}^{\mathsf{S,real}}) < \min(t_{ij}^{\mathsf{S,real}})$ in Eqs. (5) and (11) was not met, our online algorithm checks the condition again with the updated $\max(t_{kl}^{\mathsf{S,real}})$ and $\min(t_{ij}^{\mathsf{S,real}})$. If it turns out that

$$\max(t_{kl}^{\mathsf{S,real}}) < \min(t_{ij}^{\mathsf{S,real}}), \tag{15}$$

the non-deterministic edge from J_{kl} to J_{ij} becomes a deterministic one. On the other hand, with the updated $\min(t_{kl}^{\mathsf{S,real}})$ and $\max(t_{ij}^{\mathsf{S,real}})$, if it turns out that

$$\min(t_{kl}^{\mathsf{S,real}}) \geq \max(t_{ij}^{\mathsf{S,real}}), \tag{16}$$

it is clear that $t_{kl}^{\mathsf{S,real}}$ cannot be earlier than $t_{ij}^{\mathsf{S,real}}$. Thus, we remove the non-deterministic edge from J_{kl} to J_{ij}. Otherwise, the edge remains as a non-deterministic one until either it becomes deterministic or it is removed as progressing the online scheduling algorithm.

For the case where an edge from J_{kl} to the terminal node $\hat{J}_{ij}$ of J_{ij} in *OJPG* is declared non-deterministic since we were not sure whether $t_{kl}^{\mathsf{S,real}}$ is earlier than $t_{ij}^{\mathsf{F,real}}$, that is, the condition $\max(t_{kl}^{\mathsf{S,real}}) < \min(t_{ij}^{\mathsf{F,real}})$ in Eq. (8) was not met, our online algorithm checks the condition again with the updated $\max(t_{kl}^{\mathsf{S,real}})$ and $\min(t_{ij}^{\mathsf{F,real}})$. If it turns out that

$$\max(t_{kl}^{\mathsf{S,real}}) < \min(t_{ij}^{\mathsf{F,real}}), \tag{17}$$

the non-deterministic edge from J_{kl} to $\hat{J}_{ij}$ becomes a deterministic one. On the other hand, with the updated $\min(t_{kl}^{\mathsf{S,real}})$ and $\max(t_{ij}^{\mathsf{F,real}})$, if it turns out that

$$\min(t_{kl}^{\mathsf{S,real}}) \geq \max(t_{ij}^{\mathsf{F,real}}), \tag{18}$$

it is clear that $t_{kl}^{\mathsf{S,real}}$ cannot be earlier than $t_{ij}^{\mathsf{F,real}}$. Thus, we remove the non-deterministic edge from J_{kl} to J_{ij}. Otherwise, the edge remains as non-deterministic until the non-determinism is resolved.

Our online algorithm also updates the effective deadlines of jobs in *OJPG* by using the updated $\min(t_{ij}^{\mathsf{F,real}})$s for $\hat{J}_{ij}$ in Eq. (13) and newly changed deterministic edges in Eq. (14).

In summary, our online algorithm continues this process, i.e., (1) executing the job with the earliest effective deadline in the simulation ready queue, (2) adding a new job to *OJPG* for the next HP, (3) updating start time and finish time ranges, (4) resolving non-determinism, and (5) updating effective deadlines, until the simulation termination time.

This online progressive scheduling algorithm guided by the offline guider guarantees the functionally and temporally correct simulation if it can schedule all the jobs meeting their effective deadlines. This is theoretically proven in Appendix A.

The proposed approach is actually implemented as a scheduling engine of our simulation tool. Practical features for this implementation are discussed in Appendix B. For more details, interested readers are referred to our open-source project [12] and demo video clip [13].

IV. Evaluation

We first justify the mapping from the PC execution time to the ECU execution time, i.e., $e_{ij}^{\mathsf{real}} = M_i(e_{ij}^{\mathsf{sim}})$. In the design phase of the cyber-system, we can use the ECU EVB

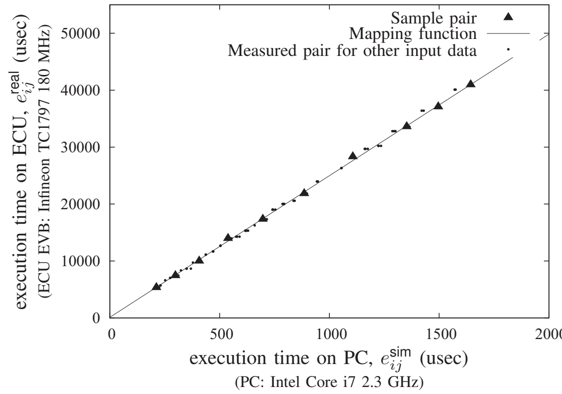


Fig. 7: Execution times for a matrix multiplication task

TABLE I: Statistics on e_{ij}^{real} estimation error

	Average	Standard deviation
Pulse code modulation	0.5845 (%)	0.9679 (%)
Data compression	1.9674 (%)	0.3191 (%)
Fast cosine transform	0.7289 (%)	0.9900 (%)
Image processing	2.8109 (%)	1.8581 (%)
Matrix multiplication	1.6882 (%)	1.3089 (%)

