공학박사 학위논문

Functionally and Temporally Correct Simulation for Cyber-Physical Systems

사이버-물리 시스템을 위한 기능적/시간적 정확성 보장 시뮬레이션 기법

2017년 8월

서울대학교 대학원

전기 · 컴퓨터공학부

위 경 수

Abstract

Functionally and Temporally Correct Simulation for Cyber-Physical Systems

Kyoung-Soo We
Department of Electrical Engineering and Computer Science
The Graduate School
Seoul National University

When developing a Cyber-Physical System (CPS), simulators are commonly used to predict the final performance of the system at the design phase. However, current simulation tools 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, and this leads to enormous time and cost for a system development, because multiple re-design and re-implementation phases are required, until an acceptable system configuration is determined. Motivated by this limitation, this dissertation proposes functionally and temporally correct simulation for the cyber-side of a CPS. 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. 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. 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.

keywords : Cyber-Physical System, Real-Time Simulation, Temporal Correctness, Functional Correctness, Efficient Simulation

Student Number : 2011-30966

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Motivation and Objective

Cyber-Physical Systems (CPS) such as an automotive control system and a smart grid system are integrations of computation (cyber-system) with physical processes (physical-system) [1]. In a CPS, the cyber-system consists networked embedded processers and tasks which are executed on the processors. This cyber-system monitors and controls the physical processes, usually with feedback loops where physical processes affect computations and vice versa [1].

Since a CPS is much more complex and large compared to the traditional embedded systems, when developing the CPS, reducing the development effort and cost is one of the most important issues to be considered. To reduce the development effort and cost, it is essential to accurately predict the final performance at the design phase. For such prediction, simulators are commonly used like Simulink [3] and LabVIEW [4]. 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, in the automotive control system, for LKAS (Lane Keeping Assistance System) that aims at keeping the vehicle at the center of the lane [2], Figure 1.1 shows noticeable gap between the pre-

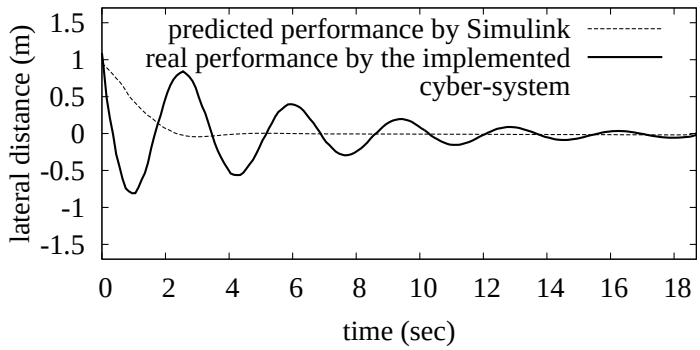


Figure 1.1 Predicted performance and real performance of LKAS

dicted performance by Simulink simulation and the real performance by the actually implemented cyber-system. By Simulink, the LKAS algorithm seems to quickly control the vehicle to the center of the lane and keep it stable. However, the actual performance says that the vehicle actually oscillates for a long time around the center of the lane. This is because Simulink executes the LKAS algorithm assuming the ideally periodic timing model, which is not true after implementation due to varying execution times and task preemptions, etc. This inaccuracy of existing simulators leads to enormous time and cost for a system development, because multiple re-design and re-implementation phases are required, until an acceptable system configuration is determined.

Motivated by this limitation, in this dissertation, 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 mod-

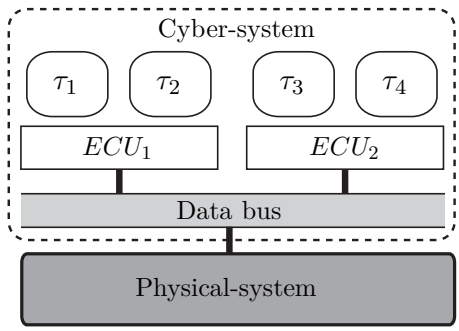


Figure 1.2 Example of a CPS

eling 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, e.g., a vehicle dynamics simulator like CarSim RT [42] or an actual vehicle for the automotive system.

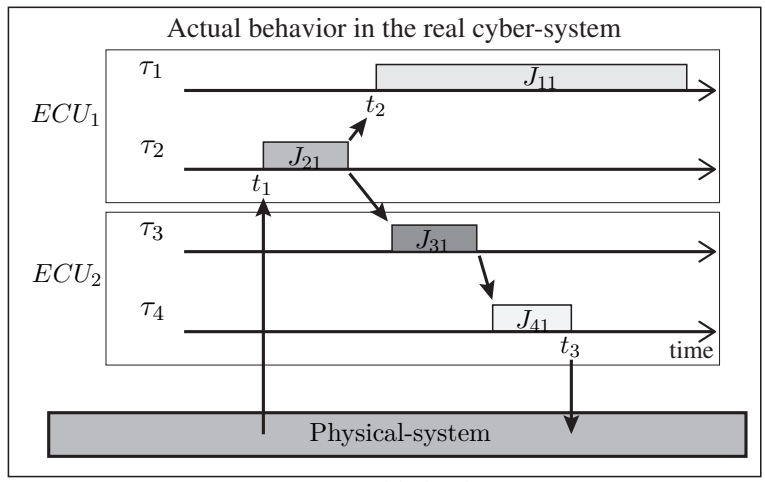
1.2 Approach

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

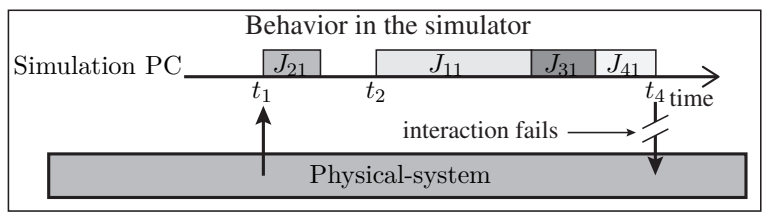
algorithm for the functionally and temporally correct real-time simulation.

In order to explain the key idea of the proposed approach, let us look at Figure 1.2. The figure shows an example of a CPS. Our simulation target, the cyber-system, consists of two embedded processors, so called ECUs (Electronic Control Unit), and four tasks $\{\tau_1, \cdots, \tau_4\}$. In order to validate the simulation target correctly with the physical-system, the functional and temporal behavior of the four tasks, from τ_1 to τ_4, on the simulated ECUs should be the same as if they are executing on their real ECUs.

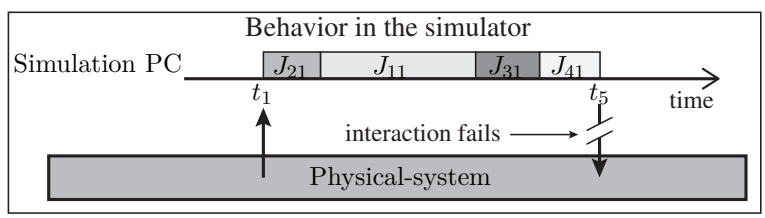
In order to explain what "correct simulation" is, let us look at Figure 1.3. Figure 1.3(a) shows an example "actual behavior" of the four tasks from τ_1 to τ_4 in Figure 1.2, which can be obtained by assuming that they are scheduled and executed on real ECUs by their scheduling policy. In the figure, the j-th job of τ_i is denoted by J_{ij}, and the arrows between jobs represent the data dependency between them. The correct simulation of the actual behavior is that "functionally and temporally" correct mimicking of the schedule in terms of the physical interaction points. In other words, J_{21} should start at t_1 with data from the physical-system, then, the output data of the job should pass through J_{31} and J_{41}, and finally, J_{41} should send its output to the physical-system at its expected finish time, i.e., t_3. From now on, let us assume that we try to mimic the actual behavior of the four tasks on the simulation PC, which has a singlecore processor. Then the problem is how to execute all the jobs in Figure 1.3(a) on a single timeline. Note that the execution time of a job on the simulation PC, i.e., powerful PC CPU, is shorter than the expected execution



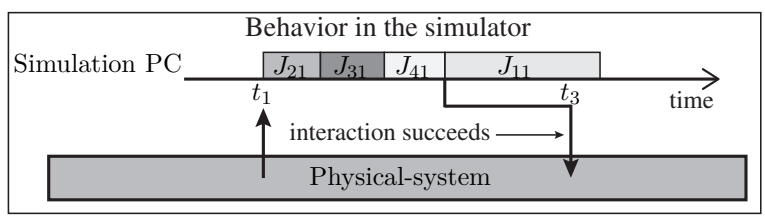
(a) Actual behavior



(b) The simplest way to execute jobs



(c) More efficient way to execute jobs



(d) The most efficient way to execute jobs

Figure 1.3 Key idea of the proposed approach

time of the job on its ECU, i.e., slow embedded processor. Figure 1.3(b) shows the simplest way to execute jobs on the simulation PC, that is, starting jobs strictly at their expected start times and emitting the outputs of the jobs strictly at their expected finish times in the actual behavior. Unfortunately, it is not always possible on the simulation PC as shown in Figure 1.3(b), that is, J_{31} and J_{41} cannot start at their expected start times and in turn J_{41} cannot send its output at its expected finish time (t_3), which is the violation of correct physical interaction. However, if we relax the starting condition of J_{11} by releasing it earlier than its expected start time (t_2) as in Figure 1.3(c), the simulator can finish J_{41} earlier (t_5) than the finish time at Figure 1.3(b) (t_4) still maintaining the data dependency. This is more efficient than Figure 1.3(b). However, J_{41} cannot still send its output to the physical-system at t_3. Now, let us look at Figure 1.3(d). In the figure, J_{31} and J_{41} start earlier than J_{11}, and finally, J_{41} can finish its execution earlier than t_3 still maintaining the data dependency. Thus, the simulator can send the correct result at the correct time (t_3) to the physical-system. Note that although the schedule in the simulation PC is not the same as the actual behavior in the real cyber-system, from the perspective of the physical-system, the simulation result is correct because the right value is emitted at the right time point.

As exemplified above, when simulating a large number of tasks, it is important to maximize the simulation capacity, so called "real-time simulatability", i.e., the number of simulatable tasks, since a single host processor has to execute a large number of jobs from dozens or hundreds of simulated ECUs. Thus,

it is important to find an efficient way to execute jobs on the simulation PC that maximizes the simulatability while maintaining the correct simulation result. In this context, let us introduce the following two major considerations of our approach:

- **Correct simulation.** From the perspective of the physical-system, the simulated ECUs should yield the right value at the right time to the physical-system as if the simulated tasks are scheduled exactly the same as the actual behavior in the real cyber-system.
- **Efficient simulation.** While maintaining the correct simulation, we should make the best use of the simulation PC by efficiently executing simulated jobs on the simulation PC.

With these two major considerations in mind, as a solution to the simulation problem, we propose a simulated job execution algorithm ensuring correct and efficient simulation. More specifically, we solve the simulation problem into two steps. In the first step, for simplicity, we solve the problem assuming all tasks' execution times in the real cyber-system are fixed. Then, for practicality, we extend the previous approach assuming all tasks' execution times are varying. In each step, we first transform our simulation problem into a real-time job scheduling problem by considering the temporal and functional dependencies among simulated tasks and interactions with the physical-system. Then, we propose a proper scheduling algorithm for a singlecore PC.

1.3 Contributions

The contributions of this dissertation can be summarized as follows:

- We identify a new real-time simulation problem for correct interactions between the cyber-system and the physical-system and propose a systematic way to transform it to a real-time job scheduling problem.
- For the transformed job scheduling problem, we propose a proper algorithm for a simulation PC who has a singlecore processor.

1.4 Organization

The rest of this dissertation is organized as follows:

- Chapter 2 surveys the related work, which includes design and verification of CPSs, verification approaches, and real-time scheduling approaches.
- Chapter 3 presents the system model, assumptions, and terms, and formally describes the problem for this dissertation.
- Chapter 4 presents our simulation approach when all tasks' execution times in the real cyber-system are fixed.
- Chapter 5 extends the previous approach for more general system that all tasks' execution times in the real cyber-system are varying.
- Chapter 7 extends the proposed approach considering practicality.

- Chapter 8 concludes this dissertation and discusses the future work.

Chapter 2
Related Work

2.1 Design and Verification of Cyber-Physical Systems

In CPSs, since the cyber-system monitors and controls the physical processes, usually with feedback loops where physical processes affect computations and vice versa, CPSs design involves various distinct disciplines such as control engineering, software engineering, mechanical engineers, network engineering, etc [5].

A complex system like CPSs is currently being developed following the V-cycle process as in Figure 2.1. This process basically consists of (1) requirements analysis, (2) system design & architecture, (3) module design & verification, (4) coding, (5) unit testing, (6) integration testing, and (7) acceptance testing. In the requirements analysis phase, the requirements of the system to be developed are analyzed. Then, the overall system design and HW/SW architecturing are conducted in the second phase. In this phase, the developers decide how many ECUs are used for the system and which control algorithms are executed on each ECU. After that, each algorithm is developed and verified individually in the module design & verification phase. In this phase, model-

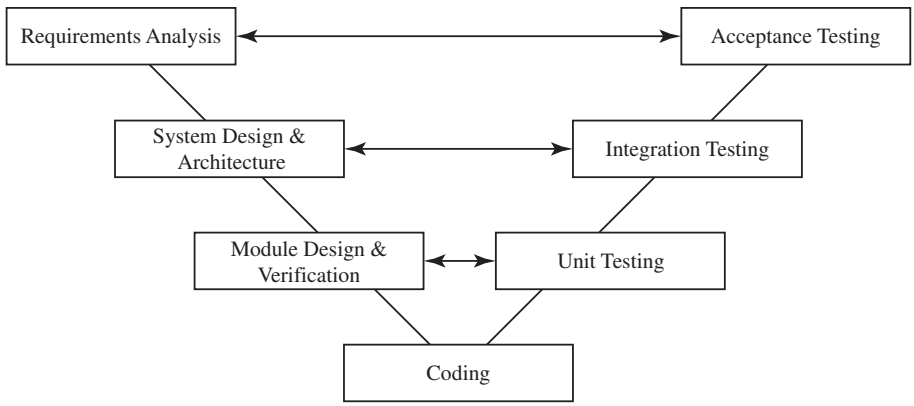


Figure 2.1 Development process of a CPS

based design and verification tool such as Simulink [3] and LabVIEW [4] are commonly used. Also, to verify the functionality of the control algorithm in a real-time environment, the rapid prototyping is conducted. For this, the algorithm is automatically downloaded to the rapid prototyping hardwares such as AutoBox [12], which are actually connected to the physical-system. After completing the module design & verification phase, to execute the control algorithm on the actual ECU, the algorithm is coded in the coding phase. For this, the developers can write the program code manually, or use automatic code generation tools such as Embedded Coder [6] and TargetLink [7]. After the developed program code is compiled and download to the target ECU, the actual performance of the program code on the ECU is verified. This is called as the unit testing. In this phase, the developers compare the performance of the program code on the ECU with the performance of the control algorithm on the model-based verification tools or on the rapid prototyping

hardwares. If the result of the unit testing phase is far different from the result of the module design & verification phase, the developers have to return to the module design & verification phase, and redesign and reverify of the control algorithm. Even they may return to the system design & architecture phase, if there are critical performance problems in the unit testing phase such as lack of the computing power of the ECU. After the unit testing phase, the actual ECUs are integrated and verified all together. This is called as the integration testing. In this phase, the system is operated well as intended in the system design & architecture phase while ECUs are correctly interacting with other ECUs and the physical-system. Finally, before the system is deployed, the acceptance testing is conducted to verify whether all requirements of the system are satisfied.

2.2 Verification Approaches

2.2.1 Model-Based Simulations

When developing a complex real-time system like a CPS, an important principle is the separation between the functionality of control algorithms and the platform where the algorithms are implemented. This principle is adopted by modern design methodologies such as Model Driven Architecture (MDA) and Platform Based Design (PBD) [8]. To support this principle when the developers develop control algorithms, various model-based development tools are used. Among them, in this subsection, we study model-based simulation

approaches.

In the commercial domain, Simulink [3] and LabVIEW [4] are widely used for the design time verification of the control algorithms. The algorithms are described by hierarchical models of graphical block diagrams and data exchange between models in Simulink and LabVIEW. They supports rich and expressive models of computation with continuous and discrete time domain. However, they do not correctly model the timing events such as 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.

To overcome this limitation, Timing Definition Language (TDL) is proposed [9]. In the concept of TDL, not only the semantics but also the temporal requirements of the control algorithms are considered. Ptolemy [10] is the one of model-based simulation tools which support this TDL concept. Ptolemy is an open-source software framework supporting experimentation with actor-oriented design [10]. In Ptolemy, the developers can describe execution times of software components, then Ptolemy can simulate the target system considering these execution times. However, since Ptolemy does not consider how the software components are actually deployed in the cyber-system, i.e., task-ECU mapping, there is a limitation that Ptolemy cannot reflect actual timing events such as task preemptions which will happen once the cyber-system is really implemented.

TrueTime [11] is another simulation tool which supports the TDL concept.

This tries to simulate jobs accurately modeling the scheduling events that will happen on multiple embedded processors. Therefore, TrueTime can correctly predict the final performance of the cyber-system in the design phase. However, it tries to exactly follow the job execution order that will happen in the real cyber-system. Thus, it cannot enjoy the freedom of job execution order as ours and hence the real-time simulatability is quite limited.

To sum up, since the all of model-based simulation tools introduced in this subsection have the limitation that they are aiming at offline simulation, they cannot provide real-time validation, that is, performing in real-time while interacting with the real working physical-side.

2.2.2 Cycle-Accurate Simulations and Host-Compiled Simulations

Meanwhile, for accurate modeling of events in the cyber-system, we can think of cycle-accurate instruction set simulators [15–20]. They can correctly predict the temporal and functional performance of the control algorithms as if the algorithms are executed on the real ECU. However, they are too slow to simulate all the cyber-system including application tasks and operating systems of multiple ECUs.

To overcome the limitation of the existing cycle-accurate instruction set simulators, recently, host-compiled simulation draws much attention due to its fast and time-accurate simulation. In the host-compiled simulation, each control algorithm is provided in the form of C code with the target specific ex-

ecution timing information, and the algorithm is simulated on top of abstract models of the operating system and the processor hardware [21]. However, most of host-compiled simulations [21–23] are targeting non-real-time systems like a smart phone without interactions with the physical-system. Targeting real-time systems, [24–26] proposes an RTOS simulator which executes jobs simulating RTOS scheduling events. However, it is limited to a single embedded processor, and still does not consider real-time interactions with the physical-system.

To sum up, all of cycle-accurate simulations or host-compiled simulations just focus on the accurate simulation of the target system, and does not consider real-time interactions with the physical-system.

2.2.3 Real-Time Execution Platforms

For the real-time validation, there are several real-time execution platforms. In the commercial domain, first, the rapid prototyping hardwares like AutoBox [12] is commonly used. Since these hardwares have special features for interacting the physical-system such as Control Area Network (CAN) connectors and power supply for 12 V, 24 V, and 48 V electrical systems, the developers can easily test the control algorithms with the physical-system. However, the rapid prototyping hardwares just provides fast executions of control algorithms for the interaction with real working physical systems. They 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 [13] and NI-HIL [14], have the same limitation.

In the literature, to support the actual execution of TDL-based control algorithms, a real-time execution model called E-machine is proposed [27]. In E-machine, TDL-based control algorithms are transformed into virtual instructions called E-code, then E-machine interprets the instructions at runtime and ensures the correct timing behavior. Meanwhile, the another trend in the real-time execution platform is FPGA-based execution of the simulation models [28]. In this approach, the simulation timestep can be very small, in the order of 250 nanoseconds, and computation time within each timestep is almost independent of system size because of the parallel nature of FPGAs [29]. However, they do not consider how to enhance the real-time simulatability.

To sum up, all approaches introduced in this subsection can support the real-time execution of the simulation models. However, none of them can consider improving the real-time simulatability by enjoying the freedom of scheduling simulated jobs while keeping the functional and temporal correctness.

