공학박사학위논문

자동차 제어 시스템을 위한 컴포넌트 기반의 스케쥴링 및 시스템 최적화

Component-based Scheduling and System Optimization for Automotive Control Systems

2013년 8월

서울대학교 대학원
전기 · 컴퓨터공학부
김종찬

초 록

복잡한 자동차 제어 소프트웨어를 개발하기 위해 전체 시스템을 단순하고 쉽게 검증 가능한 소프트웨어 컴포넌트를 조립하여 개발하는 컴포넌트 기반의 개발 방법이 많이 사용되고 있다. 따라서, 단위 소프트웨어 컴포넌트의 기능을 검증함으로써 전체 시스템의 기능적 완결성을 쉽게 검증할 수 있다. 하지만, 소프트웨어 컴포넌트의 입력에서 출력까지의 지연 시간은 하드웨어의 성능과 하드웨어 자원을 공유하는 소프트웨어 컴포넌트들에 따라 일관되게 유지되지 못한다. 이와 같은 문제 때문에 컴포넌트 기반으로 시스템을 설계하더라도 전체 시스템의 시간 정확성을 검증하기는 매우 어렵다. 이 문제를 해결하기 위해서 본 논문은 각 소프트웨어 컴포넌트의 지연 시간을 일정하게 보장하면서 주어진 제어 트랜잭션들을 네트워크로 연결된 ECU 위에 구현하는 새로운 프레임워크를 제안한다. 제안하는 프레임워크를 사용하면 각 소프트웨어 컴포넌트의 지연 시간을 정확히 보장함으로써 전체 시스템의 시간 정확성을 단위 컴포넌트의 지연 시간으로부터 쉽게 검증할 수 있다. 또한 컴포넌트 스케쥴링을 위하여 기존에 사용되는 자원 배분 방법은 각 소프트웨어 컴포넌트에 지속적으로 주어진 하드웨어 용량을 할당하기 때문에 자동차 제어 트랜잭션처럼 소프트웨어 콤포넌트들이 순차적으로 수행되는 경우 심각한 하드웨어 자원 낭비를 초래하게 된다. 이 문제에 대한 해결 방법으로 본 논문은 하드웨어 자원 용량을 각 소프트웨어 컴포넌트가 실제로 필요로 하는 시간 동안만 제공하는 새로운 자원 공유 방법을 제안한다. 실험 결과에 의하면 본 논문의 결과는 주어진 하드웨어 자원을 효율적으로 활용하여 기존

의 자원 배분 방법보다 약 세 배 많은 수의 제어 트랜잭션을 수용할 수 있다. 이와 같은 자원 활용 방법을 기반으로 본 논문은 또한 주어진 제어 트랜잭션을 구현할 수 있는 최소 비용의 하드웨어 시스템 설정을 찾는 방법을 제시한다. 제안하는 프레임워크의 성능은 시뮬레이션과 구현 연구에 의해서 검증된다. 특히 제안하는 프레임워크를 툴체인과 컴포넌트 수행 커널로 구현함으로써 제안하는 프레임워크의 유용성을 검증한다.

주요어 : 자동차 제어 소프트웨어, 스케쥴링, 시스템 최적화
학번 : 2008-30219

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Along with the increasing needs for advanced features of safety and comfortability, the automotive software system becomes more important and also more complex. Modern automotive control system is composed of a large number of networked ECUs (Electronic Control Units) that basically accepts sensing data from sensors, calculates the actuation commands, and sends the actuation commands to actuators as in Figure 1. In a high-end vehicle, there are over 70 ECUs connected by more than 5 different networks [2]. When developing such a complex software system, component-based development method is commonly used, under which the entire complex system is developed by composing less complex SW components. Thus, the entire system's functional correctness can be easily validated by verifying each SW component's functional behavior.

However, if we shift our focus from the functional correctness to the non-functional correctness of the system, things are totally different. Particularly, under the commonly used SW component scheduling principles such as RM (Rate Monotonic) and EDF (Earliest Deadline First) scheduling algorithms, each SW component's temporal behavior (i.e., input-output delay) is significantly affected by the underlying HWs and surrounding SW components running concurrently on the same HW. Thus, a slight environ-

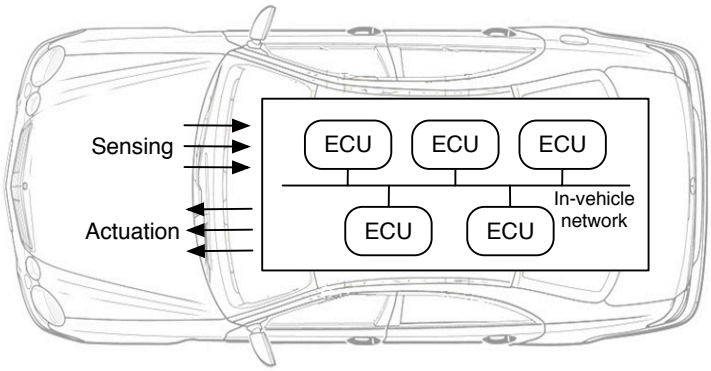


Figure. 1: Automotive control system

mental change can make a SW component's temporal behavior totally different from the original design intention. Moreover, the entire system's temporal behavior (i.e., end-to-end delay from the sensors to the actuators) is not defined as a simple composition of constructing SW components' temporal behavior. Rather, complex analysis methods [3, 4] are needed when deriving the end-to-end temporal behavior from component-level temporal properties.