(evaluation board) to find the correlation between e_{ij}^{sim} and e_{ij}^{real}. The black triangles in Fig. 7 are 10 sample pairs of $(e_{ij}^{\mathsf{sim}}, e_{ij}^{\mathsf{real}})$ we measured for a matrix multiplication task. For the 10 sample pairs, by applying the linear regression, we can get the mapping function M_i as depicted by the solid line in the figure. The small dots in the figure are 100 measured pairs of $(e_{ij}^{\mathsf{sim}}, e_{ij}^{\mathsf{real}})$ for other input data. Their close placement on the mapping function implies that the mapping function can closely estimate e_{ij}^{real} from e_{ij}^{sim}. We conducted the same experiments for five example tasks. TABLE I shows reasonably small errors of e_{ij}^{real} estimation for the five tasks. More precise estimation of e_{ij}^{real} is beyond the scope of this paper. It is our future work.

Now, we evaluate the proposed approach using synthesized cyber-systems and also through actual implementation.

A. Evaluation using synthesized cyber-systems

In this subsection, we synthesize 1000 cyber-systems and evaluate the "simulatability", i.e., how many of them are correctly simulated.

Each cyber-system is synthesized as follows. The number of ECUs is determined from *uniform*[3,10]. The number of tasks on each ECU is determined from *uniform*[1,5]. Then, we form the data producer-consumer relations among all the tasks and the physical-system. Specifically,

- Each task becomes a data producer of *uniform*[0,2] randomly selected tasks.
- Physical-read ratio f_{PR}: Out of all the tasks, f_{PR}% randomly selected tasks read data from the physical system. $f_{PR} = 30\%$ if not otherwise mentioned.
- Physical-write ratio f_{PW}: Out of all the tasks, f_{PW}% randomly selected tasks write data to the physical system. $f_{PW} = 30\%$ if not otherwise mentioned.

For each task τ_i, its parameters are randomly generated as follows. Its period P_i is randomly generated from *uniform*[10 ms,100 ms] while the offset Φ_i is assumed zero. Then, the best case execution time C_i^{best} is randomly determined as *uniform*[5,10]% of P_i. From such determined C_i^{best}, the worst case execution time C_i^{worst} is determined by multiplying C_i^{best} and the "execution time variation factor" f_{var}, which is randomly selected from *uniform*[1.0,2.0] if not otherwise mentioned.

With such determined C_i^{best} and C_i^{worst}, the real execution time e_{ij}^{real} of each instance J_{ij} of τ_i on the real cyber-system is assumed to be one value from *uniform*[C_i^{best},C_i^{worst}]. Also, we assume the following simple mapping function between e_{ij}^{sim} and e_{ij}^{real}:

$$e_{ij}^{\text{real}} = M_i(e_{ij}^{\text{sim}}) = \frac{e_{ij}^{\text{sim}}}{0.3}.$$

For such a synthesized cyber-system, we perform simulation for ten HPs using the following four approaches:

- **Baseline**: This approach is trying to mimic the real cyber system as much as possible, that is, start each simulated job J_{ij} after its real start time $t_{ij}^{\text{S,real}}$ and keep the job simulation order the same as the job execution order in the real cyber-system.
- **TrueTime**: This is a real-time version of TrueTime [4]. TrueTime is originally designed to simulate jobs exactly following the job execution order in the real cyber-system aiming at offline simulation. From the original TrueTime, we make its real-time version by enforcing $t_{ij}^{\text{S,sim}} \geq t_{ij}^{\text{S,real}}$ only for the jobs J_{ij}s with physical-reads. For other jobs, this approach is free to start them earlier. In short, this approach enjoys the freedom of simulated job start times but not the freedom of simulated job execution order.
- **Ours**: This is our proposed approach that maximally enjoys both freedoms of simulated job start times and job execution order while satisfying only the constraints in Eqs. (1) and (3).
- **Ideal**: This is an ideal approach that assumes the real execution times of all the jobs are deterministically known. In this case, in the offline phase, we can deterministically compute the job schedule of the real cyber-system. Thus, we can draw a deterministic job-level precedence graph. For such a deterministic precedence graph, it is proven to be optimal to schedule jobs using EDF scheduling policy according to their effective release times and deadlines [11]. Thus, **Ideal** uses this optimal scheduling approach.

All the above four approaches guarantee that each simulated job uses the same physical data and producer data as the real jobs. Thus, if all the simulated jobs with physical-writes can be finished satisfying $t_{ij}^{\text{F,sim}} \leq t_{ij}^{\text{F,real}}$ in Eq. (2), they can write the same physical data at the same time as the real cyber-system using the "Delayed Data Write", which guarantees the simulation correctness. Thus, for the given synthesized cyber-system, if an approach can meet all of its finish time constraints, we count it as "simulatable" by the approach.

Fig. 8(a) compares the simulatabilities of the four approaches as changing the physical-read ratio f_{PR} from 0% to 100%. **Baseline** shows a poor simulatability in the whole range of f_{PR} since it simulates jobs without enjoying the start time freedom and the job execution order freedom. **TrueTime** enjoys the start time freedom for the jobs without physical-reads. Thus, it shows a bit better simulatability when the physical-read ratio is low. **Ours**, which enjoys both freedoms of start times and job execution order, shows a significantly higher simulatability in the whole range of f_{PR}. Especially, when f_{PR} is low, **Ours** can take full advantage of start time freedom and hence the improvement of simulatability is large. As f_{PR} increases, the benefit of start time freedom diminishes. Nevertheless, **Ours** still can enjoy the freedom of job execution order and hence it shows non-negligible improvement over **Baseline** and **TrueTime** even when f_{PR} is 100%. Another important observation is that **Ours**, which progressively resolves non-determinism, shows a comparable simulatability with **Ideal**, which ideally assumes everything is deterministic.