2.2.4 Distributed Simulations

Recent studies on “Distributed Simulation” or “Co-Simulation” consider interactions between simulators and the physical-system. To support interactions between simulators, Functional Mock-up Interface (FMI) [30] and High Level Architecture (HLA) [31] are standardized. The goal of them are supporting

interoperability of distributed simulations. Thus, they just focus on common interfaces between simulations, they neither consider the real-time interactions between simulations. Likewise, [32–36] care the issue about time synchronization between two different simulators. However, both of them do not care the real-time interactions with the physical-system.

For the real-time interactions, [37, 38] care the issue about time synchronization between simulated time and wall-clock time which occurs when a simulator interacts with the physical-system. However, they do not study on improving the real-time simulatability by enjoying the freedom of scheduling simulated jobs while keeping the functional and temporal correctness.

2.3 Job Scheduling Approaches

In this dissertation, we transform the simulation problem into a real-time job scheduling problem by considering the temporal and functional dependencies among simulated tasks and interactions with the physical-system. For this, we first construct a job-level precedence graph with non-determinism caused by varying execution times of jobs. Then we schedule jobs on the simulation PC guided by the graph while resolving non-determinism progressively. In this regard, we first introduce researches about DAG (Directed Acyclic Graph) model for tasks. In the job scheduling domain, precedence relations between tasks are described as a DAG. Especially, a DAG who includes non-deterministic precedence relations has been recently studied under the name of "conditional

DAG" [58–60]. In the conditional DAG, non-determinism is caused by the conditional statement in task codes, and only one non-deterministic successor becomes valid among successors of a job after the conditional statement in the job is executed. On the other hand, in our proposed algorithm, non-determinism in a DAG is caused by varying execution times of tasks. Also, although non-determinism is resolved, jobs in the DAG are still valid.

From now on, we present researches related job scheduling on the DAG. In terms of job scheduling on the given DAG, the proposed scheduling algorithm in this dissertation is a kind of study that decides topological ordering of jobs in the DAG. Studies on topological ordering can be divided into three categories. First, there are studies to reduce the time complexity or execution time of the algorithm that performs all jobs while maintaining partial order [61–63]. Second, there are studies to reduce the system time which is needed to execute all jobs considering weights of edges and jobs in a DAG [64–66]. Last, there are studies to find a feasible schedule which meets all deadlines of jobs when deadlines of the jobs are given [40,67]. Among these three categories, our study belongs to the third category. Therefore, we focus on this category which has been studied in real-time systems domain.

A real-time system is a system which is defined as not only the functional correctness but also the temporal correctness. It is common to divide temporal constraints into two types: hard and soft [39]. If the failure to meet the timing constraint is considered to be a fatal fault, the timing constraint is *hard*. In contrast, a few the deadline misses are permitted, the system is *soft* real-time.

Since a CPS is the hard real-time system and we assume that the simulation PC has a singlecore, we focus on how to schedule hard real-time tasks on the singlecore processor in this section.

There are three commonly used approaches for scheduling real-time systems: (1) clock-driven, (2) weighted round-robin, and (3) priority-driven approach [39]. In the clock-driven (also called time-driven) approach, all scheduling decisions are decided before the system begins its execution. Therefore, scheduling overhead can be minimized. In the weighted round-robin approach, each job has its own *weight*. All jobs are executed in First-In-First-Out (FIFO) manner, but the time that each job can occupy the processor is decided based on the job's weight. In the priority-driven approach, each job has its own priority. Therefore, all jobs are executed in the order of their priorities. From now on, we study the priority-driven scheduling approach.

First, we study task-level static priority scheduling approaches (also called fixed-priority scheduling approaches), which assign a fixed priority to each task. In this domain, RM (Rate Monotonic) scheduling is known to be optimal for a periodic task set with implicit deadlines in the fully preemptive system. If tasks has arbitrary deadlines, then DM (Deadline Monotonic) shceduling is known to be optimal.

In order to check the schedulability of a task set under a certain scheduling approach, two types of schedulability analysis are commonly used. The first one is the utilization bound check. Liu and Layland [40] showed that a set of

periodic tasks $\{\tau_1, \tau_2, \cdots, \tau_n\}$ is schedulable under RM, if

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{\frac{1}{n}} - 1), \tag{2.1}$$

where C_i is the worst-case execution time and P_i is the period of τ_i. Note that this is only a sufficient condition, thus a set of tasks with its system utilization over Eq. (2.1) may be schedulable. The other schedulability test is the worst-case response time analysis. Audsley et al. [41] showed the following equation can compute the worst-case response time R_i of τ_i under RM:

$$R_i^{k+1} = C_i + \sum_{\tau_m \in HP(i)} \left\lceil \frac{R_i^k}{P_m} \right\rceil C_m, \tag{2.2}$$

where $HP(i)$ is a set of tasks whose priorities are higher than the priority of τ_i and $R_I^0 = C_i$. Eq.(2.2) continues until $R_i^{k+1} = R_i^k$. If and only if all response times of tasks are less than or equal to their deadlines, the system is schedulable.

Now, we study job-level static priority scheduling approaches (also called dynamic-priority scheduling approaches), which assign a fixed priority to each job. In this domain, EDF (Earliest Deadline First) scheduling is known to be optimal for scheduling preemptive jobs with arbitrary release times and deadlines on a singlecore processor.

In order to check the schedulability of the system under EDF scheduling approach, Liu and Layland [40] showed that a set of periodic tasks $\{\tau_1, \tau_2, \cdots, \tau_n\}$

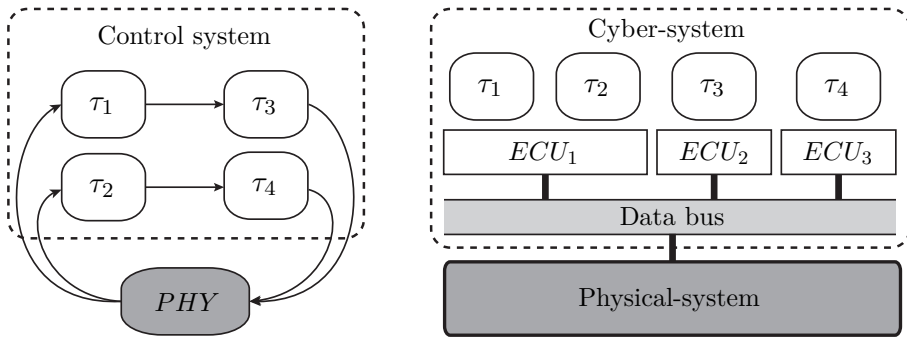
with implicit deadlines are schedulable, if and only if the following equation is satisfied,

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1. \tag{2.3}$$

Basically we use this EDF scheduling to schedule jobs on the simulation PC. However, whenever a job finishes its execution on the PC, for the simulation efficiency, we reconstruct the job-level precedence graph considering the actual execution time of the job on the ECU, then recalculate deadlines of remaining jobs in the graph. This is the main contribution of the proposed approach in terms of job scheduling.

Chapter 3
System Model and Problem Description

Control engineers design a control system as in Figure 3.1(a). Then, it should be realized as a cyber-system as in Figure 3.1(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 Figure 3.2(a). Our problem is to simulate jobs beforehand on the simulation PC. In the rest of this dissertation, by **simulating a job**, we mean **executing a job on the simulation PC**. If we simulate the jobs as in Figure 3.2(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



(a) Control system (b) Cyber-system

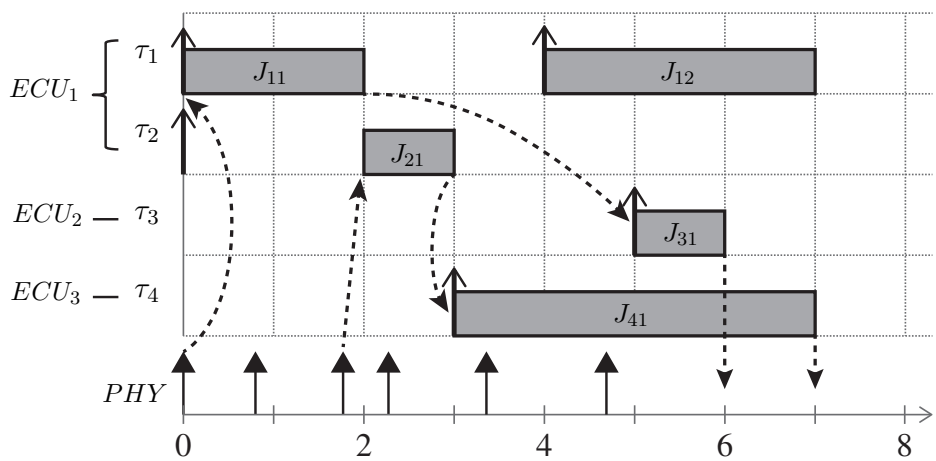
Figure 3.1 Cyber-physical system design

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.

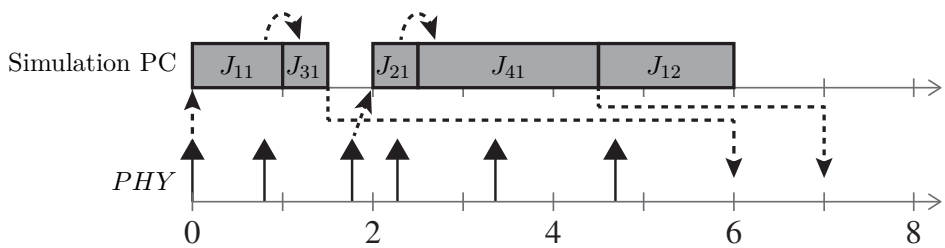
3.1 Description on the real cyber-system

The control system designed by control engineers is given as a graph $G = (V, E)$ as in Figure 3.1(a). V is the set of nodes where 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 rep-



(a) Execution scenario of the real cyber-system



(b) Execution scenario of the simulated cyber-system

Figure 3.2 Execution scenario

resented by m ECUs, i.e.,

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

and tasks mapped to each ECU as in Figure 3.1(b). We assume that each ECU is an embedded computer equipped with a single core processor like Infineon TC1797 [43], NXP MPC5566 [44] 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) [45] and FlexRay [46]. 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 Chapter 6, 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.

For example, J_{21} in Figure 3.2(a) reads all the input data as an atomic operation at the entry and writes all the output data as another atomic operation at the exit. Also, at the start time of J_{21}, the most recent PHY data is used for the execution of J_{21}.

3.2 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 [47] in PC vs. TC1797 [43] 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 Chapter 5.5. 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.

[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.

- **Tagged 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 with a time-tag and intentionally delay its actual write to a specific time of the time-tag.

3.3 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.

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.,

$$t_{ij}^{\mathsf{S,sim}} \geq t_{ij}^{\mathsf{S,real}}. \tag{3.1}$$

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.,

$$t_{ij}^{\mathsf{F,sim}} \leq t_{ij}^{\mathsf{F,real}}. \tag{3.2}$$

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 "Tagged 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.,

$$t_{i'j'}^{\mathsf{F,sim}} \leq t_{ij}^{\mathsf{S,sim}}. \tag{3.3}$$

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.

Chapter 4
Real-Time Simulation for Deterministic Cyber-Systems

4.1 Introduction

In this chapter, we explain our simulation approach when the real cyber-system is deterministic, that is, we assume all execution times of tasks are fixed, i.e., $C_i^{\mathsf{best}} = C_i^{\mathsf{worst}}$ for each task τ_i. The approach consists of two steps: (1) in the offline phase, we construct a job-level precedence graph, so called an *offline guider*, which represents the aforementioned constraints and (2) in the online phase, we progressive schedule jobs on the simulation PC guided by the offline guider, which we call *online progressive scheduling*.

4.2 Construction of Offline Guider

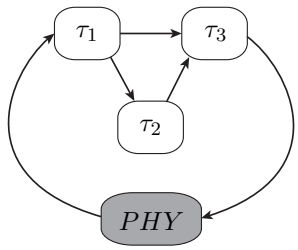
For constructing the offline guider, we consider only jobs in one hyperperiod, i.e., HP. During one HP, a task τ_i has $n_i = HP/P_i$ jobs. For those jobs in our interested HP, we index them as $J_{i1}, J_{i2}, \cdots, J_{in_i}$ while we index those in the previous HP as $J_{i(-(n_i-1))}, \cdots, J_{i(-1)}, J_{i0}$ 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.

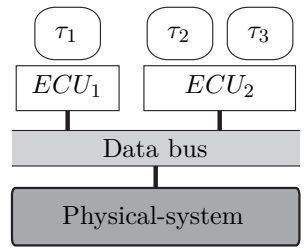
Here, without loss of generality, we use a single job $J_{ij} \in \{J_{i1}, J_{i2}, \cdots, J_{in_i}\}$ to explain how to determine its predecessors. For J_{ij}, we first add its previous job $J_{i(j-1)}$ as its deterministic predecessor, since control task usually uses the context of its previous instance 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 generate a "reference schedule" of the real cyber-system. In order to do that, we generate a timeline for each ECU ECU_i that indicates at which time points each job starts, is preempted, resumes, and finishes according to its scheduling policy. Note that since execution times are fixed, all timings in the reference schedule are deterministic. The reference schedule is generated for one HP. In the example of Figure 4.1(a), if periods of τ_1, τ_2, and τ_3 are 10, 10, and 15, and execution times on each ECU are 3, 4, and 4, respectively, we can generate the reference schedule from time 0 to their hyperperiod, i.e., 30, as in the figure. As a result, for every job $J_{i,j}$ in the reference schedule, we can get the start time $t_{ij}^{\mathsf{S,real}}$ and finish time $t_{ij}^{\mathsf{F,real}}$ of each job J_{ij}.

After generating the reference schedule, we construct the offline guider considering three constraints for simulation correctness. If a job J_{ij} has a physical-read constraint, we just mark R on the J_{ij} to express that this job should be executed on the simulation PC meeting Eq. (3.1). For example, in Figure 4.1(b), J_{11}, J_{12}, and $J13$ have R mark since they have physical-read constraints.



Control design

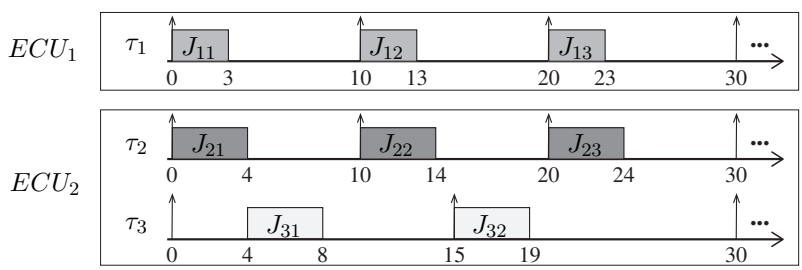


Task-ECU mapping

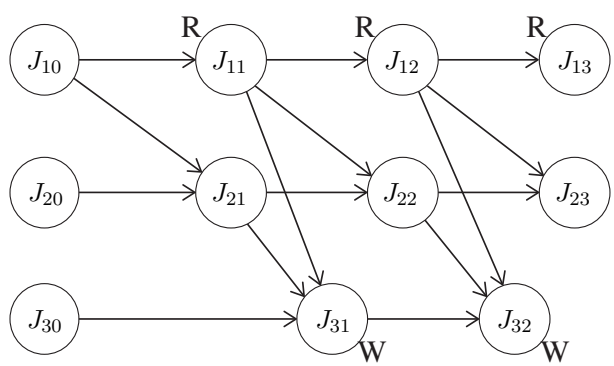
	Φ_i	P_i	C_i^{best}	C_i^{worst}
τ_1	0	10	3	3
τ_2	0	10	4	4
τ_3	0	15	4	4

Task parameters

(a) Example cyber-system



(b) Reference schedule



(c) Offline guider

Figure 4.1 Example of constructing the offline guider when the system is deterministic

If a job J_{ij} has a physical-write constraint, we just mark W on the J_{ij} to express that this job should be executed on the simulation PC meeting Eq. (3.2). For example, in Figure 4.1(b), J_{31} and $J32$ have W mark since they have physical-write constraints.

The last constraint to be considered is a producer-consumer constraint. By "Most Recent Data Use" and "Entry Read and Exit Write" properties of the real cyber-system, if $\tau'_i \rightarrow \tau_i$, $J_{i'j'}$ becomes a data producer of J_{ij} where $t^{\mathsf{F,real}}_{i'j'} \leq t^{\mathsf{S,real}}_{ij} < t^{\mathsf{F,real}}_{i'(j'+1)}$. Then, for $J_{i'j'}$ and J_{ij}, since Eq. (3.3) should be met, $J_{i'j'}$ should become a predecessor of J_{ij}. For example, in Figure 4.1(b), J_{11} is a predecessor of J_{31} since $\tau_1 \rightarrow \tau_3$ and $t^{\mathsf{F,real}}_{11} \leq t^{\mathsf{S,real}}_{31} < t^{\mathsf{F,real}}_{12}$. Likewise, J_{11} is a predecessor of J_{22}.

4.3 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 precedence edges in the offline guider are copied to *OJPG*. 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 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, $t_{ij}^{\mathsf{S,real}}$. 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., $t_{ij}^{\mathsf{S,sim}} \geq t_{ij}^{\mathsf{S,real}}$ in Eq. (3.1).

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 [49]. Thus, we assign the effective deadline $t_{ij}^{\mathsf{D,sim}}$ to each job $J_{ij} \in OJPG$ as follows:

$$t_{ij}^{\mathsf{D,sim}} = \begin{cases} \min(t_{ij}^{\mathsf{F,real}}) & \text{for } J_{ij} \text{ who has } W \text{ mark} \\ \infty & \text{otherwise} \end{cases}, \tag{4.1}$$

$$t_{ij}^{\mathsf{D,sim}} = \min\left(t_{ij}^{\mathsf{D,sim}}, \min_{\forall J_{kl} \in \mathbb{J}^{\mathsf{succ}}(J_{ij})}\left(t_{kl}^{\mathsf{D,sim}}\right)\right) \tag{4.2}$$

where $\mathbb{J}^{\mathsf{succ}}(J_{ij})$ is the set of successors of J_{ij}. In Eq. (4.1), we first initialize the deadline of each job in *OJPG*. For a job J_{ij} who has W mark, i.e., a physical-write constraint, we initialize its deadline as $t_{ij}^{\mathsf{F,real}}$ 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. (4.2), we backtrace jobs along the precedence edges and set each job J_{ij}'s deadline $t_{ij}^{\mathsf{D},\mathsf{sim}}$ as the minimum of its successors' deadlines.

At the time when the simulation PC is about to start a job J_{ij} with a physical-read constraint, the current time is already later than $t_{ij}^{\mathsf{S},\mathsf{real}}$. 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} have finished. 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 precedence edges from other jobs in $OJPG$ guided by the offline guider. Then, our online algorithm updates the effective deadlines of jobs in $OJPG$ by using the $t_{ij}^{\mathsf{F},\mathsf{real}}$s for J_{ij} who has W mark in Eq. (4.1) and newly added precedence edges in Eq. (4.2). This makes the simulation continue across multiple HPs.

This online 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. From now on, we prove this theoretically.

Lemma 4.1. *For any job J_{ij} that has a physical-read constraint, Eq. (3.1) is always satisfied.*

Proof. By our online algorithm, J_{ij} can be added to the simulation ready queue at $t_{ij}^{\mathsf{S},\mathsf{real}}$. Then, the simulation PC can start J_{ij} later or equal to $t_{ij}^{\mathsf{S},\mathsf{real}}$.

Thus, it follows that $t_{ij}^{\mathsf{S,sim}} \leq t_{ij}^{\mathsf{S,real}}$. □

Lemma 4.2. *For any job J_{ij}, all of its data producers have finished when the simulation PC starts J_{ij}.*

Proof. All of data producers of J_{ij} are predecessors of J_{ij} by the offline guider. Then, since our online algorithm add a job to the simulation ready queue only when there is no unexecuted predecessor of the job, when the simulation PC starts J_{ij}, all of data producers of J_{ij} have finished. □

Theorem 4.1. *If our online scheduling algorithm can schedule all the simulated jobs meeting their effective deadlines, it guarantees the functionally and temporally correct simulation.*

Proof. By Lemma 4.1 and Lemma 4.2, our algorithm can simulate each job with the same data as the real cyber-system, using the tagged data read.