With the above motivation, this dissertation proposes a novel framework where each SW component has its guaranteed invariant input-output delay and the entire system's end-to-end response time becomes a simple composition of those invariant delays. This per-component invariant delay property makes each SW component a complete building block not only in the functional domain but also in the temporal domain. Thus, the entire system can be developed by composing each SW component's function F

and also invariant delay d. As a result, the entire system's both functional and temporal correctness can be easily validated from the SW components' (F, d)s.

Under our framework, control engineers first design end-to-end control transactions from sensing to actuation by composing (F, d)s of SW components. At that time, they do not care about the underlying scheduling and the system configuration (i.e., a set of ECUs, a set of CANs, and SW/HW mapping) upon which the control transactions will be actually implemented. Rather, they focus on developing their control algorithms assuming each component's delay property. Then, using (F, d)s as a design contract [5] between control engineers and software engineers, software engineers can systematically find the optimal scheduling and system configuration that guarantee each SW component's functional correctness F and invariant delay property d using our framework. This clear role separation between control engineers and software engineers is also beneficial since each party can focus on their own responsibilities without being burdened with the other party's concerns. In order to develop such a framework, we specifically deal with the following two problems:

- *Scheduling Problem.* Find SW component scheduling ensuring component's delay property for given set of control transactions and given system configuration.

- *System Optimization Problem.* In case there is no given system configuration, find the minimal cost system configuration with a feasible SW component scheduling for given set of control transactions.

In the remainder of this chapter, we summarize our motivation and challenge, state our objective, explain the high-level description of our approach, highlight the research contributions, and outline the rest of this dissertation.

1.1 Motivation and Objective

When designing complex automotive control software systems, they are designed by composing SW components — "component-based design". However, after they are actually implemented on networked ECUs, each component's temporal behavior is not guaranteed as constant due to the underlying scheduling with concurrently running SW components. As a result, the resulting system's temporal behavior is very difficult to analyze. Thus, there needs a new framework that guarantees each component's design-time invariant delay property and simplifies each control transaction's end-to-end temporal analysis as a simple composition of its constructing SW components' delays. In such a way, control engineers can design their control algorithm assuming each component's invariant delay, and software engineers can implement it by ensuring the component's invariant delay property using our framework.

Following the above motivation, in this dissertation, we develop a framework that realizes given control transactions on networked ECUs ensuring component's invariant delay property.

1.2 Approach

In order to fundamentally achieve our objective, we first find SW component scheduling that ensures component's delay property. As a baseline scheduling principle, we employ existing bandwidth reservation mechanisms. However, since the existing methods simply assign a permanent utilization to each SW component, they waste away the assigned HW capacity during which certain SW components are not running. In a control transaction where SW components execute sequentially, this resource waste is significant and hence not tolerable. Thus, our specific scheduling problem is to find SW component scheduling not only ensuring component's delay property but also eliminating resource waste for given set of control transactions and given system configuration.

In order to eliminate the resource waste, which is inherent to traditional bandwidth reservation mechanisms, our approach is to provide the required resource utilization only during each SW component's *active windows* which are the time durations the SW component actually needs the resource utilization. Each component's active windows are first determined by considering intra-transaction component dependency then optimized by exploiting inter-transaction phase dependency. This active windows information is then transferred to our component execution kernel, which actually schedules SW components by creating servers according to the active windows information in a time-triggered manner.

For given control transactions with a freedom to choose a system configuration, our second problem is to find the minimal cost system configu-

ration with a feasible SW component scheduling. This is important because we can reduce the manufacturing cost by finding the optimal system configuration. However, this system optimization problem is known to be NP-hard, thus we have to employ a heuristic algorithm that searches the solution space.

As a solution to the system optimization problem, we propose a *heuristic iterative search* algorithm. Starting from an initial solution where each SW component is dedicated to its own ECU, i.e., one-to-one mapping, our algorithm iteratively searches the near-optimal solution by merging a pair of HW resources until the optimal system configuration with the minimal system cost is found.

By solving above two problems, i.e., scheduling problem and system optimization problem, we develop a framework that implements given control transactions in a resource efficient way guaranteeing each component's invariant delay property.

1.3 Contributions

The contributions of this dissertation can be summarized as follows:

- We propose a new framework that ensures each SW component's invariant delay property thus making a SW component an invariant building block not only in the functional domain but also in the temporal domain.

- As a specific problem, we find SW component scheduling that ensures SW component's invariant delay property without wasting resources

for given control transactions and given system configuration.

- Under the above SW component scheduling mechanism, we also develop a system optimization method that finds a near-optimal system configuration, that is, a set of ECUs, a set of CANs, and SW/HW mapping, with the minimal system cost.

- Based on the above approaches, we implement an end-to-end toolchain and component execution kernel to demonstrate the usability of our framework.

1.4 Organization

The rest of this dissertation is organized as follows:

- Chapter 2 surveys the related work, which includes the automotive control systems, component-based design and development, frameworks with invariant delay property, real-time scheduling, and system optimization.

- Chapter 3 presents the system model, assumptions, and terms and notations for this dissertation.