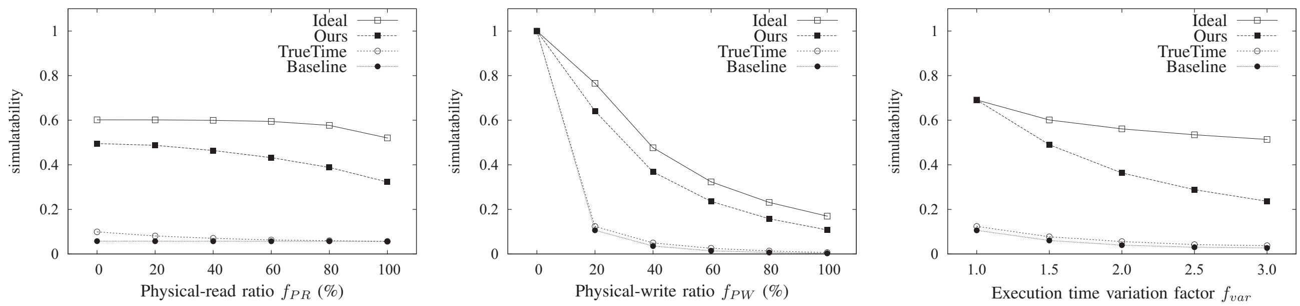
Fig. 8(b) compares the simulatabilities of the four approaches as changing the physical-write ratio f_{PW} from 0% to 100%. When f_{PW} = 0%, all the four approaches show the simulatabilities of one. This is because f_{PW} = 0% means that no tasks write data to the physical system and hence there is no real-time deadline before which simulated jobs should finish. As increasing f_{PW}, the simulatabilities of all the four approaches drop. Nevertheless, the simulatability of **Ours** stays significantly higher than **Baseline** and **TrueTime** in the whole range of f_{PW}. This is because **Ours** enjoys the freedom of job execution order and hence can schedule more urgent deadline jobs earlier than others without being restricted by the job execution order in the real cyber-system. Again, **Ours** shows a comparable simulatability with **Ideal**.

Fig. 8(c) compares the four approaches as changing the execution time variation factor f_{var} from 1.0 to 3.0. Due to the same reason, **Ours** shows significantly higher simulatability than **Baseline** and **TrueTime** in the whole range of f_{var}. Comparing **Ours** with **Ideal**, when $f_{var} = 1.0$, that is, when $C_i^{\text{best}} = C_i^{\text{worst}}$ for every τ_i, **Ours** shows the same simulatability as that of **Ideal**. This means that when there is no non-determinism in the job execution times, **Ours** performs exactly same as **Ideal** and shows the optimal performance. As increasing f_{var}, the cyber-system has increasing non-determinism. This is the reason for the increasing gap between **Ours** and **Ideal**.

B. Implementation

We actually implement our simulator using the proposed approach as an application of Linux on the simulation PC equipped with 2.3 GHz quadcore Intel Core i7 processor. Among the four cores, we dedicate two cores for our simulator: (1) the first one is always used for the simulator's main thread which runs our proposed scheduling algorithm completely isolated from other Linux applications and kernel and (2) the second core is always used for executing the simulated jobs by the commands of the main thread in the first core. It is our future work to improve the simulatability by using multiple cores for executing simulated jobs.

As an example cyber-system, we use an automotive control system composed of two ECUs connected by the TTCAN bus [7]. Each ECU is equipped with an Infineon TC1797 180 MHz microprocessor [5]. The first ECU performs a CC (Cruise-Control) function by executing two periodic tasks τ_1

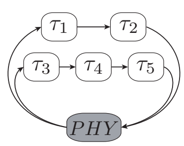


(a) Simulatability as changing the physical-read ratio f_{PR}

(b) Simulatability as changing the physical-write ratio f_{PW}

(c) Simulatability as changing the execution time variation factor f_{var}

Fig. 8: Simulatability comparison



(a) Control design

Unit: millisecond

		Φ_i	P_i	C_i^{best}	C_i^{worst}
ECU_1	τ_1	0	50	4.9	10.1
	τ_2	0	100	9.9	30.3
ECU_2	τ_3	0	100	9.8	20.2
	τ_4	0	200	19.8	30.3
	τ_5	0	400	49.4	202.8