Meanwhile, our online algorithm assigns the real finish time $t_{ij}^{\mathsf{F,real}}$ as the effective deadline of the simulated job J_{ij} with physical-write, using Eqs. (5.10) and (5.11). 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 tagged 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. □

4.4 Evaluation

In this section, we justify the proposed approach through simulation. For this, 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].
- 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]
- Its offset Φ_i is assumed zero.

- The best case execution time C_i^{best} is randomly determined as *uniform*[10,50]% of P_i. The worst case execution time C_i^{worst} is same as C_i^{best}.

We assume that each ECU is a singlecore system and RM (Rate Monotonic) scheduling policy is used. We also assume that simulated tasks are always schedulable on the ECU.

For such a synthesized cyber-system, we perform simulation for one HP using the following three 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 $t_{ij}^{\mathsf{S,real}}$ 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 [11]. 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 $t_{ij}^{\mathsf{S,sim}} \geq t_{ij}^{\mathsf{S,real}}$ 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 satisfy-

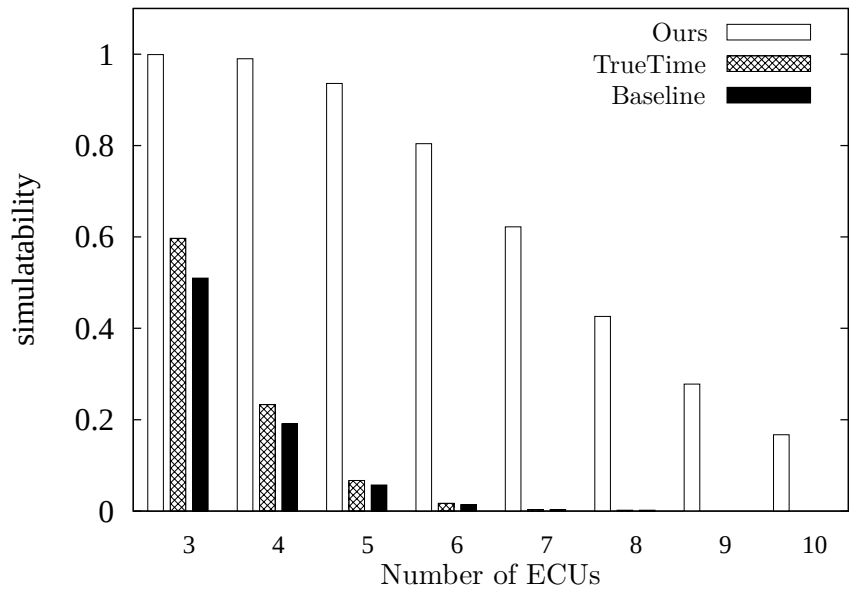


Figure 4.2 Simulatability as changing the number of the ECUs

ing only the constraints in Eqs. (3.1) and (3.3).

All the above three 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 $t_{ij}^{\mathsf{F},\mathsf{sim}} \leq t_{ij}^{\mathsf{F},\mathsf{real}}$ in Eq. (3.2), they can write the same physical data at the same time as the real cyber-system using the "Tagged 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.

Figure 4.2 compares the simulatabilities of the above three approaches as changing the number of the ECUs in the target system. The X-axis represents the number of the ECUs in the target system, and the Y-axis represents simulatability. For each number of the ECUs, we synthesize 1000 target systems with the aforementioned parameters. Compared with the baseline approach, i.e., **Baseline**, by enjoying the start time freedom, **TrueTime** can only achieve small improvement. However, if we execute jobs applying both freedoms of start times and job execution order, i.e., **Ours**, then the simulatability increases dramatically compared to **Baseline** and **TrueTime**. Note that our approach is optimal since for 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 [49].

Next, in order to observe the tendency of the simulatability according to relaxing each job's timing constraint, we measure the simulatability with adjusting the percentage of the simulated tasks who read data from the physical-system, i.e., f_{PR}, and write data to the physical-system, i.e, f_{PW}. First, Figure 4.3 compares the three approaches when f_{PR} varies from 0% to 100%. In the figure, X-axis represents such percentage while and Y-axis is the same as the previous graph. As shown in the figure, **Baseline** shows low constant simulatability regardless of f_{PR}. In contrast, **TrueTime** and **Ours** show varying improvement depending on f_{PR}. More specifically, **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

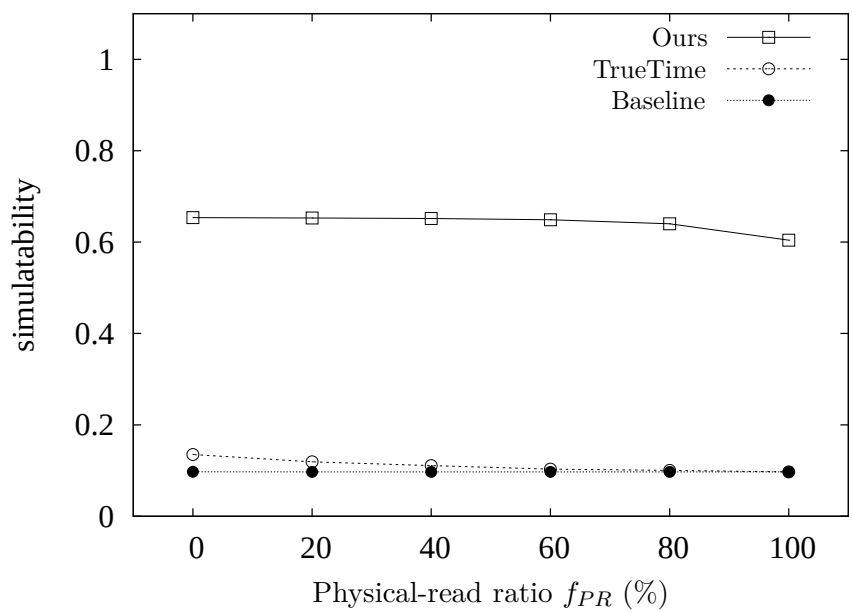


Figure 4.3 Simulatability as changing the physical-read ratio f_{PR}

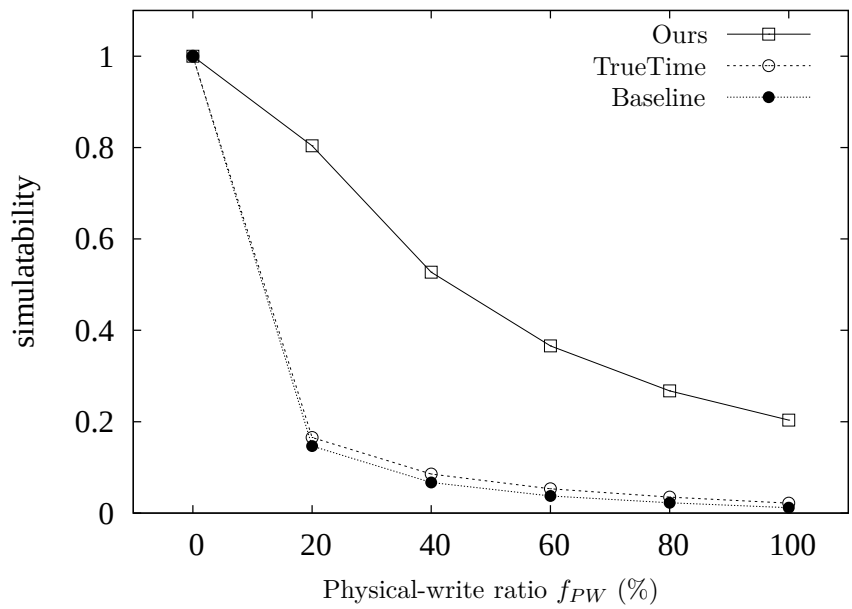


Figure 4.4 Simulatability as changing the physical-write ratio f_{PW}

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%

Figure 4.4 compares the three approaches as changing the physical-write ratio f_{PW} from 0% to 100%. For all the three approaches, the simulatabilities decrease as increasing f_{PW}. This is because jobs who write data to the

physical-system create deadlines to be enforced for the correct simulation. Due to this reason, when f_{PW} is zero, that is, no deadline to be enforced, all the target systems are correctly simulatable by all the three approaches. As increasing f_{PW}, the simulatabilities of all the three 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.

Chapter 5
Real-Time Simulation for Non-Deterministic Cyber-Systems

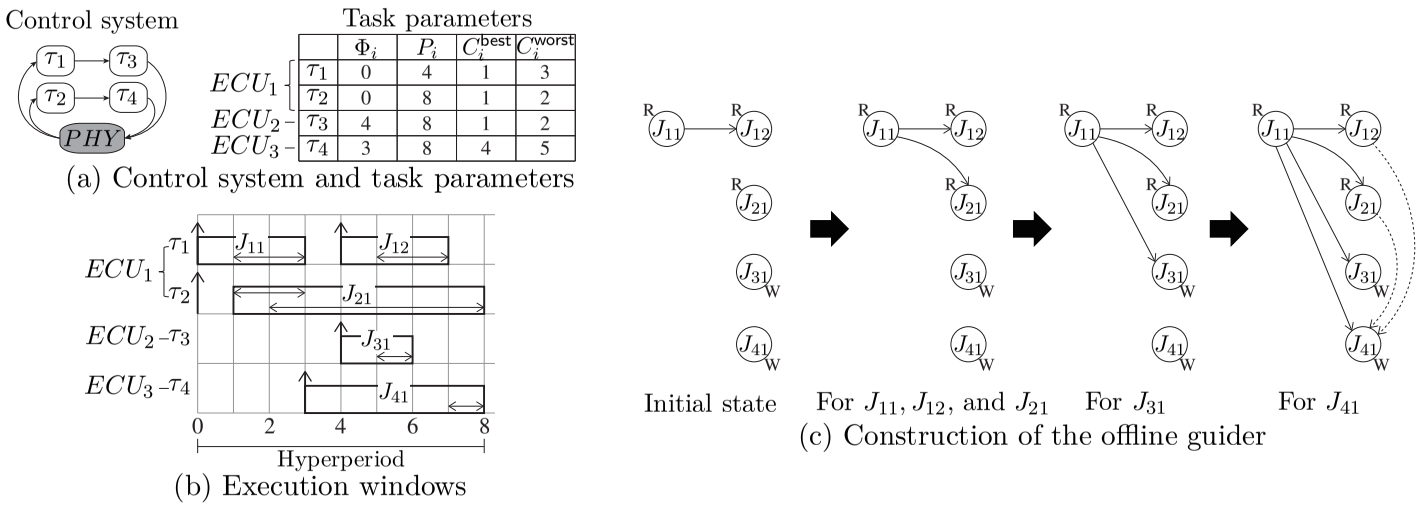
5.1 Introduction

In this chapter, we explain our simulation approach assuming all execution times of tasks can vary, i.e., $C_i^{\mathsf{best}} \leq C_i^{\mathsf{worst}}$ for each task τ_i. For the simulation PC to schedule jobs meeting the three types of constraints introduced Chapter 3.3, 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*.

5.2 Overview of Approach

Offline guider: Let us use the example in Figure 5.1(a) assuming the RM



(a) Control system and task parameters

(b) Execution windows

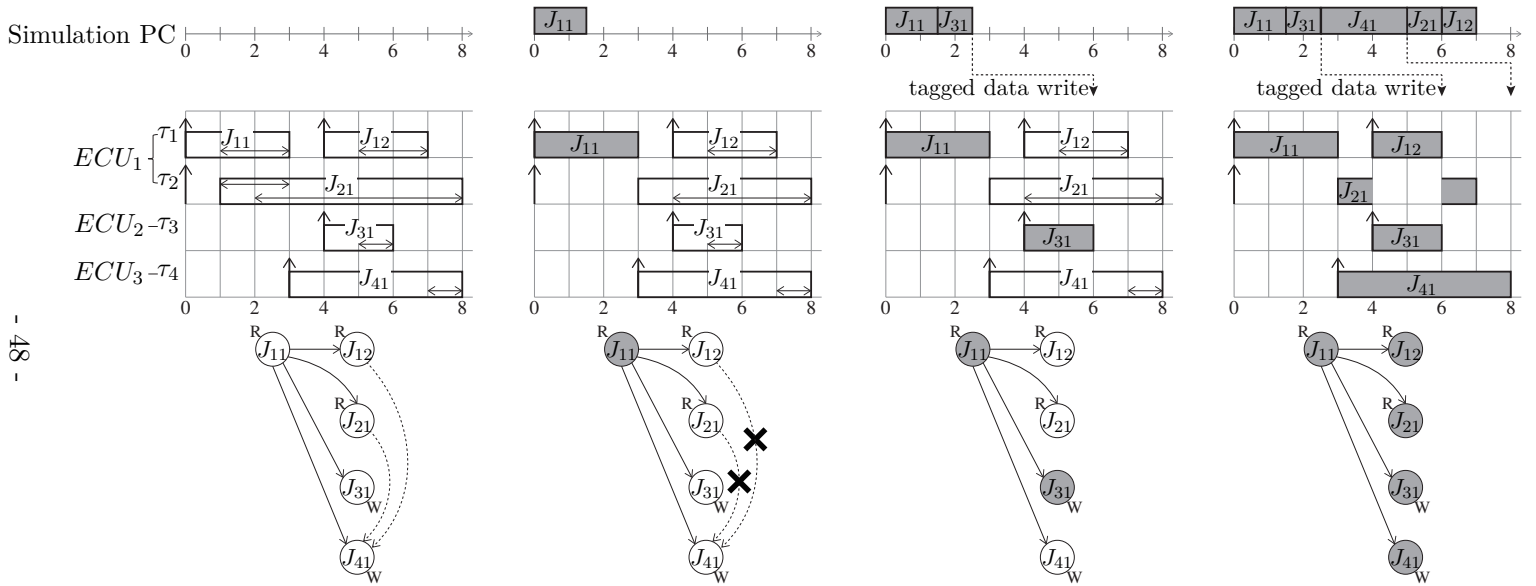
(c) Construction of the offline guider

Figure 5.1 Offline guider

scheduling policy for each ECU. For all the jobs during one hyperperiod of the four tasks as in Figure 5.1(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 Figure 5.1(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 Figure 5.1(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. (3.1), 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 Figure 5.1(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



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

Figure 5.2 Online progressive scheduling

in Eq. (3.2), we have to know $t_{41}^{\mathsf{F},\mathsf{real}}$. 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 Figure 5.1(b). If $t_{21}^{\mathsf{F},\mathsf{real}} \leq 3$, 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 $t_{21}^{\mathsf{F},\mathsf{real}}$ before simulating J_{41}. Thus, jobs that can affect $t_{21}^{\mathsf{F},\mathsf{real}}$, i.e., J_{11}, J_{12}, and J_{21} in the example, become predecessors of J_{41} in Figure 5.1(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 Figure 5.1(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}.

Online progressive scheduling: We overview our online progressive scheduling with an example in Figure 5.2 assuming $e_{ij}^{\mathsf{real}} = 2 \cdot e_{ij}^{\mathsf{sim}}$. In the beginning, the simulation PC starts with the offline guider as in Figure 5.2(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 Figure 5.2(b). Let us assume that $e_{11}^{\mathsf{sim}} = 1.5$ and hence $e_{11}^{\mathsf{real}} = 3$. Using $e_{11}^{\mathsf{real}} = 3$, it can recalculate J_{21}'s finish time range as in Figure 5.2(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 pre-

decessors, 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., $t_{31}^{\mathsf{F},\mathsf{real}} < t_{41}^{\mathsf{F},\mathsf{real}}$, due to the physical-write constraints. Thus, the simulation PC selects J_{31} as the next simulation job as in Figure 5.2(c). After completing J_{31} at 2.5, the simulation PC knows $e_{31}^{\mathsf{sim}} = 1$ and hence $e_{31}^{\mathsf{real}} = 2$. Using $e_{31}^{\mathsf{real}} = 2$, it computes the schedule on ECU_2 and knows the real finish time of J_{31} is 6 as in Figure 5.2(c). Thus, it plans the tagged data write operation that will happen at 6. Like this, it progresses simulating jobs while resolving non-determinism guided by the offline guider.

5.3 Construction of Offline Guider

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. (3.1), we have to know $t_{ij}^{\mathsf{S,real}}$ in order to ensure that the simulation PC starts J_{ij} after $t_{ij}^{\mathsf{S,real}}$, i.e., $t_{ij}^{\mathsf{S,sim}} \geq t_{ij}^{\mathsf{S,real}}$. 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 [48]. For this, we compute the last worst case busy period $WCBP(J_{ij}) = [WCBP_{ij}^{\mathsf{start,\ real}}, WCBP_{ij}^{\mathsf{end,\ real}}]$ 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 $[\min(t_{ij}^{\mathsf{S,real}}), \max(t_{ij}^{\mathsf{S,real}})]$ of J_{ij} on the real cyber-system, where $\min(t_{ij}^{\mathsf{S,real}})$ and $\max(t_{ij}^{\mathsf{S,real}})$ 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 $[\min(t_{ij}^{\mathsf{S,real}}), \max(t_{ij}^{\mathsf{S,real}})]$ of J_{ij}, we can conservatively compute the set of jobs

that can possibly affect $t_{ij}^{\mathsf{S,real}}$ as follows:

$$\mathbb{J}^{\mathsf{S}}(J_{ij}) = \{J_{kl} | J_{kl} \in \mathbb{J}^{\mathsf{high}}(J_{ij}), WCBP_{ij}^{\mathsf{start,real}} \leq t_{kl}^{\mathsf{R,real}} < \max(t_{ij}^{\mathsf{S,real}})\} \quad (5.1)$$

where $\mathbb{J}^{\mathsf{high}}(J_{ij})$ 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 $t_{kl}^{\mathsf{R,real}}$ is after $WCBP_{ij}^{\mathsf{start,real}}$ but before J_{ij}'s latest start time $\max(t_{ij}^{\mathsf{S,real}})$ has potential to affect the start of J_{ij}. Thus, the set of such jobs, i.e., $\mathbb{J}^{\mathsf{S}}(J_{ij})$, is called the "start time set" of J_{ij}.

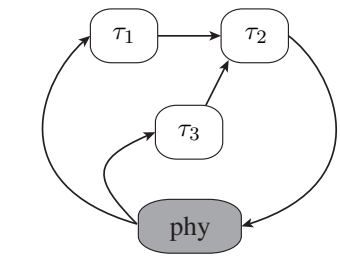
Out of the jobs in $\mathbb{J}^{\mathsf{S}}(J_{ij})$, the jobs whose latest start times are before the earliest start time of J_{ij} definitely affect the start of J_{ij} in the real cyber-system. Thus, the simulation PC should definitely execute them before J_{ij} to know their simulated execution times,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

$$\mathbb{J}^{\mathsf{S\text{-}det}}(J_{ij}) = \{J_{kl} | J_{kl} \in \mathbb{J}^{\mathsf{S}}(J_{ij}), \max(t_{kl}^{\mathsf{S,real}}) < \min(t_{ij}^{\mathsf{S,real}})\} \quad (5.2)$$

are designated as deterministic predecessors of J_{ij}.