- Chapter 4 presents our SW component scheduling algorithm that most efficiently utilizes underlying HW resources by exploiting intra and inter-transaction component dependencies while guaranteeing each SW component's invariant delay property.

- Chapter 5 presents our system optimization algorithm, which systematically finds the system configuration with the minimum system cost from given set of control transactions.
- Chapter 6 briefly demonstrates our end-to-end toolchain and component execution kernel.
- Chapter 7 concludes this dissertation and discusses the future work.

Chapter 2

Related Work

2.1 Automotive Control Systems

Automotive control systems are rapidly transforming into software-centric electrical systems. Automakers are putting an enormous amount of investment to developing advanced automotive control software. According to this trend, consumers are seeking advanced features like LKAS (Lane Keeping Assist System) and ACC (Adaptive Cruise Control) when buying even mid-range cars. Governments are also accelerating the trend by safety and environmental regulations. For example, the European Commission enforces the mandatory installation of ESC (Electronic Stability Control) on all new cars to be sold in the European Union (EU) from 2014.

Automotive control system is composed of ECUs, which are small embedded computers, connected by in-vehicle networks. In each ECU, control algorithms are implemented as control software. Nowadays almost every mechanical component in a vehicle is controlled by software. More specifically, in the power-train domain, EMS (Engine Management System) controls the flow of fuel into the cylinder according to the throttle position and the timing of ignition for each cylinder according to the crankshaft position. Automatic transmission is also controlled by TMS (Transmission Management System) which controls the timing to change gears in the vehicle to

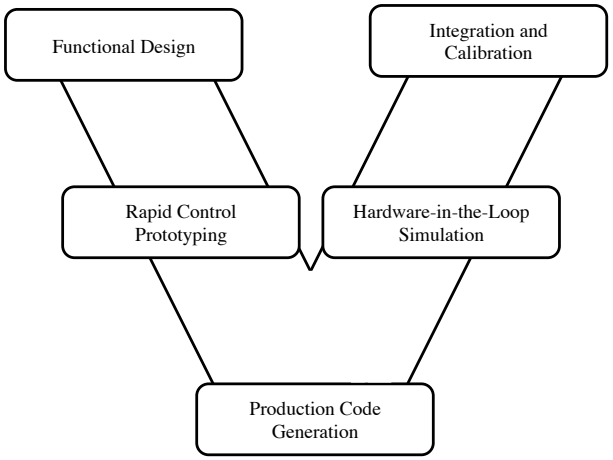


Figure. 2: V-cycle process

optimize performance and fuel efficiency. In the chassis and safety control domain, various controllers such as ESC, EPS (Electric Power Steering), and SCM (Suspension Control Module) control brakes, steering, and suspensions in order to make the vehicle move closely following the driver's intention and also ensuring safety. Recently, most advanced safety features like LKAS and ACC (Adaptive Cruise Control) are related to chassis control systems. In the body control and comfort control domain, even wipers, seats, and door locks are controlled by software in BCM (Body Control Module).

A new vehicle with such complex software system is currently being developed following the V-cycle process as in Figure 2. The V-cycle basically consists of (1) functional design, (2) rapid control prototyping, (3) production code generation, (4) Hardware-in-the-Loop Simulation (HiLS), and (5) integration and calibration. In the functional design phase, automaker's control engineers design their control algorithms using various model-based

design tools such as Simulink [6], Ascet [7], and LabView [8]. The functional correctness of the control algorithms are validated through PC-based simulation environments. In order to validate the function of the control algorithms in a real-time environment, in the rapid control prototyping phase, the control algorithms are automatically downloaded to rapid prototyping hardwares such as AutoBox [9], which are actually connected to mechanical components to measure the control performance of the designed algorithms. Then, in the production code generation phase, source codes for the control algorithms are generated. When generating the source codes, automatic code generation tools are commonly used to prevent errors that can be injected during hand-coding. In the HiLS phase, source codes are compiled and deployed to proper ECUs and each ECU is tested with its target mechanical components. In the final stage, i.e., integration and calibration phase, ECUs are integrated through in-vehicle networks and control parameters are calibrated to make a final production system.

When integrating various features onto a vehicle, the traditional approach is running only a single function on each ECU, thus called *single-function ECUs*. With this traditional method, however, the number of ECUs needed for a vehicle drastically increases as new advanced features are added. This is a new challenge to the automakers, since they want to continue adding new advanced features but at the same time they also want to reduce the number of ECUs for reducing manufacturing cost, simplifying the wire harness, and making more space for passengers [10, 11]. Thanks to the recent advance in the computing power of automotive microprocessors, the automakers are now seeking a new way of reducing the required num-

ber of ECUs in a vehicle while adding an ever-increasing number of new advanced features. However, in order to reduce the number of ECUs, it is inevitable to integrated multiple independently-developed SW components in a single ECU, i.e., *multi-function ECU*. This SW component integration technology is relatively new in the automotive control software domain. The most serious hurdle toward the multi-function ECUs is that automotive control software is not developed with portability in mind at all. Usually, they are developed tightly coupled with the underlying ECU HWs and firmware, thus not portable to other HW platforms.