(b) Task parameters

Fig. 9: Cyber-system to be simulated and implemented

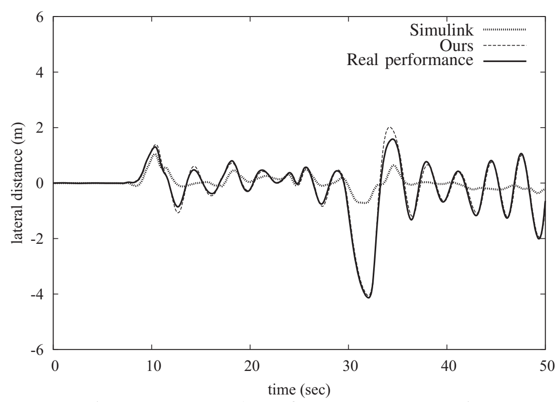
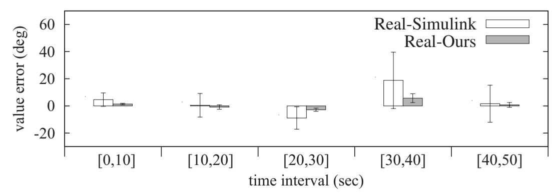


Fig. 10: Control performance comparison

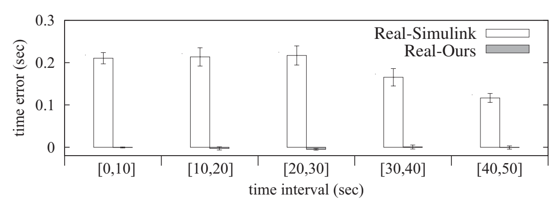
and τ_2. τ_1 reads the current speed of the vehicle and τ_2 calculates and writes the necessary brake or acceleration signal to keep the desired vehicle speed. The second ECU performs a LK (Lane-Keeping) function by executing three periodic tasks, τ_3, τ_4, and τ_5. τ_3 reads the front-view and τ_4 converts it to the lateral distance, that is, the vehicle's distance from the center of the lane. Then, τ_5 calculates and writes the necessary steering angle to keep the vehicle at the center of the lane. Thus, the task graph of such a control system is Fig. 9(a). The parameters of the five tasks are given in Fig. 9(b). The tasks on each ECU are scheduled by the Rate-Monotonic scheduling policy.

As the physical-system, we use CarSim RT [3], which is a commercial real-time vehicle dynamics simulator.

Fig. 10 shows three LK control performances, i.e., the three lateral distance curves, for 50 secs; the first one is estimated by the Simulink simulation, the second one is estimated by our simulation, and the third one is the real performance given by the real implemented cyber-system. The figure says that the



(a) Value error



(b) Time error

Fig. 11: Statistics on value errors and time errors

control performance estimated by the Simulink simulation has non-negligible differences from the real control performance at many time points. This is because the Simulink simulation does not correctly simulate the temporal behavior of the real cyber-system. On the other hand, our simulation shows control performance estimation very close to the real performance.

In order to more deeply investigate the simulation correctness in both functional and temporal aspects, Figs. 11(a) and (b) show statistics on the value errors and time errors, respectively, by Simulink and Ours compared with the real performance. In the figures, the 50 sec experimental duration of Fig. 10 is divided into 10 sec windows. For each 10 sec window, we collect value errors and time errors for the physical-write points and calculate the average and 90% confidence interval. Simulink shows large average errors in both value and time and also wide 90% confidence intervals. On the other hand, by our simulation, the average errors of value and time are almost zero in the all 10 sec windows. Also, the 90% confidence intervals are very narrow.

The reason for non-zero errors of our simulation is mainly due to imperfect mapping from the job execution time on the simulation PC to the one on the real ECU. Such imperfect mapping can cause slight differences of the physical-read time points, which make the simulated cyber-system read a bit different physical data and in turn write a bit different data to the physical system. If the execution time mapping were perfect, our simulation would give zero errors in both aspects of value and timing as theoretically proven in Appendix A.

V. Related Works

In the automotive industry, simulation tools such as Simulink [1] and LabVIEW [2] are widely used for the design time verification of the control algorithms. However, they do not correctly model the events such as varying execution times and task preemptions that will happen once the cyber-system is really implemented. Thus, their control performance predictions are far different from the real control performance. Moreover, they are aiming at offline simulation without real-time interaction with real working physical-systems.

For the real-time validation, the rapid prototyping of the cyber-system on AutoBox [14] is commonly used. However, AutoBox just provides fast executions of control algorithms for the interaction with real working physical systems. Autobox neither correctly models the events of the real cyber-system and hence it cannot provide functionally and temporally correct simulation. Other commercial tools advertising "real-time simulation", such as RT-Sim [15] and NI-HIL [16], have the same limitation.

For accurate modeling of events in the real cyber-system, we can think of an emulation approach such as cycle-accurate instruction set simulators [17], [18], [19]. However, they are too slow to emulate all the cyber-system including application tasks and operating systems of multiple ECUs. Recently, host-compiled simulation draws much attention due to its fast and time-accurate simulation [20], [21]. However, it is targeting non-real-time systems like a smart phone without interactions with a physical system. Targeting real-time systems, [22] proposes an RTOS emulator which executes jobs simulating RTOS scheduling events. However, the job execution time on the RTOS emulator is totally different from the real one and hence the resulting schedule on the emulator is different from the real schedule, which violates the functional/temporal simulation correctness.

TrueTime [4], chronSIM [23], and TA Simulator [24] have the most similar goal as ours, that is, they try to simulate jobs accurately modeling the scheduling events that will happen on multiple embedded processors. However, they try to exactly follow the job execution order that will happen in the real cyber-system. Thus, they cannot enjoy the freedom of job execution order as ours and hence the real-time simulatability is quite limited. Thus, they cannot provide real-time simulation interacting with real-working physical systems.