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

$$\mathbb{J}^{\mathsf{S\text{-}nodet}}(J_{ij}) = \mathbb{J}^{\mathsf{S}}(J_{ij}) - \mathbb{J}^{\mathsf{S\text{-}det}}(J_{ij}) \quad (5.3)$$

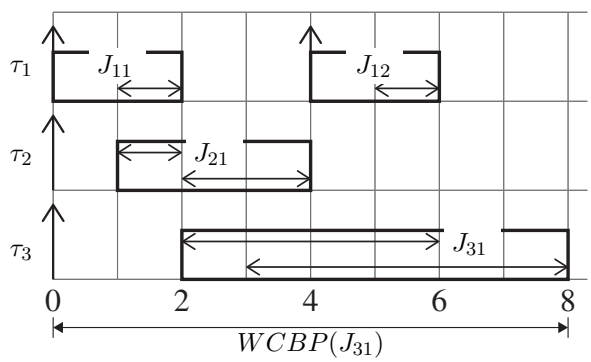


(a) Control system

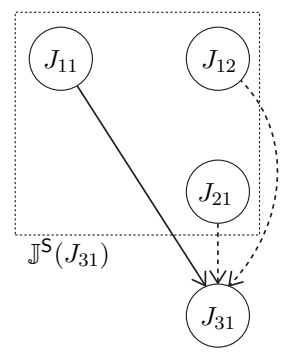
ECU_1

	Φ_i	P_i	C_i^{best}	C_i^{worst}
τ_1	0	4	1	2
τ_2	0	8	1	2
τ_3	0	8	1	2

(b) Task parameters



(c) Execution windows



(d) Precedence relations for the physical-read constraint of J_{31}

Figure 5.3 Precedence relations for the physical-read constraint

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

We explain how to set precedence relations for a job who has a physical-read constraint with an example. Let us use the example in Figure 5.3(a) and (b) assuming the RM scheduling policy for ECU_1. For all the jobs during one hyperperiod of the three tasks as in Figure 5.3(c), we can compute their start time and finish time ranges by considering the RM scheduling with the best and worst case execution times. In this example, we construct the offline

guider for J_{31} who has the physical-read constraint. First, the last worst case busy period of J_{31} $WCBP(J_{31})$ is [0,8] as shown in Figure 5.3(c). Then, by Eq. (5.1), $\mathbb{J}^{\mathsf{S}}(J_{31})$ is $\{J_{11}, J_{12}, J_{21}\}$ as in Figure 5.3(d). Among jobs in $\mathbb{J}^{\mathsf{S}}(J_{31})$, since $\max(t_{11}^{\mathsf{S,real}}) < \min(t_{31}^{\mathsf{S,real}})$, only J_{11} is in $\mathbb{J}^{\mathsf{S\text{-}det}}(J_{31})$. Thus, J_{11} becomes a deterministic predecessor of J_{31}, while other jobs in $\mathbb{J}^{\mathsf{S}}(J_{31})$, i.e., J_{12} and J_{21}, are non-deterministic predecessors of J_{31} as shown in Figure 5.3(d).

If J_{ij} has a physical-write constraint, i.e., Eq. (3.2), we have to know $t_{ij}^{\mathsf{F,real}}$ in order to ensure that the simulation PC finishes J_{ij} before $t_{ij}^{\mathsf{F,real}}$, i.e., $t_{ij}^{\mathsf{F,sim}} \leq t_{ij}^{\mathsf{F,real}}$. Similarly to the case of physical-read constraint, we can compute the finish time range $[\min(t_{ij}^{\mathsf{F,real}}), \max(t_{ij}^{\mathsf{F,real}})]$ of J_{ij}, where $\min(t_{ij}^{\mathsf{F,real}})$ and $\max(t_{ij}^{\mathsf{F,real}})$ 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 $t_{kl}^{\mathsf{R,real}}$ after $WCBP_{ij}^{\mathsf{start,real}}$ but before J_{ij}'s latest finish time $\max(t_{ij}^{\mathsf{F,real}})$ has potential to affect the finish time $t_{ij}^{\mathsf{F,real}}$ of J_{ij}. Also, J_{ij} itself affects $t_{ij}^{\mathsf{F,real}}$. Thus, a conservative set of jobs to be executed by the simulation PC to know $t_{ij}^{\mathsf{F,real}}$ is given as follows:

$$\begin{aligned}\mathbb{J}^{\mathsf{F}}(J_{ij}) =& \{J_{kl} | J_{kl} \in \mathbb{J}^{\mathsf{high}}(J_{ij}), WCBP_{ij}^{\mathsf{start,real}} \leq t_{kl}^{\mathsf{R,real}} < \max(t_{ij}^{\mathsf{F,real}})\} \\ & \cup \{J_{ij}\}.\end{aligned} \tag{5.4}$$

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 $t_{ij}^{\mathsf{F,real}}$ as long as it can finish J_{ij} before $t_{ij}^{\mathsf{F,real}}$. Thus, the jobs in $\mathbb{J}^{\mathsf{F}}(J_{ij})$ 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 Figure 5.1(c)) with zero execution time and designate all the jobs in $\mathbb{J}^{\mathsf{F}}(J_{ij})$ as predecessors of $\hat{J}_{ij}$. This way, it is enough to know $t_{ij}^{\mathsf{F,real}}$ before starting zero execution time job $\hat{J}_{ij}$. If the simulation PC can start and also finish $\hat{J}_{ij}$ before $t_{ij}^{\mathsf{F,real}}$, it means that the simulation PC finishes J_{ij} before $t_{ij}^{\mathsf{F,real}}$ meeting J_{ij}'s physical-write constraint.

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

$$\mathbb{J}^{\mathsf{F\text{-}det}}(\hat{J}_{ij}) = \{J_{kl} | J_{kl} \in \mathbb{J}^{\mathsf{F}}(J_{ij}), \max(t_{kl}^{\mathsf{S,real}}) < \min(t_{ij}^{\mathsf{F,real}})\} \tag{5.5}$$

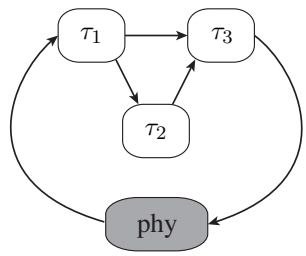
are designated as deterministic predecessors of J_{ij}'s terminal node $\hat{J}_{ij}$.

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

$$\mathbb{J}^{\mathsf{F\text{-}nodet}}(\hat{J}_{ij}) = \mathbb{J}^{\mathsf{F}}(J_{ij}) - \mathbb{J}^{\mathsf{F\text{-}det}}(\hat{J}_{ij}) \tag{5.6}$$

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

We explain how to set precedence relations for a job who has a physical-write constraint with an example. Let us use the example in Figure 5.4(a) and (b) assuming the RM scheduling policy for ECU_1. For all the jobs during one hyperperiod of the three tasks as in Figure 5.4(c), we can compute their start

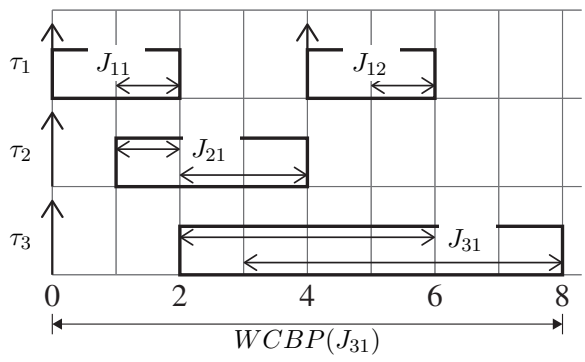


(a) Control system

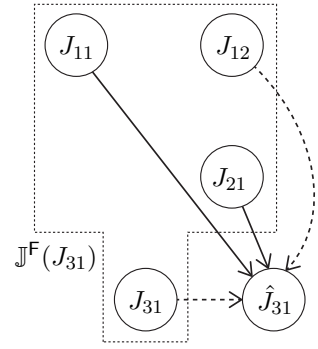
	Φ_i	P_i	C_i^{best}	C_i^{worst}
τ_1	0	4	1	2
τ_2	0	8	1	2
τ_3	0	8	1	2

ECU_1

(b) Task parameters



(c) Execution windows



(d) Precedence relations for the physical-write constraint of J_{31}

Figure 5.4 Precedence relations for the physical-write constraint

time and finish time ranges by considering the RM scheduling with the best and worst case execution times. In this example, we construct the offline guider for J_{31} who has the physical-write constraint. First, the last worst case busy period of J_{31} $WCBP(J_{31})$ is [0,8] as shown in Figure 5.4(c). Then, by Eq. (5.4), $\mathbb{J}^{\mathsf{F}}(J_{31})$ is $\{J_{11}, J_{12}, J_{21}, J_{31}\}$ as in Figure 5.4(d). Among jobs in $\mathbb{J}^{\mathsf{F}}(J_{31})$, since $\max(t_{11}^{\mathsf{S,real}}) < \min(t_{31}^{\mathsf{F,real}})$, J_{11} is in $\mathbb{J}^{\mathsf{F\text{-}det}}(\hat{J}_{31})$, where $\hat{J}_{31}$ is a terminal node of J_{31}. Likewise, J_{21} is an element of $\mathbb{J}^{\mathsf{F\text{-}det}}(\hat{J}_{31})$, also. Thus, J_{11} and J_{21} becomes deterministic predecessor of $\hat{J}_{31}$, while other jobs in $\mathbb{J}^{\mathsf{S}}(J_{31})$, i.e., J_{12} and J_{31}, are non-deterministic predecessors of $\hat{J}_{31}$ as shown in Figure 5.4(d).

Lastly, if J_{ij} has a producer-consumer constraint, i.e, Eq. (3.3), 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., $t_{i'j'}^{\mathsf{F,sim}} \leq t_{ij}^{\mathsf{S,sim}}$. 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 Figure 5.5 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.

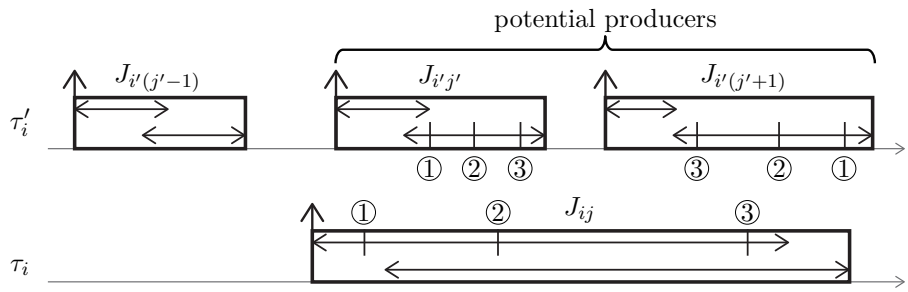


Figure 5.5 Potential producers

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:

$$\mathbb{J}^{\mathsf{P}}(J_{ij}) = \left(\bigcup_{\forall \text{potential producers } J_{i'j'}\text{s}} \mathbb{J}^{\mathsf{F}}(J_{i'j'}) \right) \cup \mathbb{J}^{\mathsf{S}}(J_{ij}) \tag{5.7}$$

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:

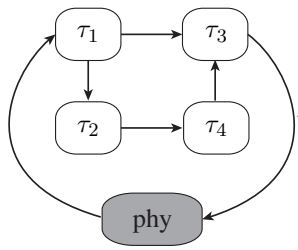
$$\begin{aligned} \mathbb{J}^{\mathsf{P\text{-}det}}(J_{ij}) =& \{J_{kl} | J_{kl} \in \mathbb{J}^{\mathsf{P}}(J_{ij}), \max(t_{kl}^{\mathsf{S,real}}) < \min(t_{ij}^{\mathsf{S,real}})\} \cup \{J_{i'(j'-1)}\} \\ & \text{where } \max(t_{i'(j'-1)}^{\mathsf{F,real}}) \leq \min(t_{ij}^{\mathsf{S,real}}) < \max(t_{i'j'}^{\mathsf{F,real}}). \end{aligned} \tag{5.8}$$

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:

$$\mathbb{J}^{\mathsf{P\text{-}nodet}}(J_{ij}) = \mathbb{J}^{\mathsf{P}}(J_{ij}) - \mathbb{J}^{\mathsf{P\text{-}det}}(J_{ij}). \tag{5.9}$$

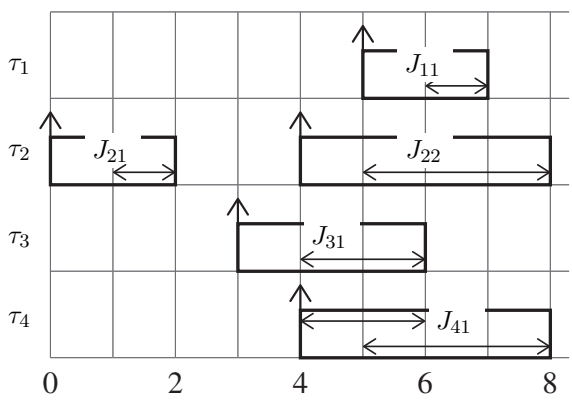
We explain how to set precedence relations for a job who has a producer-consumer constraint with an example. Let us use the example in Figure 5.6(a) and (b) assuming the priority of the lower index task is higher than that of the higher index task on ECU_1 and ECU_2, respectively. For all the jobs during one hyperperiod of the four tasks as in Figure 5.3(c), we can compute their start time and finish time ranges by considering the priority-based scheduling with the best and worst case execution times. In this example, we construct the offline guider for J_{41} who has the producer-consumer constraint. Since $\tau_2 \rightarrow \tau_4$ and $t_{22}^{\mathsf{F,real}}$ overlaps with $t_{41}^{\mathsf{S,real}}$ as in Figure 5.3(c), we cannot deterministically determine the data producer of J_{41}. Since J_{22} is a potential producer of J_{41}, $\mathbb{J}^{\mathsf{P}}(J_{41})$ contains $\mathbb{J}^{\mathsf{F}}(J_{22})$ and $\mathbb{J}^{\mathsf{S}}(J_{41})$ by Eq. (5.7). By Eq. (5.4), J_{11} and J_{22} are elements of $\mathbb{J}^{\mathsf{F}}(J_{22})$, and by Eq. (5.1), J_{31} is an element of $\mathbb{J}^{\mathsf{S}}(J_{41})$ as in Figure 5.3(d). Then, among them, by Eq. (5.8), J_{22} becomes a deterministic



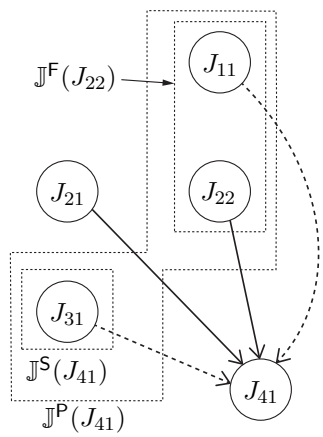
(a) Control system

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

(b) Task parameters



(c) Execution windows

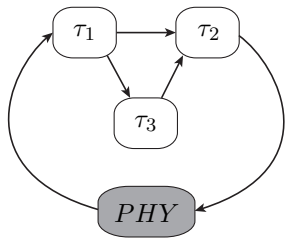


(d) Precedence relations for the producer-consumer constraint of J_{41}

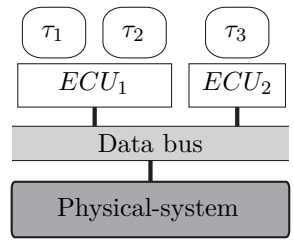
Figure 5.6 Precedence relations for the producer-consumer constraint

predecessor of J_{41}. Also, note that J_{21} is a deterministic predecessor of J_{41} since J_{21} is the last job just before the potential producers of J_{41} as in Eq. (5.8). Then, by Eq. (5.9), the remaining jobs in $\mathbb{J}^{\mathsf{P}}(J_{41})$, i.e., J_{11} and J_{31} are non-deterministic predecessors of J_{41} as in the figure.

Now, we explain how to construct the offline guider overall with an example. Let us use the example in Figure 5.7(a) assuming the RM scheduling policy for each ECU. Then, we have to construct the offline guider for four jobs, i.e., J_{11}, J_{12}, J_{21}, and J_{31}, in the hyperperiod as shown in execution windows in Figure 5.7(b). Let us consider J_{11} first. Since J_{10} and J_{11} are two consecutive jobs, J_{10} is a deterministic predecessor of J_{11}. Then, since J_{11} has only a physical-read constraint as marked by R, we consider $\mathbb{J}^{\mathsf{S}}(J_{11})$. In this case, since $\mathbb{J}^{\mathsf{S}}(J_{11})$ is an empty set, no predecessor is added to J_{11}. Likewise, J_{11} is only a deterministic predecessors of J_{12}. Now, let us consider J_{21}. First, because of the consecutiveness of jobs, J_{20} is a deterministic predecessor of J_{21}. Then, let us consider which constraints J_{21} has. Since $\tau_2 \rightarrow PHY$, J_{21} has a physical-write constraint. Meanwhile, since $\tau_1 \rightarrow \tau_2$ and $\tau_3 \rightarrow \tau_2$, J_{21} has producer-consumer constraints. Among two types of constraints, let us look at the physical-write constraint. For this constraint, a terminal node $\hat{J}_{21}$ is needed with W mark as in the figure. Then, we consider $\mathbb{J}^{\mathsf{F}}(J_{21})$ whose elements are J_{11}, J_{12}, and J_{21}. Among jobs in $\mathbb{J}^{\mathsf{F}}(J_{21})$, by Eqs. (5.5) and (5.6), J_{11} becomes a deterministic predecessor, and J_{12} and J_{21} are non-deterministic predecessor of $\hat{J}_{21}$. For the producer-consumer constraints of J_{21}, let us first consider the producer-consumer constraint generated by τ_1. Since the range



Control design

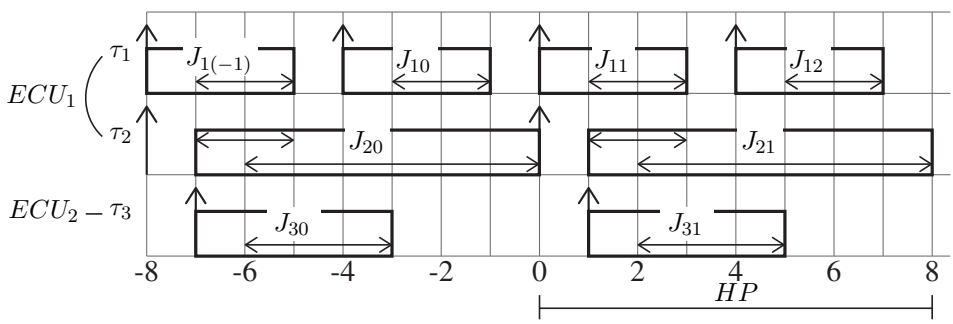


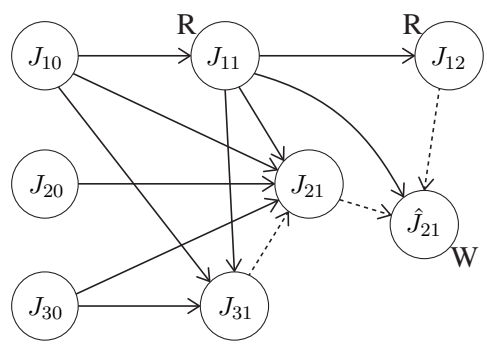
Task-ECU mapping

	Φ_i	P_i	C_i^{best}	C_i^{worst}
τ_1	0	4	1	3
τ_2	0	8	1	2
τ_3	1	8	1	2

Task parameters

(a) Example cyber-system





(b) Execution windows and the offline guider

Figure 5.7 Overall process of constructing the offline guider

of $t_{11}^{\mathsf{F},\mathsf{real}}$ overlaps with $t_{21}^{\mathsf{S},\mathsf{real}}$, J_{11} is a potential producer of J_{21}. In this case, J_{11} is an element of $\mathbb{J}^{\mathsf{P}}(J_{21})$, because $\mathbb{J}^{\mathsf{F}}(J_{11})$ and $\mathbb{J}^{\mathsf{S}}(J_{21})$ have J_{11} as their element, respectively. Likewise, for the producer-consumer constraint of J_{21} generated by τ_3, J_{31} becomes an element of $\mathbb{J}^{\mathsf{P}}(J_{21})$. Then, by Eqs. (5.8) and (5.9), J_{11}, J_{10}, and J_{30} become deterministic predecessors, while J_{31} becomes a non-deterministic predecessor of J_{21}. Lastly, let us consider J_{31}. Because of the consecutiveness of jobs, J_{30} is a deterministic predecessor of J_{31}. Considering constraints of J_{31}, since $\tau_1 \rightarrow \tau_3$, J_{31} has a producer-consumer constraint only. Since $t_{11}^{\mathsf{F},\mathsf{real}}$ and $t_{31}^{\mathsf{S},\mathsf{real}}$ are overlapped, by Eq. (5.7), J_{11} is only an element of $\mathbb{J}^{\mathsf{P}}(J_{31})$. Then, by Eqs. (5.8), J_{11} and J_{10} are deterministic predecessors of J_{31} as shown in Figure 5.7(b). In this way, we construct the offline guider within the hyperperiod.