To overcome this limitation, major automotive industry leaders are collaborating under the flag of AUTOSAR [12]. The AUTOSAR standard clearly separates the application layer and the platform layer and defines the standard SW component interface to make them portable across different HW platforms. However, its underlying OS OSEK [13] is based on the fixed-priority scheduling, which causes significant temporal deviation to SW components from the original design time intention. This temporal deviation requires a painful revalidation step at the system integration phase, which is a important motivation of this dissertation.

Another driving force to the multi-function ECUs era is multicore ECUs. The traditional approach of scaling up the clock speed of a CPU (Central Processing Unit) to increase the computing power reaches its limit due to the side effects such as excessive power consumption and heat problem [14, 15]. Thus, instead of scaling up the clock speed, increasing the number of cores is considered a proper solution to get more computing power. Following the trend, automotive CPU vendors are about to mass-produce multicore

CPUs [16, 17, 18]. However, in the industry, multicore ECUs are not commonly used at least for now. The problem lies in the software, not in the hardware issues. To use the computing power of multicore CPUs efficiently, the entire software stack should be redesigned from the compiler and OS to the run-time environment. Particularly, the multicore CPU scheduling is still an open research issue. Finding the optimal scheduling policy for multicore CPU is not easy as the singlecore CPU case where the fixed-priority scheduling with RM (Rate Monotonic) priority assignment is dominant for its simplicity and optimality.

There are over 70 ECUs across a car connected by various in-vehicle networks [2]. Since in-vehicle networks require timely delivery of critical sensor data and actuation commands, general-purpose networking solutions do not fit. Instead, specially designed networking standards are used in a vehicle. The most widely used in-vehicle network is the CAN (Controller Area Network) bus, which was originally designed by Robert Bosch GmbH [19] in the 1980s. It supports up to 1 Mbps bandwidth which is subject to the cable length. Its robustness to errors and arbitration-free transmission mechanism make it suitable for in-vehicle networking. CAN supports fixed-priority non-preemptive scheduling where the priority is given by the CAN message id. Due to its wide acceptance in the industry, there have been numerous researches about CAN in the literature. Tindell et al. [20] first introduced the worst-case response time analysis method for CAN messages, which was later revised by Davis et al. [21]. Andersson and Tovar [22] proved that the utilization bound of original CAN is 25%. Another stream of researches are about using CAN under the EDF schedul-

ing algorithm [23, 24, 25], which was mainly motivated by the inherent low utilization of the fixed-priority scheduling algorithm. To improve the time predictability of CAN, TTCAN [26], a time-triggered extension of CAN, was introduced, which was also extended by FTT-CAN [27, 28] that supports both time-triggered and event-triggered messages. Based on FTT-CAN, Nolte et al. [29] introduced a server-based scheduling algorithm for the CAN bus that supports bandwidth isolation.

Recently, flexray is also gaining popularity especially for the x-by-wire applications with its time predictable architecture and high bandwidth (10 Mbps). LIN (Local Interconnect Network) is a master-slave serial communication network with a lower bandwidth compared to CAN bus. MOST (Media Oriented Systems Transport) is a special purpose automotive network technology for transferring high bandwidth multimedia data.

2.2 Component-based Design and Development

When designing a complex software system, the entire system is designed by composing less complex SW components, which is a logical unit of computation and deployment. Thus, through simply verifying each SW component's functional behavior, the entire system's functional correctness can be easily verified. In the embedded systems industry, model-based design tools such as Simulink [6], LabView [8], and Ascet [7] are commonly used when designing complex control systems like automotive and avionics control systems. Using these model-based design tools, control engineers can design complex control algorithms by drawing block diagrams and ver-

ify their functional correctness in simulation environments.

As a theoretical foundation, most model-based design tools are based on the actor model [30]. In the actor model, the unit of computation is an actor. An actor has a number of input ports and output ports. Messages are exchanged through input/output ports (not necessarily) asynchronously. When an actor receives inputs through its input ports, it calculates the values to be sent through the output ports. Output ports are also connected to other actors' input ports. It is strictly required that messages are the only way to share information between actors. Thus, we can assume that there is no shared resource between actors. Using the actor model, the resulting system can be described as a directed acyclic graph (DAG) where each vertex is an actor and each edge is a message.

After designing the system as a model, the resulting model can be translated into an equivalent Lustre [31], Esterel [32], and Z formal language [33] for formal verification [34]. Also, the functionality of the designed control algorithms are verified through synchronous reactive simulations [35]. Note that the time in the synchronous reactive simulation environment does not reflect the real-time clock since it is a virtual time and the simulation does not consider the time delay which will be incurred in the execution platform. Instead, the simulation environment assumes that the time delay from the inputs to the outputs of each block is always zero. Thus, even after the system is verified in the simulation environment, its temporal correctness should be verified in another way such as worst-case response time analysis [36, 37] and real-time simulation [38]. After the model's functional correctness is fully verified by model checking and simulation, automatic code generation

tools [39, 40] generate source codes which can be used to implement the production system. Since the automatically generated codes are free of syntactic and semantic errors, they can enhance the trustworthy of the resulting production system.