VI. Conclusion

This paper proposes a novel approach for functionally and temporally correct simulation of the cyber-side of an automotive system. The approach consists of two steps: (1) offline construction of a guider including non-determinism and (2) online scheduling progressively resolving non-determinism. It significantly improves the real-time simulatability by enjoying the freedom of job scheduling while respecting only the minimal set of constraints for the functional and temporal correctness. Its functional and temporal correctness is theoretically proven and also empirically validated through actual implementation.

In our future work, we plan to extend our approach so that we can utilize multicore to further improve the real-time simulatability. We also plan to study accurate mapping from PC execution times to ECU execution times. In the long term, we plan to make our approach as a general simulator applicable for a broader spectrum of CPSs.

Acknowledgment

This research was supported in part by SW Starlab Program (IITP-2015-000209) through IITP (Institute for Information & communications Technology Promotion) and in part by Next-Generation Information Computing Development Program (2017M3C4A7065925) through NRF (National Research Foundation of Korea), both funded by Ministry of Science and ICT. The Institute of Computer Technology at Seoul National University provides research facilities for this study. The corresponding author is Chang-Gun Lee.

References

[1] MathWorks Inc., *Simulink*. http://www.mathworks.com.
[2] National Instruments Corporation, *LabVIEW*. http://www.ni.com.
[3] Mechanical Simulation Corporation, *CarSim RT*. https://www.carsim.com.
[4] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K. -E. Arzen. *How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime*. IEEE Control Systems, 23(3):16–30, May 2003.
[5] Infineon Technologies AG, *TC1797*, http://www.infineon.com.
[6] NXP Semiconductors, *MPC5566*, http://www.nxp.com.
[7] T. Fuhrer, B. Muller, W. Dieterle, F. Hartwich, R. Hugel, and M. Walther *Time Triggered Communication on CAN (Time Triggered CAN-TTCAN)*. Robert Bosch GmbH, 2000.
[8] National Instruments Corporation, *FlexRay Automatic Communication Bus Overview*. http://www.ni.com.
[9] Intel Corporation, *Intel Core i5-7600 Processor*. https://www.intel.com.
[10] M. G. Harbour, M. H. Klein, and J. P. Lehoczky. *Timing analysis for fixed-priority scheduling of hard real-time systems*. IEEE Transactions on Software Engineering, 20(1):13–28, January 1994.
[11] M. Spuri and J. A. Stankovic. *How to integrate precedence constraints and shared resources in real-time scheduling*. IEEE Transactions on Computers, 43(12):1407–1412, December 1994.
[12] RUBIS Lab., *CPSim*. https://github.com/rubis-lab/CPSim_linux.
[13] RUBIS Lab., *A Real-Time Simulator for Automotive Systems*. http://rubis.snu.ac.kr/researches#25796.
[14] dSPACE GmbH, *AutoBox*. https://www.dspace.com.
[15] C. Dufour, C. Andreade, and J. Belaneger. *Real-Time Simulation Technologies in Education: a Link to Modern Engineering Methods and Practices*. Proc. of International Conference on Engineering and Technology Education (INTERTECH), March 2010.
[16] National Instruments Corporation, *NI HIL Simulator Reference System*, http://www.ni.com.
[17] J. Zhu and D. Gajski. *An Ultra-Fast Instruction Set Simulator*. IEEE Transactions on Very Large Scale Integration systems, 10(3):363–373, June 2002.
[18] F. Bellard. *QEMU, a Fast and Portable Dynamic Translator*. Proc. of Usenix Annual Technical Conference (ATEC), pp. 41–46, April 2005.
[19] Imperas Software, *Open Virtual Platforms*. http://www.ovpworld.org.
[20] D. Mueller-Gritschneder, K. Lu, and U. Schlichtmann. *Control-flow-driven Source Level Timing Annotation for Embedded Software Models on Transaction Level*. Proc. of Euromicro Conference on Digital System Design (DSD), pp. 600–607, August 2011.
[21] S. Roloff, F. Hannig, and J. Teich. *Fast Architecture Evaluation of Heterogeneous MPSoCs by Host-Compiled Simulation*. Proc. of International Workshop on Software and Compilers for Embedded Systems (SCOPES), pp. 52–61, May 2012.
[22] P. Razaghi and A. Gerstlauer. *Host-Compiled Multicore RTOS Simulator for Embedded Real-Time Software Development*. Proc. of Design, Automation and Test in Europe Conference (DATE), pp. 1–6, March 2011.
[23] INCHRON GmbH, *chronSIM*. https://www.inchron.com.
[24] Timing-Architects Embedded Systems GmbH, *TA Simulator*. https://www.timing-architects.com.

Appendix A: Proof of Functional and Temporal Correctness of Our Simulation

This section gives a formal proof on the functional and temporal correctness of our proposed simulation approach.