5.4 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 $[\min(t_{ij}^{\mathsf{S},\mathsf{real}}), \max(t_{ij}^{\mathsf{S},\mathsf{real}})]$ and finish time range $[\min(t_{ij}^{\mathsf{F},\mathsf{real}}), \max(t_{ij}^{\mathsf{F},\mathsf{real}})]$ of every job $J_{ij} \in OJPG$ con-

sidering 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 5.2 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., $t_{ij}^{\mathsf{S,sim}} \geq t_{ij}^{\mathsf{S,real}}$ in Eq. (3.1).

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 [49]. Thus, we assign the effective deadline $t_{ij}^{\mathsf{D,sim}}$ to each job $J_{ij} \in OJPG$ as follows:

$$t_{ij}^{\mathsf{D,sim}} = \begin{cases} \min(t_{ij}^{\mathsf{F,real}}) & \text{for } \hat{J}_{ij} \\ \infty & \text{otherwise} \end{cases}, \tag{5.10}$$

$$t_{ij}^{\mathsf{D,sim}} = \min\left(t_{ij}^{\mathsf{D,sim}}, \min_{\forall J_{kl} \in \mathbb{J}^{\mathsf{succ\text{-}det}}(J_{ij})}\left(t_{kl}^{\mathsf{D,sim}}\right)\right) \tag{5.11}$$

where $\mathbb{J}^{\mathsf{succ\text{-}det}}(J_{ij})$ is the set of deterministic successors of J_{ij}. In Eq. (5.10), 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. (5.11), 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 5.2) 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 5.4). 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. (5.2) and (5.8) was not met, our online algorithm checks the condition again with the updated $\max(t_{kl}^{\mathsf{S,real}})$ and $\min(t_{ij}^{\mathsf{S,real}})$. If it turns out that

$$\max(t_{kl}^{\mathsf{S,real}}) < \min(t_{ij}^{\mathsf{S,real}}), \tag{5.12}$$

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

$$\min(t_{kl}^{\mathsf{S,real}}) \geq \max(t_{ij}^{\mathsf{S,real}}), \tag{5.13}$$

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. (5.5) 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

$$\max(t_{kl}^{\mathsf{S,real}}) < \min(t_{ij}^{\mathsf{F,real}}), \tag{5.14}$$

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

$$\min(t_{kl}^{\mathsf{S,real}}) \geq \max(t_{ij}^{\mathsf{F,real}}), \tag{5.15}$$

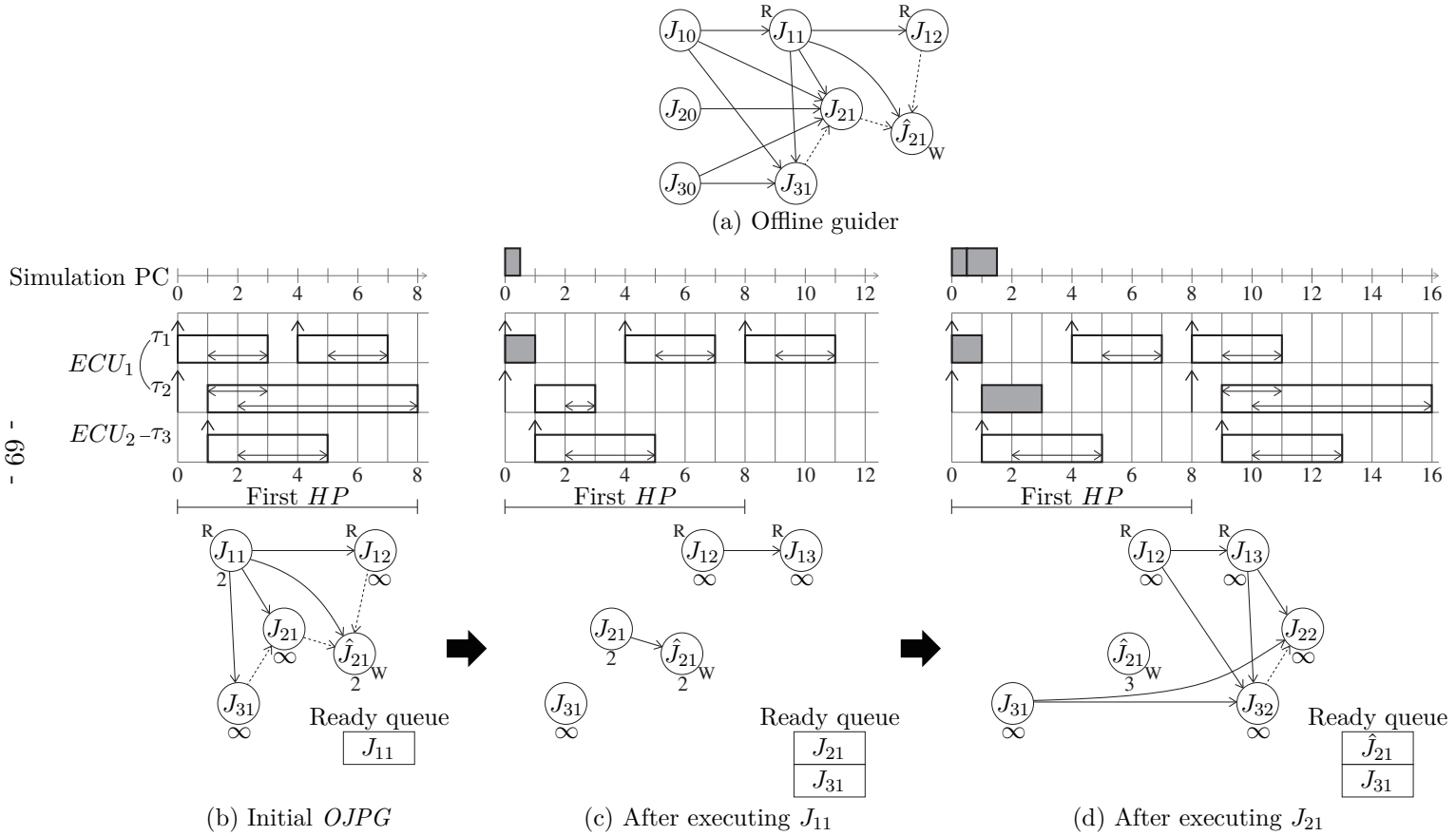
it is clear that $t_{kl}^{\mathsf{S,real}}$ cannot be earlier than $t_{ij}^{\mathsf{F,real}}$. Thus, we remove the non-deterministic edge from J_{kl} to J_{ij}. Otherwise, the edge remains as non-

deterministic until the non-determinism is resolved.

Our online algorithm also updates the effective deadlines of jobs in *OJPG* by using the updated $\min(t_{ij}^{\mathsf{F},\mathsf{real}})$s for $\hat{J}_{ij}$ in Eq. (5.10) and newly changed deterministic edges in Eq. (5.11).

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.

Now, we explain how the online progressive scheduling algorithm works with an example assuming $e_{ij}^{\mathsf{real}} = 2 \cdot e_{ij}^{\mathsf{sim}}$. Let us assume that the offline guider is given as in Figure 5.8(a) and the simulation start time is time 0. Then, from the offline guider, all the jobs who are in the first HP are copied to *OJPG* as shown in Figure 5.8(b). Also, all the associated deterministic and non-deterministic precedence edges in the offline guider are copied to *OJPG*. With this initial *OJPG*, we first calculate the effective deadline of each job which is denoted under each node in *OJPG*. Since $\min(t_{21}^{\mathsf{F},\mathsf{real}})$ is 2, by Eq. (5.10), the initial value of the effective deadline of $\hat{J}_{21}$ is 2, and initial values of effective deadlines of all other jobs are ∞. Then, by Eq. (5.11), the effective deadline of J_{11} is finally updated into 2. In this situation, J_{11} is the only job who can be executed on the simulation PC, since J_{11} has no deterministic predecessor and the physical-read constraint of J_{11} is met. Thus, J_{11} is in the simulation ready queue at time 0. Let us assume that $e_{11}^{\mathsf{sim}} = 0.5$ and hence



(a) Offline guider

(b) Initial $OJPG$

(c) After executing J_{11}

(d) After executing J_{21}

Figure 5.8 Overall process of the online progressive scheduling

$e_{11}^{\mathsf{real}} = 1$. After J_{11} is executed on the PC, a new job J_{13} is added to $OJPG$ with the associated deterministic and non-deterministic precedence edges as in Figure 5.8(c). After that, start time and finish time ranges are updated. More specifically, using $e_{11}^{\mathsf{real}} = 1$, J_{21}'s start time and finish time range is recalculated as in Figure 5.8(c). Then, we clearly know J_{12} is not a predecessor of $\hat{J}_{21}$, while J_{21} is a deterministic predecessor of $\hat{J}_{21}$ by Eqs. (5.14) and (5.15). Also we can know J_{31} is not a predecessor of J_{21} by Eq. (5.13). Lastly, the effective deadline of each job is updated. Now, at time 0.5, J_{21} and J_{31} are in the ready queue. Out of them, J_{21} is selected since $t_{21}^{\mathsf{D},\mathsf{sim}} < t_{31}^{\mathsf{D},\mathsf{sim}}$. After J_{21} is executed on the PC, a new job J_{22} is added to $OJPG$. Note that J_{32} is also added to $OJPG$ since J_{32} is a non-deterministic predecessor of J_{22} as in Figure 5.8(a). Then, assuming that $e_{21}^{\mathsf{sim}} = 1$, we update start time and finish time ranges, resolve non-determinism, and update effective deadlines as shown in Figure 5.8(d). In this way, our online progressive scheduling algorithm works.

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. From now on, we prove this theoretically.

Lemma 5.1. *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. (5.2) and/or the producer-consumer constraints as in Eq. (5.8). This is because the physical-write constraint in Eq. (5.5) 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 $\max(t_{kl}^{\mathsf{S},\mathsf{real}}) < \min(t_{ij}^{\mathsf{S},\mathsf{real}})$ due to Eqs. (5.2) and/or (5.8) in the offline guider construction and also Eq. (5.12) in the online resolution of non-determinism. Thus, it is clear that $t_{kl}^{\mathsf{S},\mathsf{real}} < t_{ij}^{\mathsf{S},\mathsf{real}}$. Also, in the cycle, there should be a path from J_{ij} to J_{kl} consisting of deterministic edges. It now implies $\max(t_{ij}^{\mathsf{S},\mathsf{real}}) < \min(t_{kl}^{\mathsf{S},\mathsf{real}})$ and hence $t_{ij}^{\mathsf{S},\mathsf{real}} < t_{kl}^{\mathsf{S},\mathsf{real}}$. It is a contradiction. □

Lemma 5.2. *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 $t_{ij}^{\mathsf{S},\mathsf{real}}$ 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 $t_{ij}^{\mathsf{S},\mathsf{real}}$ 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 $\min(t_{kl}^{\mathsf{S},\mathsf{real}})$. Since J_{kl} would delay J_{ij}'s start, its priority is higher than J_{ij} on the same ECU and its $\min(t_{kl}^{\mathsf{S},\mathsf{real}})$ is prior to $\min(t_{ij}^{\mathsf{S},\mathsf{real}})$. For such J_{kl}, if its start time $t_{kl}^{\mathsf{S},\mathsf{real}}$ 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 $\min(t_{mn}^{\mathsf{S},\mathsf{real}}) < \min(t_{kl}^{\mathsf{S},\mathsf{real}})$. 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. (5.12). It is a contradiction. □

Lemma 5.3. *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. (5.14). It is a contradiction. □

Lemma 5.4. *At any time point of our simulation, for a job J_{ij} in OJPG with a producer-consumer constraint due to $\tau_{i'} \rightarrow \tau_i$, 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. (5.7) 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 5.2. **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. (5.8) and (5.9). 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. (5.12). 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. (5.14), 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 5.3. **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 5.1. *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 5.1, our algorithm can continue simulating jobs in *OJPG* without being stuck in a cycle. Also, by Lemma 5.2 and Lemma 5.4, our algorithm can simulate each job with the same data as the real cyber-system, using the tagged data read. In addition, by Lemma 5.3, our online algorithm can assign the real finish time $t_{ij}^{\mathsf{F,real}}$ as the effective deadline of the simulated job J_{ij} with physical-write, using Eqs. (5.10) and (5.11). 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 tagged 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. □

5.5 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

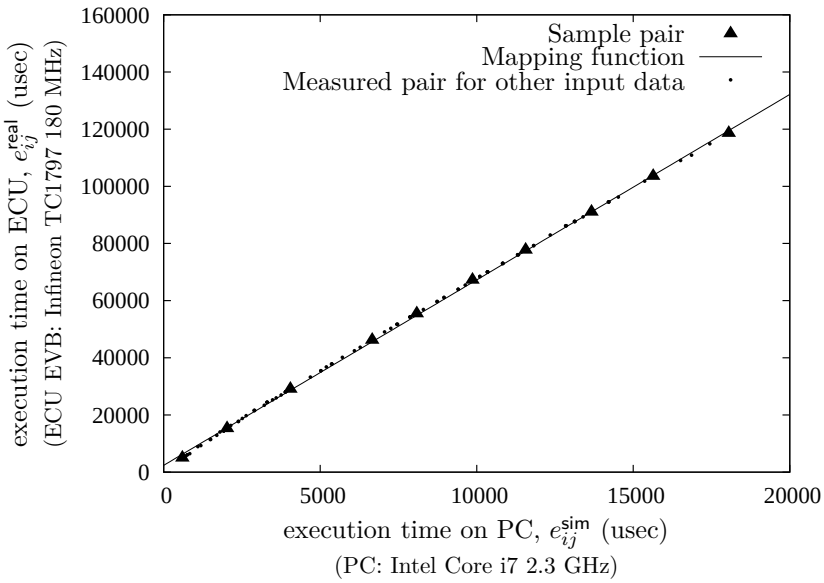


Figure 5.9 Execution times for a pulse code modulation task

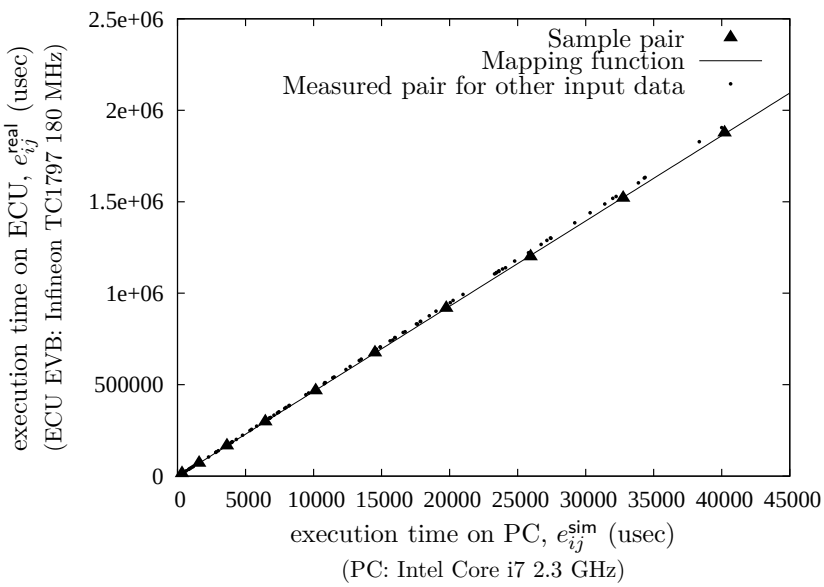


Figure 5.10 Execution times for a data compression task

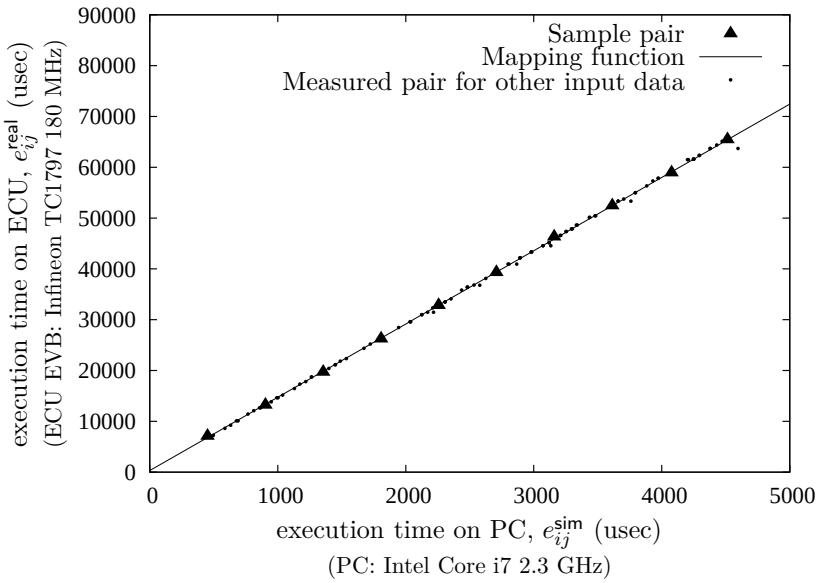


Figure 5.11 Execution times for a fast cosine transform task

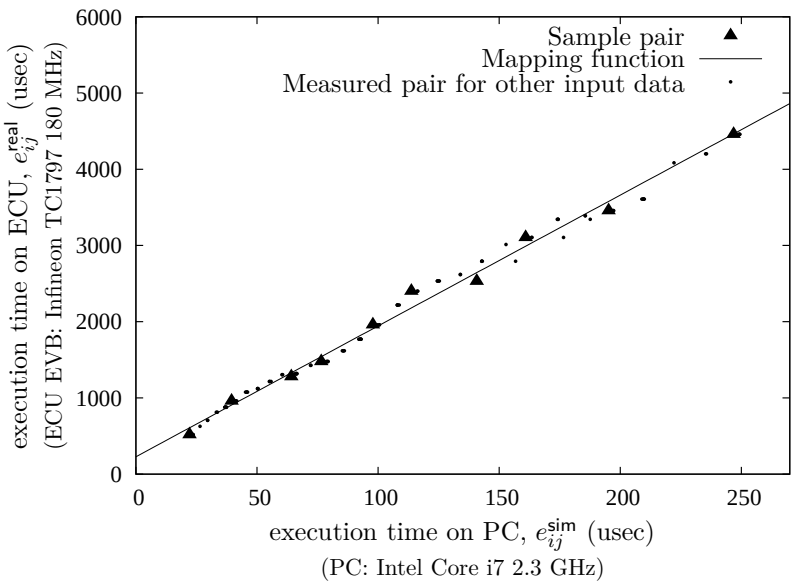


Figure 5.12 Execution times for a image processing task

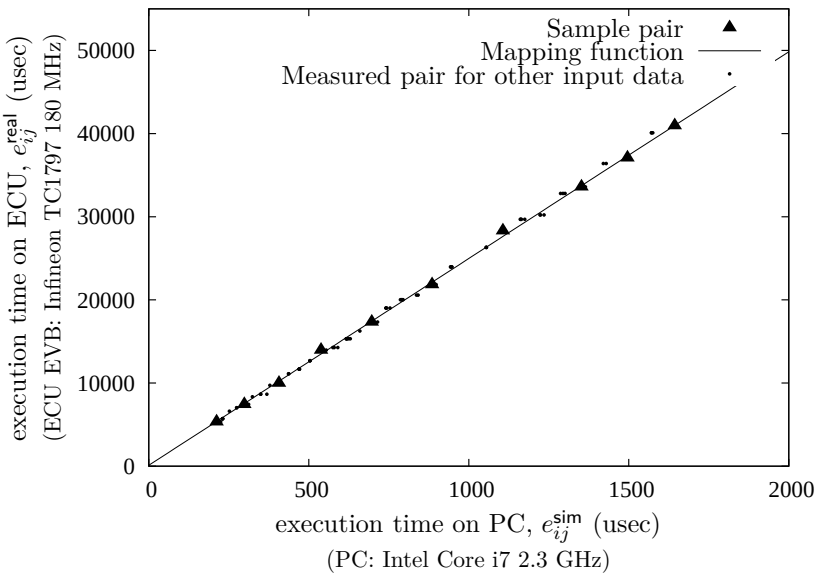


Figure 5.13 Execution times for a matrix multiplication task

Table 5.1 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 (%)

use the ECU EVB (evaluation board) to find the correlation between e_{ij}^{sim} and e_{ij}^{real}. Figure 5.9, 5.10, 5.11, 5.12, and 5.13 describe relation between the PC execution time and the ECU execution time for five example tasks. The black triangles in each figure are 10 sample pairs of $(e_{ij}^{\mathsf{sim}}, e_{ij}^{\mathsf{real}})$ we measured for each 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 each figure. The small dots in each 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}. Table 5.1 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 dissertation. It is our future work.