Most model-based design tools support transformation of models to AUTOSAR SW components. AUTOSAR SW component is composed of a number of runnables. Each runnable is mapped to a task in the OSEK kernel. The mapping between runnables and tasks becomes an important architectural decision since a poor mapping can greatly impact the schedulability and resource efficiency of the resulting system. Navet et al. [10] proposed a runnable dispatching algorithm that efficiently utilizes the underlying multicore CPUs by slot-based bin packing. Wang and Shin [41] introduced a method for task construction for model-based design of embedded control software considering end-to-end timing constraints. Although above mapping mechanisms can make a correct mapping from runnables to tasks following their own optimization objective, they are missing an important condition for providing safety. When multiple independently developed AUTOSAR SW components are mapped to the same ECU, it is possible that runnables from different AUTOSAR SW components are mapped to the same OSEK task. In such case, OSEK's task-level execution time monitoring and protection mechanisms do not work making an AUTOSAR SW component's timing fault propagate to different AUTOSAR SW components. Thus, there needs a new scheduling method that directly implements SW components as scheduling units.

2.3 Invariant Delay Property

In the literature, there have been several researches about providing invariant delay property to SW components or tasks to achieve deterministic system behavior. In that sense, Giotto [42] and Timed multitasking [43] share the similar motivation with this dissertation.

Giotto is a time-triggered architecture composed of a time-triggered programming language, a compiler, and a runtime system. Giotto specifically targets hard real-time systems with periodic workload. The key concept of Giotto is separating functionality and timing when designing a system. The control system designer makes a functionality program in a traditional programming language such as C and a timing program in Giotto language. Then, Giotto compiler automatically generates time-related parts of the final program, which is linked with the functional parts to yield a final program. This final program runs under various scheduling algorithms such as RM and EDF. Although tasks are triggered by time, according to the underlying scheduling discipline, the completion time can vary. This indeterminism can hurt the time predictability of the system severely. To overcome this limitation, Giotto proposed an additional concept LET (Logical Execution Time), which is a constant time delay from a task's inputs to outputs. In LET, when a task is starting in a time-triggered manner, it atomically reads every inputs it requires. Then, it calculates the outputs under various scheduling algorithms. After completing the calculation, the task waits for the exact time reserved for yielding outputs. At the exact output time, the outputs are given to other tasks or actuators atomically. Although

Giotto pioneered the LET concept, it is not well suited with our transaction model since in Giotto, every SW component are periodically invoked with their deadlines equal to periods. Thus, it causes inherent oversampling, that is, even there is no fresh inputs, the SW components executes in vain. To the contrary, our model assumes that SW components in a transaction are triggered sequentially.

Timed multitasking also employs the LET concept. However, it is different from the Giotto in that Timed multitasking is an event-triggered system whereas Giotto is a pure time-triggered system. In that sense, Timed multitasking is more close to our assumed transaction model where SW components execute sequentially. Since it is an event-triggered system, it is easier to handle aperiodic workload than Giotto. However, in the scheduling perspective, it is not based on a solid scheduling algorithm. Rather, it guarantees the LET constraints of SW components only when the underlying scheduling algorithm can handle the given event-driven workloads. Thus, there is no formal way to analyze the system feasibility.

2.4 Real-Time Scheduling

2.4.1 Basic Scheduling Theory

A real-time system is a system in which correctness is defined by not only the functional correctness but also the temporal correctness. All control systems are real-time systems since the end-to-end delay from the sensors to the actuators in a control system should be bounded within a certain delay bound requirement. More specifically, control systems are defined as *hard*

real-time systems since its timing constraint is hard meaning that a single violation of the timing constraint may cause a catastrophic result such as human death. To the contrary, in a *soft real-time system*, missing of its timing constraint only degrades the quality of service rather than causing serious results. VOD (Video On Demand) system is a typical example of soft real-time systems. This dissertation specifically deals with control systems, thus hard real-time systems are of more interest than soft real-time systems.

There have been numerous researches about real-time systems mostly since the Liu and Layland's seminal paper in 1973 [44]. The research interests can be roughly categorized into the following three groups: 1) workload models, 2) resource models, and 3) scheduling algorithms. Workload model defines what the system should handle. Resource model defines with what the system can handle the workload. Scheduling algorithm gives how to handle the workload with the given resources.

The most significant workload model is the classical periodic task model which is originally introduced by Liu and Layland [44] where jobs of each task are released periodically with its period. The worst-case execution time of each task is known a priori. Each task has a relative deadline, which can be equal to the period, i.e., implicit deadline, or even shorter or longer than the period. When jobs are released not strictly periodically, but with a minimum inter-release time, they are called sporadic jobs. Usually, the minimum inter-release time is considered as a deadline. Also, when jobs arrive arbitrarily with no priori information, they are called aperiodic jobs. For aperiodic jobs, no deadline is defined usually. Instead, it is required to minimize the service time of a aperiodic job, that is, the time took from the arrival of

the aperiodic job to the completion of it. Various scheduling algorithms for these different workload models have been developed with various research objectives.