Lemma 1. *At any time point of our simulation, $OJPG$ does not have a directed cycle consisting of deterministic precedence edges.*

Proof. We prove this by contradiction. Suppose that $OJPG$ has a cycle consisting of deterministic precedence edges. Those deterministic edges are due to the physical-read constraints as in Eq. (5) and/or the producer-consumer constraints as in Eq. (11). This is because the physical-write constraint in Eq. (8) never makes a cycle since it makes incoming edges to a terminal node $\hat{J}_{ij}$, which never has outgoing edges to other jobs. In the cycle, let us consider a deterministic edge from J_{kl} to J_{ij}. That edge implies $\max(t_{kl}^{\mathsf{S,real}}) < \min(t_{ij}^{\mathsf{S,real}})$ due to Eqs. (5) and/or (11) in the offline guider construction and also Eq. (15) in the online resolution of non-determinism. Thus, it is clear that $t_{kl}^{\mathsf{S,real}} < t_{ij}^{\mathsf{S,real}}$. Also, in the cycle, there should be a path from J_{ij} to J_{kl} consisting of deterministic edges. It now implies $\max(t_{ij}^{\mathsf{S,real}}) < \min(t_{kl}^{\mathsf{S,real}})$ and hence $t_{ij}^{\mathsf{S,real}} < t_{kl}^{\mathsf{S,real}}$. It is a contradiction. □

Lemma 2. *At any time point of our simulation, for a job J_{ij} in $OJPG$ with a physical-read constraint, if all of its deterministic predecessors have completed, its start time $t_{ij}^{\mathsf{S,real}}$ on the real cyber-system is known.*

Proof. We prove this by contradiction. Suppose that there exists a job J_{ij} with a physical-read constraint whose deterministic predecessors have all completed but start time $t_{ij}^{\mathsf{S,real}}$ is still unknown. This means there are uncompleted jobs—jobs whose execution times are unknown— in $\mathbb{J}^{\mathsf{S}}(J_{ij})$ that would delay J_{ij}'s start. Among them, consider a job J_{kl} with the earliest $\min(t_{kl}^{\mathsf{S,real}})$. Since J_{kl} would delay J_{ij}'s start, its priority is higher than J_{ij} on the same ECU and its $\min(t_{kl}^{\mathsf{S,real}})$ is prior to $\min(t_{ij}^{\mathsf{S,real}})$. For such J_{kl}, if its start time $t_{kl}^{\mathsf{S,real}}$ is still unknown, this means that there is another uncompleted job J_{mn} in $\mathbb{J}^{\mathsf{S}}(J_{ij})$ who would delay J_{kl}'s start and hence $\min(t_{mn}^{\mathsf{S,real}}) < \min(t_{kl}^{\mathsf{S,real}})$. This contradicts the fact that J_{kl} is the uncompleted job with the earliest $\min(t_{kl}^{\mathsf{S,real}})$. Thus, $t_{kl}^{\mathsf{S,real}}$ should be known, i.e., $\min(t_{kl}^{\mathsf{S,real}}) = \max(t_{kl}^{\mathsf{S,real}})$. Then, $\min(t_{kl}^{\mathsf{S,real}}) = \max(t_{kl}^{\mathsf{S,real}}) < \min(t_{ij}^{\mathsf{S,real}})$ and hence the uncompleted job J_{kl} is a deterministic predecessor of J_{ij} due to the condition in Eq. (15). It is a contradiction. □

Lemma 3. *At any time point of our simulation, for a job J_{ij} in $OJPG$ with a physical-write constraint, if all of the deterministic predecessors of J_{ij}'s terminal node $\hat{J}_{ij}$ have completed, its finish time $t_{ij}^{\mathsf{F,real}}$ on the real cyber-system is known.*

Proof. We prove this by contradiction. Let us consider a job J_{ij} with a physical-write constraint. For J_{ij}, suppose that all the deterministic predecessors of its terminal node $\hat{J}_{ij}$ have completed but its finish time $t_{ij}^{\mathsf{F,real}}$ is still unknown. Then, there are uncompleted jobs—jobs whose execution times are unknown— in $\mathbb{J}^{\mathsf{F}}(J_{ij})$ that would delay J_{ij}'s finish. Among them, consider a job J_{kl} with the earliest $\min(t_{kl}^{\mathsf{S,real}})$. Since J_{kl} would delay J_{ij}'s finish, its priority is higher than J_{ij} on the same ECU and its $\min(t_{kl}^{\mathsf{S,real}})$ is prior to $\min(t_{ij}^{\mathsf{F,real}})$. For such J_{kl}, if its start time $t_{kl}^{\mathsf{S,real}}$ is still unknown, this means that there is another uncompleted job J_{mn} in $\mathbb{J}^{\mathsf{F}}(J_{ij})$ who would delay J_{kl}'s start and hence $\min(t_{mn}^{\mathsf{S,real}}) < \min(t_{kl}^{\mathsf{S,real}})$. This contradicts the fact that J_{kl} is the uncompleted job with the earliest $\min(t_{kl}^{\mathsf{S,real}})$. Thus, $t_{kl}^{\mathsf{S,real}}$ should be known, i.e., $\min(t_{kl}^{\mathsf{S,real}}) = \max(t_{kl}^{\mathsf{S,real}})$. Then, $\min(t_{kl}^{\mathsf{S,real}}) = \max(t_{kl}^{\mathsf{S,real}}) < \min(t_{ij}^{\mathsf{F,real}})$ and hence the uncompleted job J_{kl} is a deterministic predecessor of $\hat{J}_{ij}$ due to the condition in Eq. (17). It is a contradiction. □