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

5.5.1 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}:

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

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 $t_{ij}^{\mathsf{S},\mathsf{real}}$ 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 [11]. 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 $t_{ij}^{\mathsf{S},\mathsf{sim}} \geq t_{ij}^{\mathsf{S},\mathsf{real}}$ 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. (3.1) and (3.3).

- **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 [49]. Thus, **Ideal** uses this optimal scheduling approach.

All the above four approaches guarantee that each simulated job uses the same

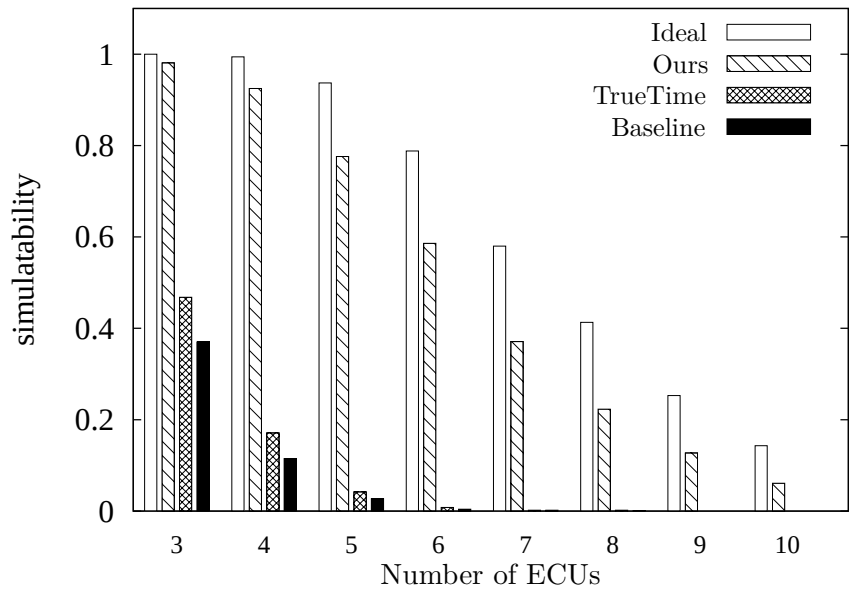


Figure 5.14 Simulatability as changing the number of the ECUs

physical data and producer data as the real jobs. Thus, if all the simulated jobs with physical-writes can be finished satisfying $t_{ij}^{\mathsf{F,sim}} \leq t_{ij}^{\mathsf{F,real}}$ in Eq. (3.2), they can write the same physical data at the same time as the real cyber-system using the "Tagged 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.

Figure 5.14 compares the simulatabilities of the above four approaches as changing the number of the ECUs in the target system. The X-axis represents the number of the ECUs in the target system, and the Y-axis represents simulatability. For each number of the ECUs, we synthesize 1000 target sys-

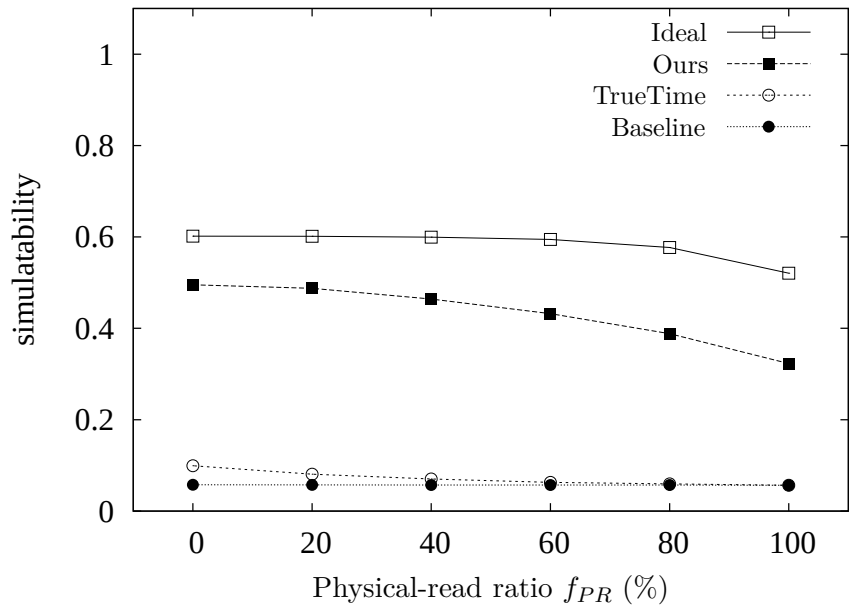


Figure 5.15 Simulatability as changing the physical-read ratio f_{PR}

tems with the aforementioned parameters. As the number of ECUs increases, simulatabilities of the all approaches decreases. Compared with **Baseline**, by enjoying the start time freedom, **TrueTime** can only achieve small improvement. However, if we execute jobs applying both freedoms of start times and job execution order, i.e., **Ours**, then the simulatability increases dramatically compared to **Baseline** and **TrueTime**. Also, we can know that **Ours** is not bad compared to **Ideal** which is an unrealistic approach.

Figure 5.15 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 enjoy-

ing 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.

Figure 5.16 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

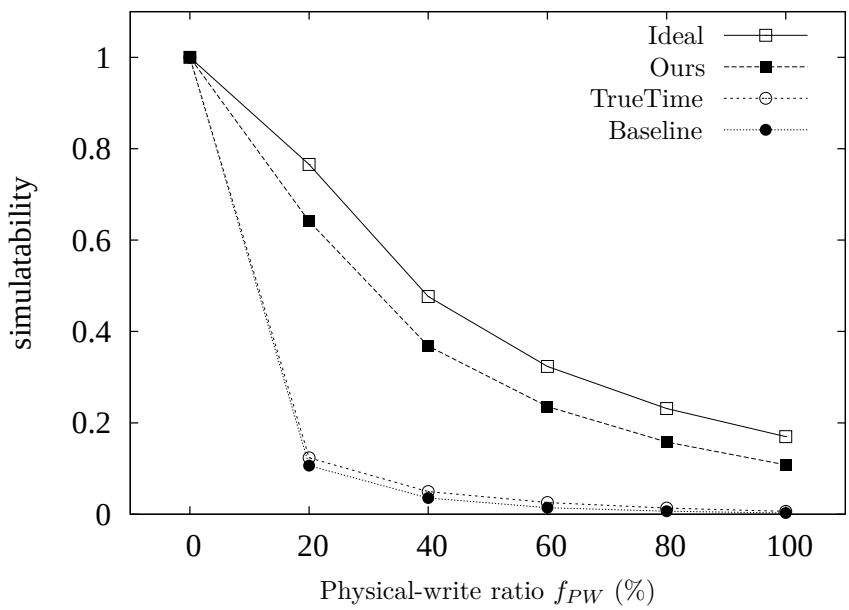


Figure 5.16 Simulatability as changing the physical-write ratio f_{PW}

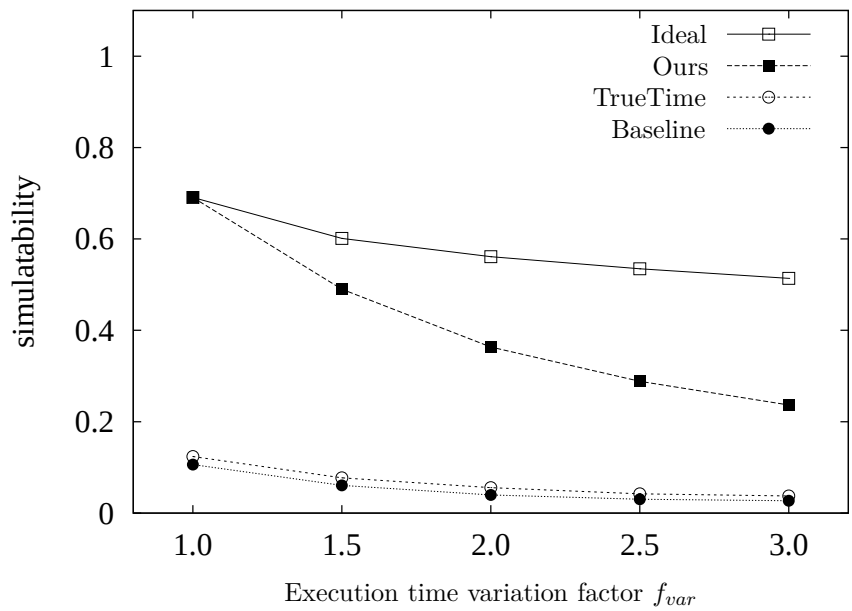
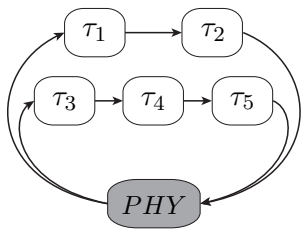


Figure 5.17 Simulatability as changing the execution time variation factor f_{var}

cyber-system. Again, **Ours** shows a comparable simulatability with **Ideal**.

Figure 5.17 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 $C_i^{\mathsf{best}} = C_i^{\mathsf{worst}}$ 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**.



(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

Figure 5.18 Cyber-system to be simulated and implemented

5.5.2 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 [45]. Each ECU is equipped with an Infineon TC1797 180 MHz microprocessor [43]. The first ECU performs a CC (Cruise-Control) function by executing two periodic tasks τ_1 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 Figure 5.18(a). The parameters of the five tasks are given in Figure 5.18(b). The tasks on each ECU are scheduled by the Rate-Monotonic scheduling policy.

As the physical-system, we use CarSim RT [42], which is a commercial real-time vehicle dynamics simulator. In CarSim RT, for the vehicle body model, we use "B-Class, Sports Car Sprung Mass" model. In the model, specifications of the body are as follows:

- Wheelbase: 2330 mm,
- Width: 1750 mm,
- Height: 1200 mm,
- Rear overhang: 530 mm,
- Front tire diameter: 660 mm,
- Rear tire diameter: 620 mm, and
- Mass: 1020 kg.

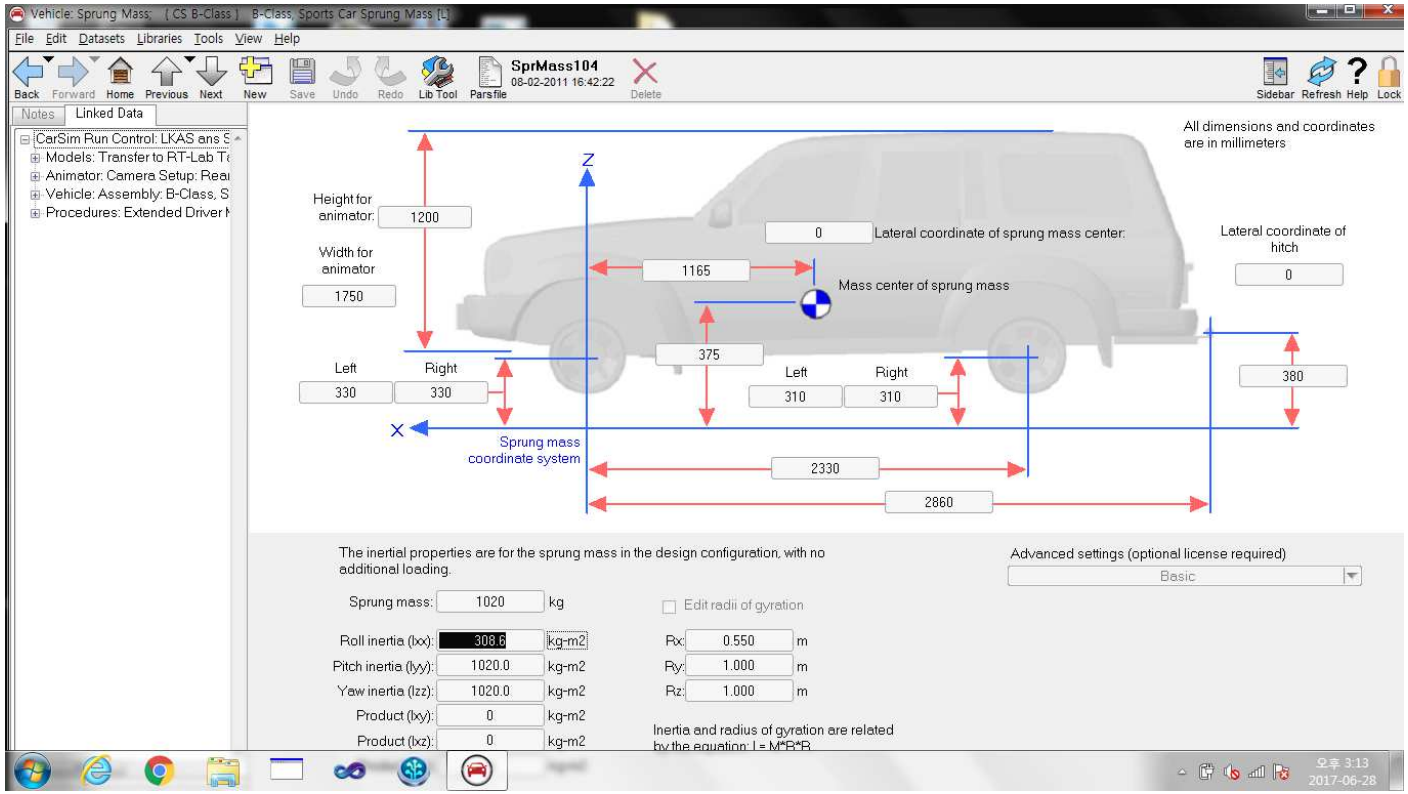


Figure 5.19 Vehicle body model

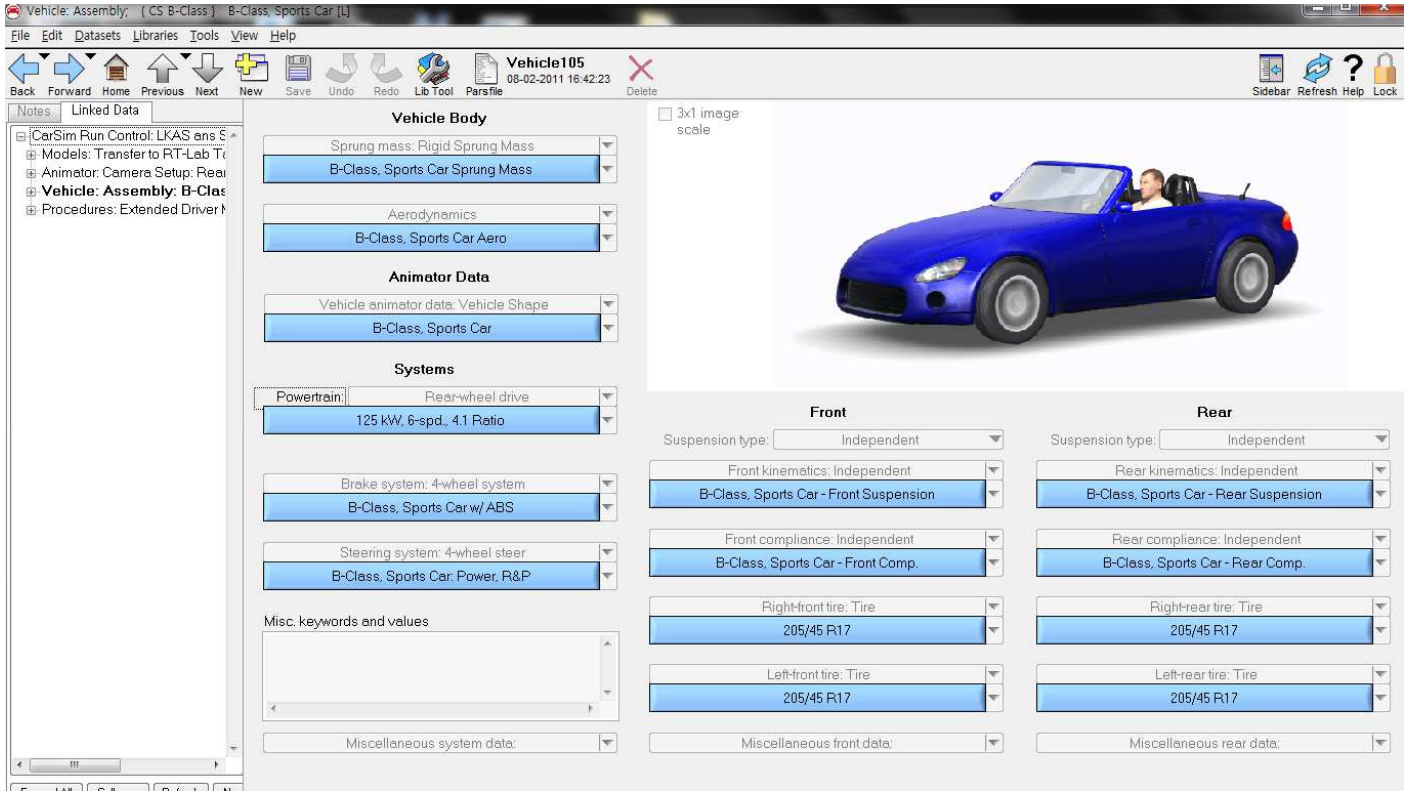


Figure 5.20 Vehicle drive system model

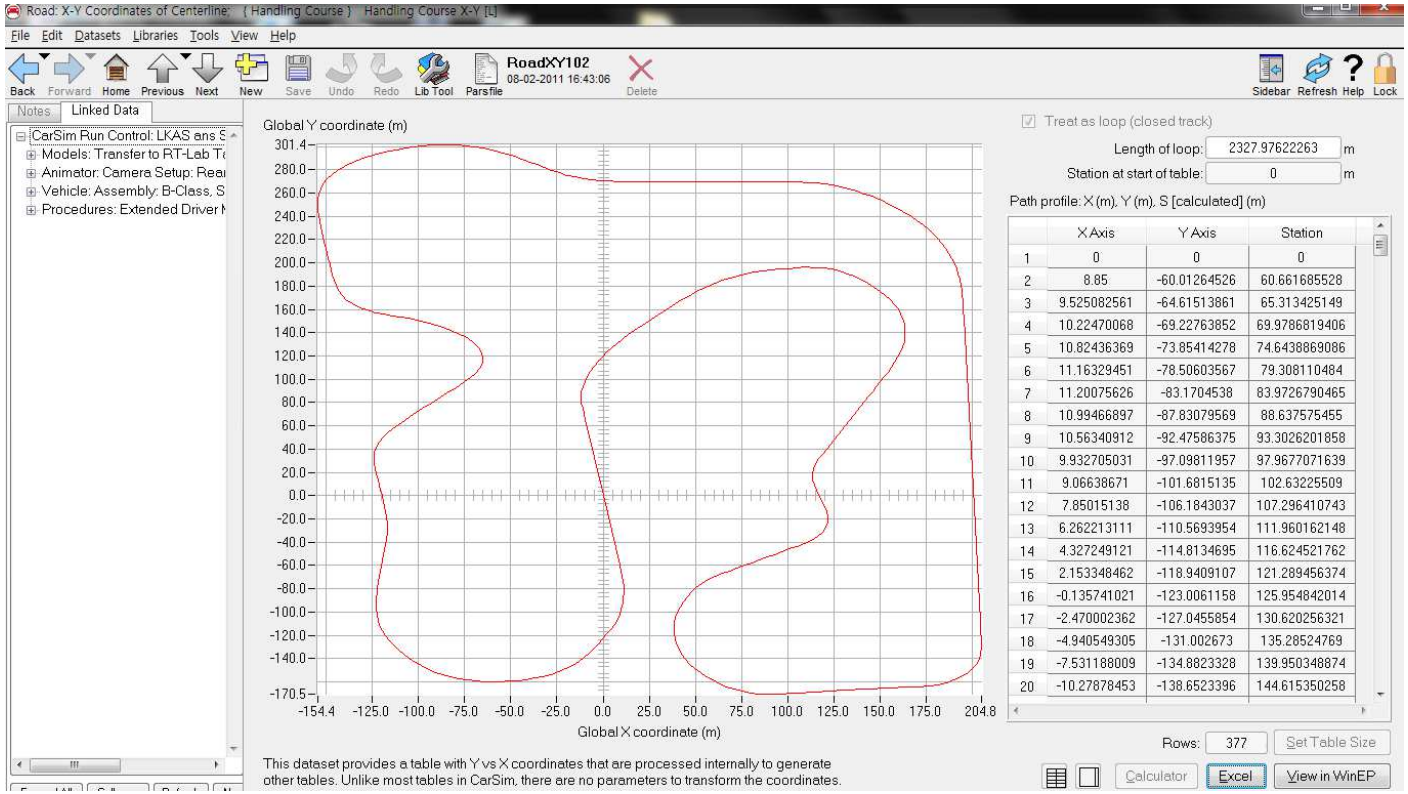


Figure 5.21 Road model

For more details, see Figure 5.19. For the vehicle drive system model, we use "B-Class, Sports Car" model. In the model, the drive system is as follows:

- Engine: rear-wheel drive, 125 kW (167.5 HP),
- Transmission: six-speed automatic,
- Brake: ABS, and
- Suspension: independent type.