Scheduling algorithms can be categorized into offline scheduling and online scheduling by when the scheduling decision is made. The offline scheduling, also known as the cyclic executive scheduling, makes scheduling decisions before the system begins its execution. It is highly predictable but inflexible meaning that once the system begins, nothing can be changed without stopping it. However, the offline scheduling is still useful for very critical systems like avionics systems. In the online scheduling algorithms, there is a simple (weighted) round-robin scheduling, which is well-suited for scheduling network packets. However, when scheduling CPU resources, priority driven approaches are more commonly used, where the scheduler always picks the highest priority job among the runnable ones. Priority driven scheduling algorithms can be also categorized into fixed-priority scheduling algorithms and dynamic-priority algorithms. In the fixed-priority scheduling algorithms, the priority of each task is fixed before the system begins. RM scheduling and DM (Deadline Monotonic) scheduling are known to be optimal in the fixed-priority scheduling domain for a periodic task set with implicit deadlines and arbitrary deadlines, respectively. In the dynamic-priority scheduling, the priority can be changed in the runtime, where EDF and LST (Least Slack Time) scheduling algorithms are known to be optimal for scheduling independent, preemptive jobs with arbitrary release times and deadlines on a single processor.

In order to check the schedulability of a task set under a certain schedul-

ing algorithm, various schedulability analysis methods for fixed-priority scheduling and EDF have been devised. For EDF, Liu and Layland [44] showed that a set of periodic tasks with implicit deadlines $\{T_1, T_2, \cdots, T_n\}$ are schedulable under EDF, if and only if the system utilization U is less than or equal to 100%. That is

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1, \tag{2.1}$$

where C_i is the worst-case execution time of T_i, and P_i is T_i's period. In case that each task's deadline D_i is not equal to its period P_i, we can use the time demand analysis introduced by Baruah et al. [45]. A set of periodic tasks with arbitrary deadlines $\{T_1, T_2, \cdots, T_n\}$ is schedulable under EDF, if and only if $U < 1$ and

$$\forall L > 0, \sum_{i=1}^{n} \left\lfloor \frac{L + P_i - D_i}{P_i} \right\rfloor \cdot C_i \leq L. \tag{2.2}$$

For RM scheduling, Liu and Layland [44] showed that a set of period tasks $\{T_1, T_2, \cdots, T_n\}$ is schedulable under RM, if

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{\frac{1}{n}} - 1). \tag{2.3}$$

As n approaches to the infinity, $n(2^{\frac{1}{n}} - 1)$ approaches to approximately 69.31%, which can be a safe utilization bound for arbitrary periodic task sets. Note that there can be a set of tasks with its system utilization over $n(2^{\frac{1}{n}} - 1)$ but schedulable under RM since Eq. 2.3 is only a sufficient condition. For the exact schedulability check under RM, Audsley et al. [3] showed that the following recursive equation can compute the worst-case response

time R_i of task T_i under RM:

$$R_i^{k+1} = C_i + \sum_{T_m \in HP(i)} \left\lceil \frac{R_i^k}{P_m} \right\rceil \cdot C_m, \tag{2.4}$$

where $HP(i)$ is a set of tasks with a higher priority than T_i and $R_i^0 = C_i$. The recursive equation continues until $R_i^k = R_i^{k+1}$.

So far, we have mainly discussed the scheduling algorithms and schedulability check methods for systems with only periodic tasks. However, a real-time system is usually composed of both time-triggered periodic tasks with hard deadlines and event-triggered aperiodic tasks with best-effort requirements. Thus, in order to schedule such a system with mixed periodic tasks and aperiodic tasks, we have the following two objectives:

- Execute aperiodic jobs without hurting the schedulability guarantees of periodic tasks.
- Minimize the service time of a aperiodic job as much as possible.

Note that the aperiodic job scheduling should try to protect periodic tasks from irregular arrival of aperiodic jobs first. It need not provide guaranteed services to aperiodic jobs. Instead, it should minimize the service time of it on the condition that it does not hurt any periodic task's schedulability. In order to separate aperiodic jobs from periodic tasks, servers are used to handle aperiodic jobs. A server is a scheduling entity with the following two parameters:

- Server execution budget C_s
- Server period P_s

An aperiodic job is serviced on top of a pre-defined server. Various server mechanisms have been proposed with different policies about when to execute aperiodic jobs and when to replenish server execution budget.

For the fixed-priority scheduling, the simplest one is using Background Service [46], which simply queues aperiodic jobs and services them in the first come and first served manner whenever there exists no periodic job to run. Although it is simple to implement, it does not provide any service guarantee to aperiodic jobs meaning that no aperiodic job is serviced at all in the worst case. Polling Server [46] is a pseudo periodic task, which is scheduled just like a periodic task with period P_s with execution budget C_s. Deferrable Server [46] improves Polling Server by servicing aperiodic jobs earlier when there is unused capacity without waiting for the start of next server period, thus improving the responsiveness of aperiodic jobs compared to Polling Server. Priority Exchange Server [47] provides a better schedulability bound for periodic tasks compared to Deferrable Server, sacrificing the responsiveness of aperiodic jobs. Sporadic Server [48] spreads the budget replenish at least P_s time units enabling Sporadic Server to be treated just as a periodic server with period P_s and execution time C_s in the schedulability check. Slack Stealing [49] does not create a server. Instead, Slack Stealing offers substantial improvements to the aperiodic job's response time by actively stealing all the processing time from the periodic tasks making them barely meet their deadlines.