Lemma 4. *At any time point of our simulation, for a job J_{ij} in $OJPG$ with a producer-consumer constraint due to $\tau_{i'} \to \tau_i$, if all of its deterministic predecessors have completed, J_{ij}'s data producer job $J_{i'j'}$ is determined and it has already completed.*

Proof. Recall Eq. (10) that says the set of jobs $\mathbb{J}^{\mathsf{P}}(J_{ij})$ to be executed by the simulation PC to know the producer job of J_{ij} is the union of finish time sets of the potential producers and the start time set of J_{ij}. **Step 1:** Since $\mathbb{J}^{\mathsf{P}}(J_{ij})$ includes the start time set $\mathbb{J}^{\mathsf{S}}(J_{ij})$ of J_{ij}, at the moment t when all of J_{ij}'s deterministic predecessors have completed, J_{ij}'s start time $t_{ij}^{\mathsf{S,real}}$ is known, i.e., $\min(t_{ij}^{\mathsf{S,real}}) = t_{ij}^{\mathsf{S,real}} = \max(t_{ij}^{\mathsf{S,real}})$, by Lemma 2. **Step 2:** Now, we prove that, at the moment t, for every potential producer $J_{i'j'}$ whose finish time $t_{i'j'}^{\mathsf{F,real}}$ will eventually turn out to be prior to $t_{ij}^{\mathsf{S,real}}$, its finish time $t_{i'j'}^{\mathsf{F,real}}$ is known as follows; (1) $\mathbb{J}^{\mathsf{P}}(J_{ij})$ includes the finish time set $\mathbb{J}^{\mathsf{F}}(J_{i'j'})$ of each potential producer $J_{i'j'}$ and all the jobs in $\mathbb{J}^{\mathsf{F}}(J_{i'j'})$ become deterministic or non-deterministic predecessors of J_{ij} by Eqs. (11) and (12). Thus, J_{ij} acts as the terminal node $\hat{J}_{i'j'}$ of $J_{i'j'}$. (2) The condition for a job J_{kl} in $\mathbb{J}^{\mathsf{P}}(J_{ij})$ to be a deterministic predecessor of J_{ij} is $\max(t_{kl}^{\mathsf{S,real}}) < \min(t_{ij}^{\mathsf{S,real}}) = t_{ij}^{\mathsf{S,real}}$ in Eq. (15). This condition is a necessary condition of the condition for a job J_{kl} in $\mathbb{J}^{\mathsf{F}}(J_{i'j'})$ to be a deterministic predecessor of $J_{i'j'}$'s terminal node $\hat{J}_{i'j'}$, i.e., $\max(t_{kl}^{\mathsf{S,real}}) < \min(t_{i'j'}^{\mathsf{F,real}})$ in Eq. (17), because $\min(t_{i'j'}^{\mathsf{F,real}}) \leq t_{i'j'}^{\mathsf{F,real}} \leq t_{ij}^{\mathsf{S,real}}$. (3) Thus, the fact that all the deterministic predecessors of J_{ij} have completed implies that all the deterministic predecessors of $J_{i'j'}$'s terminal node $\hat{J}_{i'j'}$ have completed. Thus, $t_{i'j'}^{\mathsf{F,real}}$ is known by Lemma 3. **Step 3:** Since we know J_{ij}'s start time $t_{ij}^{\mathsf{S,real}}$ and also we know every potential producer $J_{i'j'}$'s finish time $t_{i'j'}^{\mathsf{F,real}}$ if it is prior to $t_{ij}^{\mathsf{S,real}}$, we can determine the last $J_{i'j'}$ whose finish time $t_{i'j'}^{\mathsf{F,real}}$ is prior to J_{ij}'s start time $t_{ij}^{\mathsf{S,real}}$. That last $J_{i'j'}$ is the data producer job of J_{ij} and it has already completed. □

Theorem 1. *If our online progressive scheduling algorithm can schedule all the simulated jobs meeting their effective deadlines, it guarantees the functionally and temporally correct simulation.*

Proof. By Lemma 1, our algorithm can continue simulating jobs in *OJPG* without being stuck in a cycle. Also, by

Lemma 2 and Lemma 4, our algorithm can simulate each job with the same data as the real cyber-system, using the tagged data read. In addition, by Lemma 3, our online algorithm can assign the real finish time $t_{ij}^{\mathsf{F,real}}$ as the effective deadline of the the simulated job J_{ij} with physical-write, using Eqs. (13) and (14). Thus, if our online algorithm can finish J_{ij} before its assigned effective deadline, the simulation PC can write its output data at the same time as the real cyber-system, using the delayed data write. Therefore, the simulation PC using our online algorithm can write the same physical data at the same time as the real cyber-system. The theorem follows. □

Appendix B: Practical Features of Our Implemented Simulator

In this section, we explain practical features we employed for implementing our simulator so that it can be actually used in the automotive system development.

First, we address the issue of data exchange delay through the TDMA bus, which was assumed zero so far for the simplicity of explanation. Regarding the the TDMA bus, the general practice of the automotive industry is to assign a dedicated slot to every data content that needs to be exchanged among ECUs and the physical system [7]. The dedicated slots form a cycle-executive schedule and the cycle repeats. Thus, if we know that a job J_{ij} finishes and produces its data content at $t_{ij}^{\mathsf{F,real}}$, we can determine the wait time $\delta_{ij}^{\mathsf{F,real}}$ until the next dedicated slot. Thus, the time when the data content is actually received by the ECUs of its receiver jobs can be deterministically calculated as

$$R(t_{ij}^{\mathsf{F,real}}) = t_{ij}^{\mathsf{F,real}} + \delta_{ij}^{\mathsf{F,real}} + T_s \tag{19}$$

where T_s is the constant transmission time of one slot.