Figure 5.20 describes more information about the vehicle drive system model. Lastly, for the road model, we use "RoadXY102" model. The road is a closed-loop, its total length is 2328 m, and its width is 7.32 m. Figure 5.21 describes top-view of the road with the coordinate plain. In the figure, (0,0) is the start position.

Figure 5.22 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 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. 5.23(a) and (b) show statistics on the

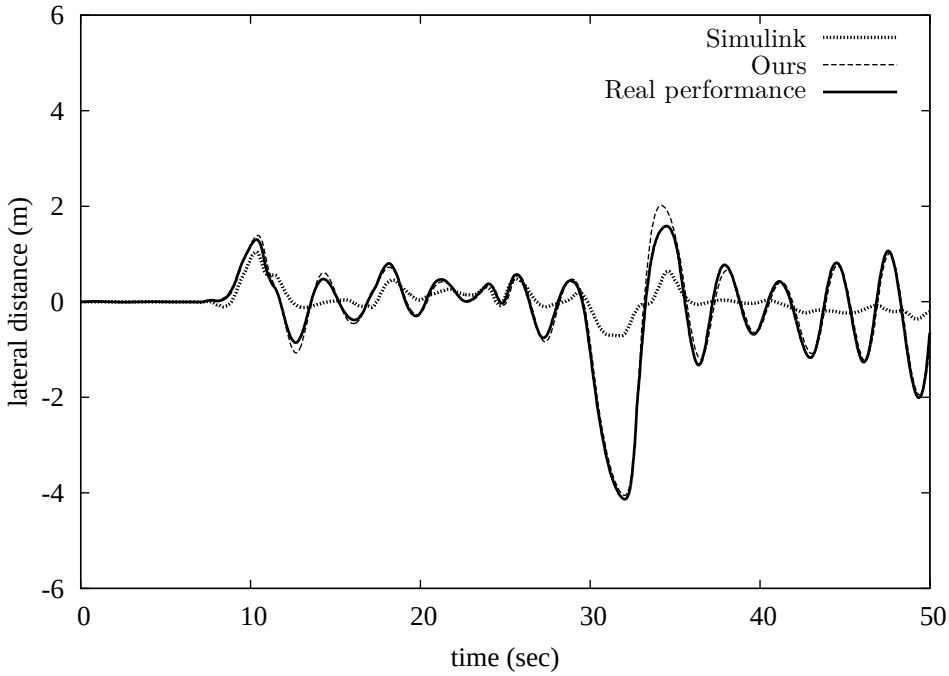
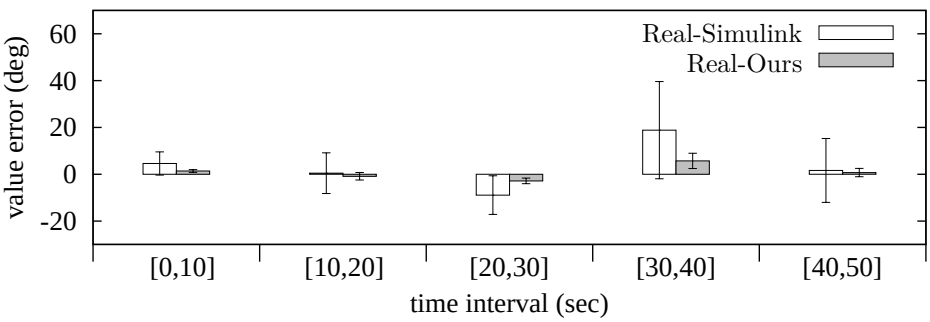
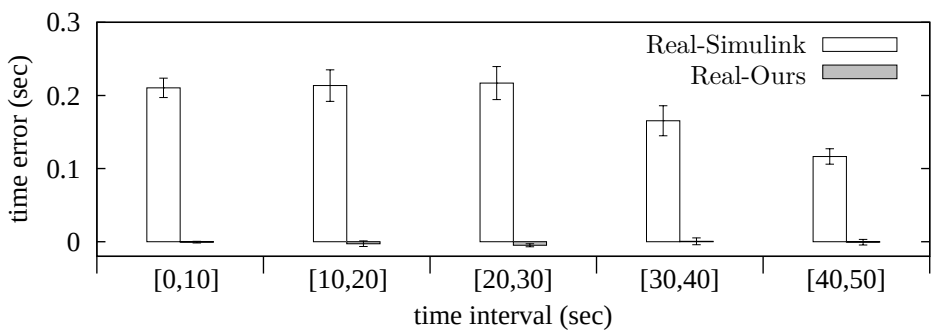


Figure 5.22 Control performance comparison



(a) Value error



(b) Time error

Figure 5.23 Statistics on value errors and time errors

value errors and time errors, respectively, by Simulink and Ours compared with the real performance. In the figures, the 50 sec experimental duration of Figure 5.22 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.

Chapter 6
Practical Discussions

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

6.1 Data Exchange Delay

In this section, 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 TDMA bus, the general practice of the real-time embedded system industry is to assign a dedicated slot to every data content that needs to be exchanged among ECUs and the physical-system [45]. 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 $t_{ij}^{\mathsf{F},\mathsf{real}}$, we can determine the wait time $\delta_{ij}^{\mathsf{F},\mathsf{real}}$ 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

$$R(t_{ij}^{\mathsf{F},\mathsf{real}}) = t_{ij}^{\mathsf{F},\mathsf{real}} + \delta_{ij}^{\mathsf{F},\mathsf{real}} + T_s \tag{6.1}$$

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 [52] 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. (3.1), 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. (3.2). 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 $t_{ij}^{\mathsf{F,real}}$.

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. (6.1). 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 $t_{i'j'}^{\mathsf{F},\mathsf{real}}$s of potential producer jobs to their corresponding reception times $R(t_{i'j'}^{\mathsf{F},\mathsf{real}})$s at τ_i's ECU. Such transformed finish times are used to determine J_{ij}'s producer job $J_{i'j'}$ as in Figure 5.5. Once the producer job $J_{i'j'}$ is such correctly identified, the simulation PC can schedule $J_{i'j'}$ and J_{ij} satisfying $t_{i'j'}^{\mathsf{F},\mathsf{sim}} \leq t_{ij}^{\mathsf{S},\mathsf{sim}}$ in Eq. (3.3), and hence it can start J_{ij} at $t_{ij}^{\mathsf{S},\mathsf{sim}}$ with the correct data produced by $J_{i'j'}$ at its finish time $t_{i'j'}^{\mathsf{F},\mathsf{sim}}$.

6.2 Simulation Overhead

In this section, we discuss the time overhead of the proposed approach.

6.2.1 Offline Overhead

In the offline phase, the proposed approach constructs the offline guider. When we construct the guider, we consider all jobs in one hyperperiod, i.e., HP. Therefore, the time complexity of the guider construction is relative to the number of tasks n and HP. We can briefly express the time complexity as follows:

$$O(n \times HP).$$

In the worst-case, if periods of all tasks are different prime numbers, HP is the product of periods of the tasks, i.e., $\prod_{i=1}^{n} P_i$. Therefore, the worse-case time

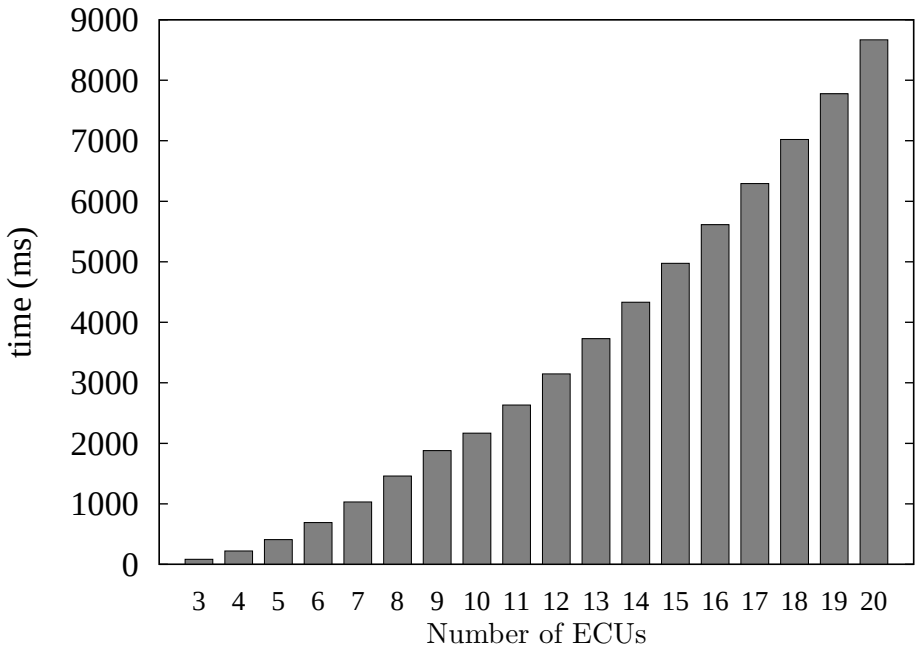


Figure 6.1 Simulator's offline overhead

complexity is exponential time like this:

$$O(2^n).$$

Although the worst-case time complexity is exponential, the actual time overhead does not grow exponentially as the size of the target system increases. This is because the actual period of each task in the cyber-system is usually decided the multiple of 5 or 10. To justify this, with synthesize cyber-systems, we measure the time overhead for constructing the offline guider as changing

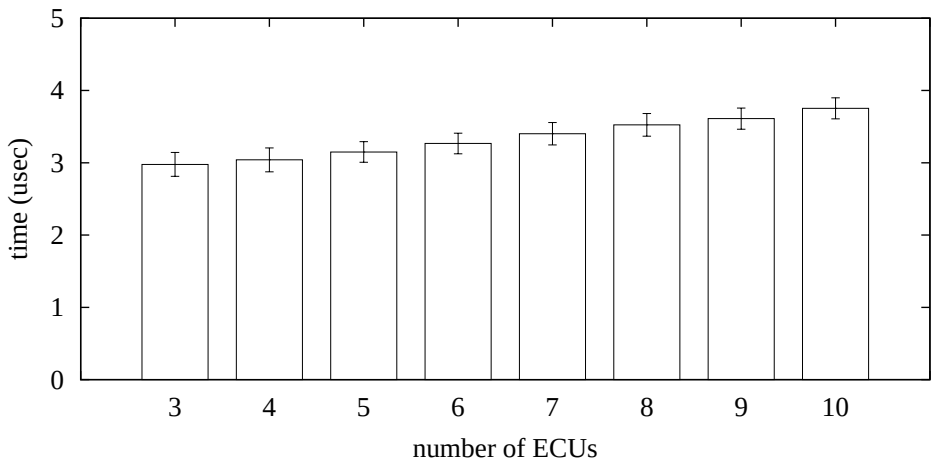


Figure 6.2 Simulator's online overhead

the number of the ECUs in the target system as shown in Figure 6.1. The X-axis represents the number of the ECUs in the target system, and the Y-axis represents the time overhead for constructing the offline guider. For each number of the ECUs, we synthesize 1000 target systems with the parameters introduced in Chapter 5.5.1. In the figure, we can find that the actual time overhead does not grow exponentially as the size of the target system increases. Especially, from 14-ECU system to 20-ECU system, the time overhead grows almost linearly. Moreover we can find that the actual time overhead stays under a few seconds even for a large cyber-system consisting of 20 ECUs.

6.2.2 Online Overhead

Our simulator implements the proposed online scheduling algorithm in an optimized way to minimize its online overhead. Figure 6.2 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.

6.3 Other Useful Features

Our simulator provides useful features for the actual design and implementation of the cyber-side of the cyber-physical systems. Here, we give just a brief introduction of them. Interested readers are referred to our open-source project [50] and demo video clip [51].

- Hybrid simulation: In the development process, some ECUs are implemented before others. Our simulator can support such a hybrid situation by simulating only un-implemented ECUs interacting with the real implemented ECUs and the real-working physical-system [53]. This hybrid simulation was presented and demonstrated at [54–57]

- 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.

Chapter 7
Extension for Multicore Simulation PC

At Chapter 4 and 5, we only use a singlecore processor of the simulation PC for executing simulated jobs. In this chapter, to further improve the real-time simulatability, we extend our approach so that we can utilize multicore of the simulation PC.

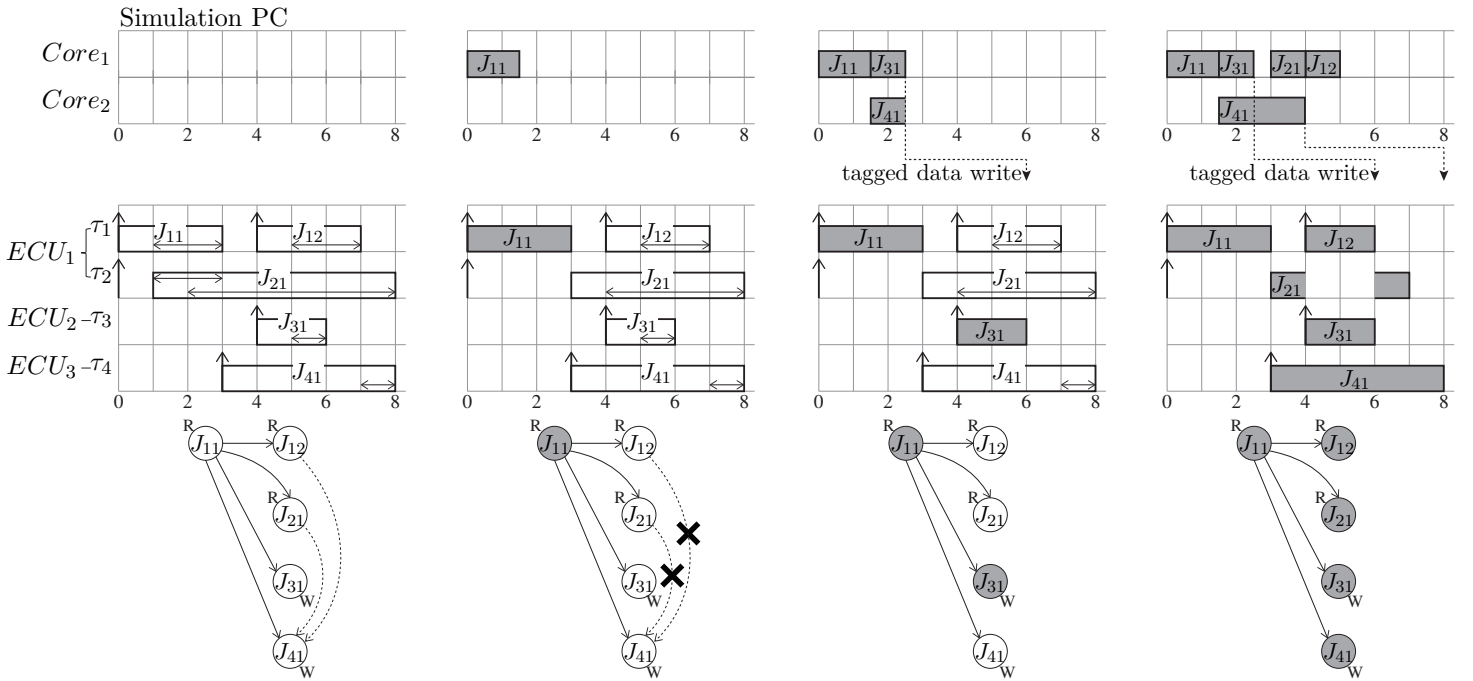
The description and assumptions on the real cyber-system, i.e., the simulation target, are same as those in Chapter 3.1. In addition to assumptions on the simulated cyber-system described in Chapter 3.2, we introduce the following assumptions for the simulation PC who has multicore processors:

- The simulation PC has two or more identical cores.
- The task migration cost between cores is negligible.

Next, even if we use multicore processors of the simulation PC, the procedure for constructing the offline guider is same as described in Chapter 5.3. Therefore, for the multicore PC, we only extend the online progressive scheduling algorithm for the singlecore PC introduced in Chapter 5.4. In the chapter, we use the preemptive EDF scheduling algorithm to execute simulated jobs which is known as the optimal scheduling policy on a singlecore processor. In the multicore domain, however, the preemptive EDF scheduling is not optimal, and the fluid scheduling algorithms are known to be optimal [68] for periodic

task sets. However, since there is no optimal scheduler for a set of jobs with two or more distinct deadlines on the multicore PC [68], we can know that there is no optimal scheduler for scheduling jobs in the offline guider. Moreover, since the fixed deadline and execution time of each task are needed for the fluid scheduling, we do not use the fluid scheduling for our online progressive scheduling. Instead, we use the preemptive "Global EDF" [69] which is an extension of the preemptive EDF on a singlecore processor. The comparison with various scheduling algorithms in terms of the real-time simulatability is beyond the scope of this dissertation. In the future work, we plan to study various scheduling algorithms for multicore processors to further improve the real-time simulatability.

From now on, we overview our online progressive scheduling for a multicore PC with an example in Figure 7.1 assuming $e_{ij}^{\mathsf{real}} = 2 \cdot e_{ij}^{\mathsf{sim}}$ and the number of cores on the simulation PC is 2, i.e., $Core_1$ and $Core_2$. In the beginning, the simulation PC starts with the offline guider as in Figure 7.1(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} on $Core_1$ as in Figure 7.1(b). Let us assume that $e_{11}^{\mathsf{sim}} = 1.5$ and hence $e_{11}^{\mathsf{real}} = 3$. Using $e_{11}^{\mathsf{real}} = 3$, it can recalculate J_{21}'s finish time range as in Figure 7.1(b). Also, let us assume that we know that the non-deterministic precedence edges from J_{12} and J_{21} to J_{41} are not needed anymore. Then, 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.