For the dynamic-priority scheduling, Spuri and Buttazzo [50] proposed five different algorithms for handling aperiodic tasks. Dynamic Priority Exchange Server and Dynamic Sporadic Server are introduced, which are ex-

tended from their original fixed-priority versions. Then, they also proposed a new algorithm Total Bandwidth Server, which enhances the performance of the previous two servers. Contrary to the other algorithms, Total Bandwidth Server needs to know the worst-case execution times of aperiodic jobs at the time they arrive. Using this information, Total Bandwidth Server sets its deadline preserving the total utilization consumed serving aperiodic jobs under a certain bandwidth U_s. Then, the remaining bandwidth $1-U_s$ is used for handling periodic tasks. They also introduced an optimal algorithm Deadline Late Server which is similar to the Stack Stealing in the fixed-priority scheduling. Since this optimal algorithm incurs serious runtime overhead, it is not practical to be deployed in a real system. An approximated algorithm Improved Dynamic Priority Exchange Server was also introduced. Later, Abeni and Buttazzo [51] proposed Constant Bandwidth Server, which requires no a priori knowledge about the worst-case execution times of aperiodic jobs. A Constant Bandwidth Server is simply characterized by server period P_s and server execution budget C_s. Then, it guarantees a constant bandwidth $\frac{C_s}{P_s}$ to aperiodic jobs.

2.4.2 Distributed Real-Time Scheduling

In a distributed system, there are multiple nodes connected by networks. In such systems, a transaction begins at a certain node. Then, after passing through several nodes, it finally finishes at also a certain node. Thus, the timing requirement is given as an end-to-end response time from the starting node to the ending node for each transaction.

For the distributed real-time scheduling, Tindell et al. [4] proposed the

holistic approach to analyze the end-to-end response time of distributed transactions. They assumed the periodic transaction model, under which when a task in a certain node finishes, it sends its output message to another node with the next task in the transaction. Then, upon receiving the message, the receiving task is released by the receiving event. In each node, priority-driven scheduling is employed and the priority is given transaction-wide. Although this holistic approach can analyze the end-to-end response time of distributed transactions, it does not guarantee intermediate deadlines for each task in a transaction. Rather, only the end-to-end deadlines are considered.

In the automotive industry, most widely accepted distributed execution model is the *periodic activation with asynchronous communication model* [52, 53, 54]. In this model, in each node, tasks are periodically executed. When a job of a task starts, it reads inputs from input buffer no matter whether it is fresh or not. After finishing the execution, it sends out the output to the receiving task on another node in a asynchronous way. In the same way, the receiving task starts according to its own period after reading inputs from input buffer. In each node, priorities are given under RM priority assignment policy and fixed-priority scheduler executes tasks. The end-to-end response time can be analyzed by considering each task's worst-base response time and period as follows: [55]

$$\ell_p = \sum_{k:o_k \in p} t_k + r_k. \tag{2.5}$$

where ℓ_p is the worst-case end-to-end latency for a certain path p and o_k is

an object, i.e, task or message, that belongs to p. The period and worst-case response time of each o_k is represented by t_k and r_k respectively. Although it is simple to implement this model, there is significant resource waste since each task always executes even when there is no fresh input, thus producing the same result as before.

2.4.3 Bandwidth Reservation

In the past, an ECU was a closed system which was solely controlled by a single supplier, i.e., single-function ECUs. However, in the multi-function ECUs era, independently developed SW components should be integrated into a single ECU, which arises a serious safety concern due to the interference caused among different SW components. Moreover, it makes it difficult for automakers and suppliers to trace the errors that incur a malfunction since an error can easily propagate into innocent SW components. Thus, the automotive safety standard ISO 26262 states the requirement of freedom from interference [56] for both spatial and temporal interferences. For the spatial interference, Hattendorf et al. [57] introduced a memory protection mechanism for multicore architectures based on MMU (Memory Management Unit) and MPU (Memory Protection Unit). Kim et al. [58] and Lee et al. [59] also proposed a memory management scheme based on the memory partitioning. For the temporal interference, Yun et al. [60] proposed a memory bandwidth reservation scheme that can bound the temporal interference caused by unbalanced memory bandwidth usage among cores. However, both works focus on per-core spatial/temporal isolation, not for per-SW component isolation. Ficek et al. [61] introduced mechanisms to implement

timing protection using the AUTOSAR's task execution time monitoring feature. However, in the AUTOSAR, runnables of SW components should be mapped to RTOS's tasks. In case runnables of different SW components are mapped to a single task, simply monitoring the task execution time does not provide the temporal isolation among SW components.

For the temporal isolation, Generalized Processor Sharing (GPS) [62] provides a ideal fluid model under the assumption that the traffic is infinitely time-divisible. In GPS, a weight is assigned to each client so that each client can enjoy the ideal fractional capacity of the HW resources, i.e., CPU capacity or network bandwidth. However, GPS is only an ideal model which is only useful as a benchmark. In a real system, scheduling decisions can be made at regular discrete time instants, i.e., ticks. EEVDF [63] is a proportional share resource allocation algorithm that closely follows the ideal fluid model keeping the gap between the actual time supply and the ideal time supply under a tick. Although EEVDF provides accurate resource provisioning, it suffers from too many context switches to emulate the ideal fluid model.

Resource reservation schemes [64, 65] can be also used, where each SW component reserves its required resource capacity in advance. The parameters for the resource reservation is no different from the parameters for servers for handling aperiodic tasks, i.e., reservation time and computation time. However, in the resource reservation, each SW component is coupled with its dedicated resource reservation whereas a server for aperiodic task handling is just a container that handles arbitrary aperiodic jobs. To be clear, the difference is that a server for handling aperiodic jobs guarantees tempo-

ral isolation from aperiodic jobs to periodic tasks whereas a resource reservation provides temporal isolation to each SW component running on it.