Using this deterministic calculation, our simulator still keeps the functional and temporal correctness as follows. Our simulation PC and the physical-system is actually connected through a real cable such as a CAN cable and a FlexRay cable. For this, we use a USB-CAN or USB-FlexRay adaptor for the PC side. In addition, on the simulation PC, we implement one TDMA bus read handler and one TDMA bus write handler. The TDMA bus read handler is invoked whenever a data content from the physical-system is received through its dedicated slot. Then, the TDMA bus read handler logs the data into the data buffer with the reception time tag. Thus, when the simulation PC starts a job J_{ij} satisfying the physical-read constraint, i.e, $t_{ij}^{\mathsf{S,sim}} \geq t_{ij}^{\mathsf{S,real}}$, in Eq. (1), it can select the correct time-tagged data from the buffer.

The TDMA bus write handler is invoked at the planned times to transmit data contents from the cyber-system to the physical system. For explaining this, let us consider a job J_{ij} completed by the simulation PC satisfying the physical-write constraint, i.e., $t_{ij}^{\mathsf{F,sim}} \leq t_{ij}^{\mathsf{F,real}}$, in Eq. (2). From the known $t_{ij}^{\mathsf{F,real}}$, the simulation PC can determine the slot time of the data transmission as $t_{ij}^{\mathsf{F,real}} + \delta_{ij}^{\mathsf{F,real}}$. Thus, the simulation PC plans the TDMA bus write handler so that it transmits the data using the slot at $t_{ij}^{\mathsf{F,real}} + \delta_{ij}^{\mathsf{F,real}}$. This way, the physical-system can receive the data at the same time, i.e., $R(t_{ij}^{\mathsf{F,real}}) = t_{ij}^{\mathsf{F,real}} + \delta_{ij}^{\mathsf{F,real}} + T_s$, as if it is transmitted from the real ECU of the real job J_{ij} that finishes at $t_{ij}^{\mathsf{F,real}}$.

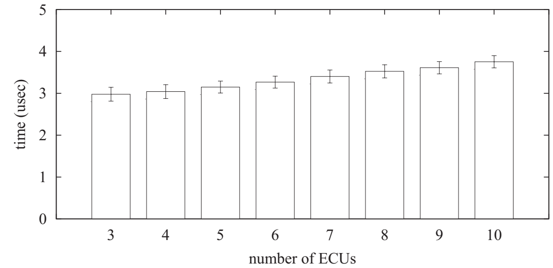


Fig. 12: Simulator's online overhead

For the data exchanges among ECUs simulated by the simulation PC, data transmissions do not actually happen on the TDMA bus. Instead, the simulation PC virtually considers TDMA bus transmission using the above calculation in Eq. (19). Specifically, regarding a producer-consumer relation from $\tau_{i'}$ running on one ECU to τ_i on another ECU, the simulation PC transforms the finish times $t_{i'j'}^{\mathsf{F,real}}$s of potential producer jobs to their corresponding reception times $R(t_{i'j'}^{\mathsf{F,real}})$s at τ_i's ECU. Such transformed finish times are used to determine J_{ij}'s producer job $J_{i'j'}$ as in Fig. 6. Once the producer job $J_{i'j'}$ is such correctly identified, the simulation PC can schedule $J_{i'j'}$ and J_{ij} satisfying $t_{i'j'}^{\mathsf{F,sim}} \leq t_{ij}^{\mathsf{S,sim}}$ in Eq. (3), and hence it can start J_{ij} at $t_{ij}^{\mathsf{S,sim}}$ with the correct data produced by $J_{i'j'}$ at its finish time $t_{i'j'}^{\mathsf{F,sim}}$.

Second, our simulator implements the proposed online scheduling algorithm in an optimized way to minimize its online overhead. Fig. 12 shows the average and 99% confidence interval of measured online overheads as increasing the number of ECUs. The overhead gradually increases as increasing the number of ECUs but stays under a few microseconds even for a large cyber-system consisting of 10 ECUs.

Furthermore, our simulator provides useful features for the actual design and implementation of the cyber-side of automotive systems. Here, we give just a brief introduction of them. Interested readers are referred to our open-source project [12] and demo video clip [13].

- Hybrid simulation: In the development process, some ECUs are implemented before others. Our simulator can support such a hybrid situation by simulating only unimplemented ECUs interacting with the real implemented ECUs and the real-working physical system.
- Static and dynamic memory analysis: Our simulator can analyze the amount of memory requirement due to the static memory such as codes and static variables and dynamic memory such as stack. This is a useful feature since engineers can predict whether the tasks can fit into the target ECU without memory overflow.
- Automatic generation of ECU executable: After verifying the correctness of the design, our simulator can automatically generate the ECU executable (ELF file) for each ECU by merging the microkernel codes and task codes together. The task codes may be hand-made C codes or MATLAB auto-generated C codes.