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

Figure 7.1 Online progressive scheduling on multicore

Thus, only J_{31} and J_{41} are ready for simulation. At time 1.5, two cores on the simulation PC are idle, therefore, the simulation PC simulates J_{31} and J_{41} at the same time. After completing J_{31} at time 2.5, the simulation PC knows $e_{31}^{\mathsf{sim}} = 1$ and hence $e_{31}^{\mathsf{real}} = 2$. Using $e_{31}^{\mathsf{real}} = 2$, it computes the schedule on ECU_2 and knows the real finish time of J_{31} is 6 as in Figure 7.1(c). Thus, it plans the tagged data write operation that will happen at 6. Note that J_{41} is being executed at time 2.5 on $Core_2$. At time 2.5, since $Core_1$ becomes idle, the simulation PC tries to find the next job who can be executed. However, J_{12} and J_{21} still do not meet the physical-read constraint yet, $Core_1$ remains idle at time 2.5. As time goes by, at time 3, now J_{21} meets its physical-read constraint, thus the simulation PC simulates J_{21} on $Core_1$. If we assume that $e_{41}^{\mathsf{sim}} = 2.5$, J_{41} finishes its execution on $Core_2$ at time 4. Then, Using $e_{41}^{\mathsf{real}} = 5$, the simulation PC computes the schedule on ECU_3 and knows the real finish time of J_{41} is 8 as in Figure 7.1(d). Thus, it plans the tagged data write operation that will happen at 8. Like this, it progresses simulating jobs on multicore processors while resolving non-determinism guided by the offline guider. The other details of the online progressive scheduling algorithm for the multicore PC, e.g., to assign effective deadlines and to resolve the non-deterministic precedence edges, are same as described in Chapter 5.4.

To evaluate the proposed algorithm, we synthesize 1000 cyber-systems and compare the "simulatability", i.e., how many of them are correctly simulated, with other three approaches introduced in Chapter 5.5.1, i.e., Baseline, True-time, and Ideal. Each cyber-system is synthesized under the condition de-

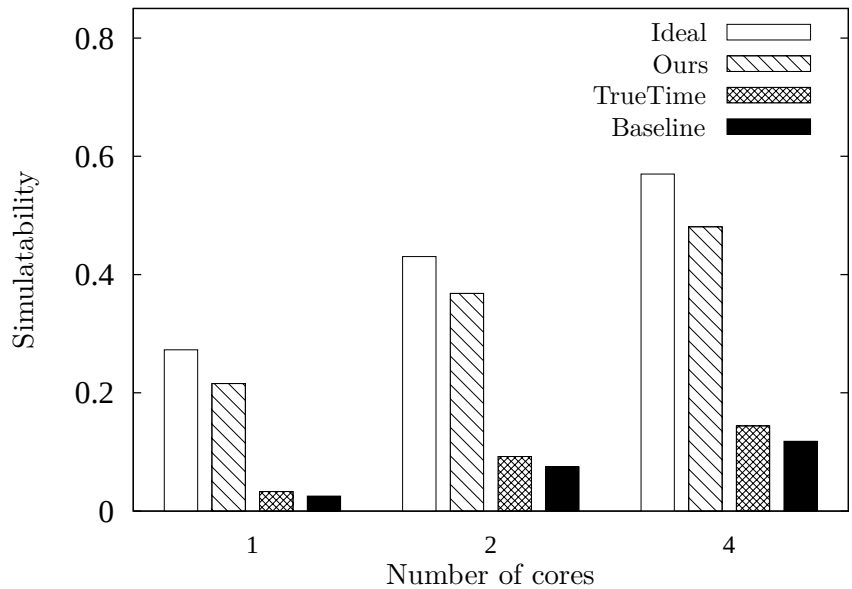


Figure 7.2 Simulatability as changing the number of cores on the simulation PC

scribed in Chapter 5.5.1 except the number of ECUs. The number of ECUs is determined from *uniform*[3,20].

Figure 7.2 compares the simulatabilities of the four approaches as changing the number of cores on the simulation PC. The X-axis represents the number of cores on the simulation PC, and the Y-axis represents simulatability. For each number of cores, we synthesize 1000 target systems with the aforementioned parameters. As the number of cores increases, simulatabilities of the all approaches increases. Compared with **Baseline**, by enjoying the start time freedom, **TrueTime** can only achieve small improvement. However, if we execute jobs applying both freedoms of start times and job execution order, i.e.,

Ours, then the simulatability increases dramatically compared to **Baseline** and **TrueTime**. Also, we can know that **Ours** is not bad compared to **Ideal** which is an unrealistic approach. Note that since EDF is an optimal scheduling algorithm for the singlecore processor, **Ideal** on the singlecore PC is the maximum simulatability we can get. However, **Ideal** on the multicore PC is not the optimal result since there is no optimal scheduler for scheduling jobs in the offline guider on the multicore PC.

Chapter 8
Conclusion

8.1 Summary

Targeting for complex cyber-physical systems, this dissertation proposes a novel approach for functionally and temporally correct simulation of the cyber-side of the cyber-physical system. The approach consists of two steps: (1) constructing the offline guider and (2) online progressive scheduling. 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.

8.2 Future Work

In this section, we identify the future research directions of our propose simulation approach as follows:

- *More study on scheduling algorithms for the multicore simulation PC.* In this dissertation, we only present the online progressive scheduling algorithm based on G-EDF for the multicore simulation PC. In the future,

we plan to study other scheduling algorithms for multicore processors to further improve the real-time simulatability.

- *Implementation for the multicore simulation PC.* In this dissertation, we only implement our simulation approach for the singlecore simulation PC. In the future, we plan to extend our implementation so that we can utilize multicore processors on the simulation PC.

- *Bounded condition for execution time mapping.* We plan to extend mapping model from PC execution times to ECU execution times so that we can allow the bounded condition. By this, we expect to improve the real-time simulatability without loss of accuracy of the simulation result.

- *Real-time simulatability analysis.* We plan to make an analysis for real-time simulatability for the given target system.

References

[1] E. Lee. *Cyber Physical Systems: Design Challenges*. Proc. of IEEE Internaltional Symposium on Object Oriented Real-Time Distributed Computing (ISORC), May 2008.

(Referenced on page 1.)

[2] S. Wu, H. Chiang, J. Perng, T. Lee, and C. Chen. *The automated lane-keeping design for an intelligent vehicle*. Proc. of IEEE Intelligent Vehicles Symposium, June 2005.

(Referenced on page 1.)

[3] MathWorks Inc., *Simulink*. `http://www.mathworks.com`.

(Referenced on pages 1, 11, and 13.)

[4] National Instruments Corporation, *LabVIEW*. `http://www.ni.com`.

(Referenced on pages 1, 11, and 13.)

[5] P. Derler. E. Lee, M. Torngren, and S. Tripakis. *Cyber-Physical System Design Contracts*. Proc. of ACM/IEEE Internaltional Conference on Cyber-Physical Systems, April 2013.

(Referenced on page 10.)

[6] MathWorks Inc., *Embedded Coder*. `http://www.mathworks.com`.

(Referenced on page 11.)

[7] dSPACE GmbH, *TargetLink*. `https://www.dspace.com`.

(Referenced on page 11.)

[8] H. Giese, G. Karsai, E. Lee, B. Rumpe, and B. Schatz. *Model-Based Engineering of Embedded Real-Time Systems.* Spinger, 2007.

(Referenced on page 12.)

[9] T Henzinger, B. Horowitz, and C. Kirsch. *Giotto: a time-triggered language for embedded programming.* Proceedings of the IEEE, 91(1):84–99, January 2003.

(Referenced on page 13.)

[10] Claudius Ptolemaeus, Editor. *System Design, Modeling, and Simulation Using Ptolemy II.* Ptolemy.org, 2014.

(Referenced on page 13.)

[11] 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.

(Referenced on pages 13, 39, and 80.)

[12] dSPACE GmbH, *AutoBox.* `https://www.dspace.com`.

(Referenced on pages 11 and 15.)

[13] 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.

(Referenced on page 16.)

[14] National Instruments Corporation, *NI HIL Simulator Reference System*, `http://www.ni.com`.

(Referenced on page 16.)

[15] 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.

(Referenced on page 14.)

[16] F. Bellard. *QEMU, a Fast and Portable Dynamic Translator*. Proc. of Usenix Annual Technical Conference (ATEC), pp. 41–46, April 2005.

(Referenced on page 14.)

[17] Imperas Software, *Open Virtual Platforms*. `http://www.ovpworld.org`.

(Referenced on page 14.)

[18] E. Witchel, and M. Rosenblum. *Embra: Fast and Flexible Machine Simulation*. Proc. of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 68–79, May 1996.

(Referenced on page 14.)

[19] R. F. Cmelik and D. Keppel. *Shade: A fast instruction-set simulator for execution profiling*. Proc. of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 128–137, May

1994.

(Referenced on page 14.)

[20] V. Zivojnvic, S. Tjiang, and H. Meyr. *Compiled simulation of programmable DSP architectures*. Proc. of IEEE Workshop VLSI Signal Processing, 1995.

(Referenced on page 14.)

[21] A. Gerstlauer. *Host-Compiled Simulation of Multi-Core Platforms*. Proc. of Rapid System Prototyping (RSP), June 2010.

(Referenced on page 15.)

[22] 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.

(Referenced on page 15.)

[23] 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.

(Referenced on page 15.)

[24] A. Gerstlauer, H. Yu, and D. D. Gajski. *RTOS modeling for system level design. Design, Automation and Test in Europe: The Most Influential*

Papers of 10 Years DATE. Springer, 2008.

(Referenced on page 15.)

[25] H. Posadas, J. A. Adamez, E. Villar, F. Blasco, and F. Escuder. *RTOS modeling in SystemC for real-time embedded SW simulation: A POSIX model*. *Design Automation for Embedded Systems*, 10(4), December 2005.

(Referenced on page 15.)

[26] 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.

(Referenced on page 15.)

[27] T. Henzinger and C. Kirsch. *The Embedded Machine: Predictable, Portable Real-Time Code*. ACM Transactions on Programming Languages and Systems, 29(6), October 2007.

(Referenced on page 16.)

[28] M. Matar and R. Iravani. *FPGA Implementation of the Power Electronic Converter Model for Real-Time Simulation of Electromagnetic Transients*. IEEE Transactions on Power Delivery, 25(2):852–860, April 2010.

(Referenced on page 16.)

[29] J. Belanger, P. Venne, and J. Paquin. *The What, Where and Why of Real-Time Simulation*. OPAL-RT TECHNOLOGIES Inc., 2010.

(Referenced on page 16.)

[30] Functional Mock-up Interface, *FMI*. `http://fmi-standard.org`.

(Referenced on page 16.)

[31] IEEE Standards Association, *High Level Architecture*. `https://standards.ieee.org/findstds/standard/1516-2010.html`.

(Referenced on page 16.)

[32] A. Nicolai and A. Paepcke. *Co-Simulation between detailed building energy performance simulation and Modelica HVAC component models*. Proc. of International Modelica Conference, pp. 63–72, May 2017.

(Referenced on page 17.)

[33] O. Chilard, J. Boes, A. Perles, G. Camilleri, M.-P. Gleizes, J.-P. Tavella, and D. Croteau. *The Modelica language and the FMI standard for modeling and simulation of Smart Grids*. Proc. of International Modelica Conference, pp. 189–196, September 2015.

(Referenced on page 17.)

[34] J. Gundermann, M. Thiele, S. Fraulob, S. Walther, K. Todtermuschke, and U. Schnabel. *The Embedded Simulation via FMI and its Application to the Simulation of Lifetime Tests Including Wear*. Proc. of International Modelica Conference, pp. 541–545, May 2017.

(Referenced on page 17.)

[35] Z. Jin and R Gupta. *LazySync: A New Synchronization Scheme for Distributed Simulation of Sensor Networks*. Proc. of International Conference

on Distributed Computing in Sensor Systems (DCOSS), pp. 103–116, June 2009.

(Referenced on page 17.)

[36] G. Lasnier, J. Cardoso, P. Siron, C. Pagetti, and P. Derler. *Distributed Simulation of Heterogeneous and Real-Time Systems*. Proc. of International Symposium on Distributed Simulation and Real Time Applications (DS-RT), pp. 55–62, November 2013.

(Referenced on page 17.)

[37] Z. Jiang. *A Novel Simulation Time and Wall Clock Time Synchronization Algorithm for HLA*. Proc. of International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), pp. 257–260, October 2012.

(Referenced on page 17.)

[38] X. Xu and G. Li. *A Management and Control Infrastructure for Integrated Real-Time Simulation Environment*. Proc. of International Symposium on Distributed Simulation and Real Time Applications (DS-RT), pp. 197–204, November 2013.

(Referenced on page 17.)

[39] Jane W. S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

(Referenced on pages 18 and 19.)

[40] C. Liu and J. Layland. *Scheduling algorithms for multiprogramming in a hard-real-time environment.* Journal of the ACM, 20(1), pp. 46–61, 1973.

(Referenced on pages 18, 19, and 20.)

[41] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A.J. Wellings. *Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling.* Software Engineering Journal, 8(5), pp. 284–292, 1993.

(Referenced on page 20.)

[42] Mechanical Simulation Corporation, *CarSim RT.* `https://www.carsim.com`.

(Referenced on pages 3 and 87.)

[43] Infineon Technologies AG, *TC1797*, `http://www.infineon.com`.

(Referenced on pages 25, 27, and 86.)

[44] NXP Semiconductors, *MPC5566*, `http://www.nxp.com`.

(Referenced on page 25.)

[45] 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.

(Referenced on pages 25, 86, and 95.)

[46] National Instruments Corporation, *FlexRay Automatic Communication Bus Overview.* `http://www.ni.com`.

(Referenced on page 25.)

[47] Intel Corporation, *Intel Core i5-7600 Processor*. https://www.intel.com.

(Referenced on page 27.)

[48] 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.

(Referenced on page 51.)

[49] 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.

(Referenced on pages 35, 41, 64, and 80.)

[50] RUBIS Lab., *CPSim*. https://github.com/rubis-lab/CPSim_linux.

(Referenced on page 100.)

[51] RUBIS Lab., *A Real-Time Simulator for Automotive Systems*. http://rubis.snu.ac.kr/researches#25796.

(Referenced on page 100.)

[52] Vector Informatic Gmbh, *VN8900*. https://vector.com.

(Referenced on page 96.)

[53] H. Joo, K.-S. We, S. Kim, and C.-G. Lee. *An End-to-End Tool for Developing CPSs from Design to Implementation*. Proc. of International Workshop on Verification and Validation of Cyber-Physical Systems (V2CPS),

June 2016.

(Referenced on page 100.)

[54] K.-S. We, J.-C. Kim, and C.-G. Lee. *A Novel Simulation Framework for Supporting Real-Time Cyber-Physical Interactions*. Proc. of International Workshop on Large-Scale Cyber-Physical Systems (LCPS), December 2011.

(Referenced on page 100.)

[55] J.-H. Han, K.-S. We, and C.-G. Lee. *WiP Abstract: Cyber Physical Simulations for Supporting Smooth Development from All-simulated Systems to All-real Systems*. Proc. of International Conference on Cyber-Physical Systems (ICCPS), April 2013.

(Referenced on page 100.)

[56] K.-S. We, J.-H. Han, J.-C. Kim, Y. Oh, C. Oh, and C.-G. Lee. *A Novel Real-Time Simulator for Smooth Development of CPSs from All-Simulated Systems to All-Real Systems*. Proc. of IEEE Real-Time Systems Symposium (RTSS@Work), December 2012.

(Referenced on page 100.)

[57] K.-S. We, J.-C. Kim, Y. Oh, S.-M. Jeong, and C.-G. Lee. *Demo Abstract: An Efficient and Easilly Reconfigurable Cyber-Physical Simulator*. Proc. of International Conference on Cyber-Physical Systems (ICCPS), April 2013.

(Referenced on page 100.)

[58] A. Melani, M. Bertogna, V. Bonifaci, A. M.-Spaccamela, and G. Buttazzo. *Response-Time Analysis of Conditional DAG Tasks in Multiprocessor Systems.* Proc. of Euromicro Conference on Real-Time Systems (ECRTS), pp. 211–221, July 2015.

(Referenced on page 18.)

[59] A. Melani, M. Bertogna, V. Bonifaci, A. M.-Spaccamela, and G. Buttazzo. *Schedulability Analysis of Conditional Parallel Task Graphs in Multicore Systems.* IEEE Transactions on Computers, 66(2):339–353, February 2017.

(Referenced on page 18.)

[60] S. Baruah, V. Bonifaci, and A. M.-Spaccamela. *The Global EDF Scheduling of Systems of Conditional Sporadic DAG Tasks.* Proc. of Euromicro Conference on Real-Time Systems (ECRTS), pp. 222–231, July 2015.

(Referenced on page 18.)

[61] J. Schutte. *Implementing Parallel Topological Sort in a Java Graph Library.* Proc. of Twente Student Conference on IT, June 2014.

(Referenced on page 18.)

[62] A. Somani and D. Singh. *Parallel Genetic Algorithm for solving Job-Shop Scheduling Problem Using Topological Sort.* Proc. of International Conference on Advances in Engineering & Technology Research (ICAETR), August 2014.

(Referenced on page 18.)

[63] G. Malewicz. *Parallel Scheduling of Complex Dags under Uncertainty.* Proc. of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 66–75, July 2005.

(Referenced on page 18.)

[64] T. Yang and A. Gerasoulis. *DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors.* IEEE Transactions on Parallel and Distributed Systems, 5(9):951–967, September 1994.

(Referenced on page 18.)

[65] R. Correa, A. Ferreira, and P. Rebreyend. *Scheduling Multiprocessor Tasks with Genetic Algorithms.* IEEE Transactions on Parallel and Distributed Systems, 10(8):825–837, August 1999.

(Referenced on page 18.)

[66] R. Tomasulo. *An Efficient Algorithm for Exploiting Multiple Arithmetic Units.* IBM Journal of Research and Development, 11(1):25–33, January 1967.

(Referenced on page 18.)

[67] J. Lee, K.-M. Phan, X. Gu, J. Lee, A. Easwaran, I. Shin, and I. Lee. *MC-Fluid: Fluid Model-based Mixed-Criticality Scheduling on Multiprocessors.* Proc. of Real-Time Systems Symposium (RTSS), December 2014.

(Referenced on page 18.)

[68] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt. *DP-FAIR: A Simple Model for Understanding Optimal Multiprocessor Scheduling*. Proc. of Euromicro Conference on Real-Time Systems (ECRTS), pp. 3–13, July 2010.

(Referenced on pages 102 and 103.)

[69] J. Li, Z. Luo, D. Ferry, K. Agrawal, C. Gill, and C. Lu. *Global EDF scheduling for parallel real-time tasks*. Real-Time Systems Journal, 51(4):395–439, July 2015.

(Referenced on page 103.)

요약(국문초록)

사이버-물리 시스템을 개발할 때, 디자인 단계에서 시스템의 최종 성능을 예측하기 위하여 시뮬레이터가 널리 이용된다. 그러나 현재의 시뮬레이터들은 태스크의 가변 수행 시간, 자원 선점 등 사이버 시스템에서 발생하는 시간 행태를 고려하지 않기때문에, 시뮬레이터에서 예측된 제어 성능이 실제 성능과 큰 차이를 보인다. 이로 인해 시스템을 재디자인하고 재구현하는 과정을 반복해야 하므로 시스템을 개발하는데 많은 시간과 비용이 소모된다. 이 한계를 극복하기 위해, 본 논문에서는 사이버-물리 시스템의 사이버 시스템에 대한 기능적/시간적 정확성 보장 시뮬레이션 기법을 제안한다. 제안하는 방법에서는 데이터와 시간 정확성을 물리적 상호작용 시점에서만 보장하고 그 외에는 시뮬레이션 작업에 대한 스케줄링 자유도를 최대한 활용함으로써 이를 이룩한다. 이를 위해 본 논문에서는 시뮬레이션 문제를 선후관계 제약조건을 가진 실시간 작업 스케줄링 문제로 변환하고, 이 스케줄링 문제를 효율적으로 해결하는 알고리즘을 제안한다. 제안된 방법은 기능적/시간적 정확성을 보장하면서도 최신의 시뮬레이션 방법들보다 실시간 시뮬레이션 용량을 크게 증가시킨다. 우리는 인위적인 워크로드를 이용한 실험과 실제 구현을 통해서 제안된 방법이 다른 최신의 시뮬레이션 방법들 대비 높은 정확성과 동시에 높은 효율을 가진다는 것을 보인다.

주요어 : 사이버-물리 시스템, 실시간 시뮬레이션, 시간적 정확성, 기능적 정확성, 효율적 시뮬레이션

학 번 : 2011-30966