Besides the server scheduling algorithms for handling aperiodic jobs, other server algorithms have been proposed with the same motivation as resource reservation [66, 67, 68, 69, 70]. However, the common disadvantage of these algorithms is that they can only make a permanent fractional resource meaning that the reserved resource utilization U_s is always consumed even during the SW component is not active. Thus, traditional approaches are useful for providing permanent QoS guarantees to long running SW components, e.g., VOD servers and media players whereas they are not well-suited for complex control applications with a lot of fine-granular SW components with also complex temporal dependency among them. Kumar et al. [71] pointed out this problem and defined Demand Bound Server which is characterized by a demand bound function rather than period and budget. This dissertation shares the same motivation. For the traditional resource reservation schemes and server mechanisms providing temporal isolation, they do not efficiently utilize the underlying HW resources for a workload in which SW components are not always active.

Hierarchical scheduling shares a similar motivation of providing temporal isolation between subsystems sharing a HW resource. More specifically, it deals with a system represented as a tree of nodes, where each node is a subsystem with its own scheduler and workload as shown in Figure 3. For the hierarchical scheduling framework, Shin and Lee [1] proposed the periodic resource model (Π, Θ) where within Π time units, Θ time units are always reserved for that periodic resource. Based on that, they presented

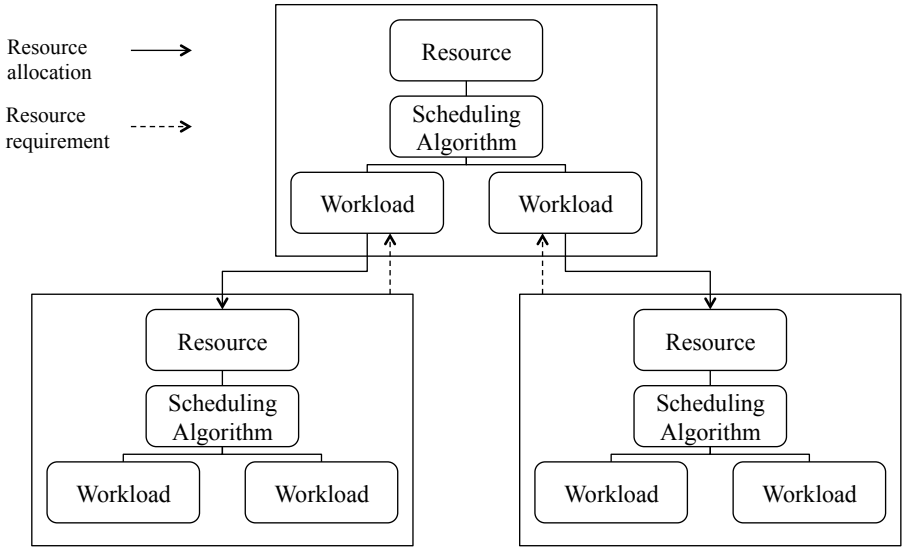


Figure. 3: Hierarchical scheduling framework [1]

the schedulability analysis method for subsystems with periodic workload under RM and EDF scheduling algorithms.Their framework supports compositional real-time guarantee since each parent scheduling model is computed from child scheduling models such that the real-time requirement of the parent scheduler is satisfied, if and only if, the real-time requirements of child schedulers are satisfied [1]. Although the periodic resource model is very useful for integrating legacy systems with periodic workload onto a single HW resource under hierarchical scheduling framework, they don't provide any fine-granular per-SW component temporal isolation.

2.5 System Optimization

When designing control algorithms in the model-based design tools, control engineers focus on the functional correctness of their control algorithms and do not care about the system configuration, i.e., ECUs, networks, and SW/HW mapping. However, this functional model eventually should be realized in a physical system configuration. Since the final physical system configuration is directly related to the manufacturing cost, it is important to find the optimal system configuration.

In order to transform model-based SW designs into a physical system configuration, Wang et al. [72] proposed a SW component allocation algorithm with multiple resource constraints. For the optimization, they used a branch and bound heuristic to reduce the computation time to find the solution. Based on the above allocation algorithm, Wang and Shin [41] also proposed a task construction algorithm where SW model is given by a DAG with synchronous data flow (SDF) execution semantic, which is similar to our SW model. However, they assume that the HW topology, i.e, number of ECUs and buses, are given a priori. Instead, they try to minimize the number of tasks and reduce the running time of their algorithm, which are not directly related to the manufacturing cost. They also used a heuristic algorithm that merges tasks with rate similarity and execution overlap to reduce the number of tasks.

Zheng et al. [52] and Zhu et al. [54] proposed a MILP-based optimal task allocation and priority assignment algorithm, which minimizes the end-to-end latencies assuming the periodic activation with asynchronous com-

munication model. This execution model is the most practical one which is widely used in the automotive industry. In this model, each task is independently triggered periodically. When a job of a task starts, it reads inputs from buffers asynchronously. The contents of a buffer is independently updated by the producer task. Even the recent data is not read by the consumer task yet, more recent data will update it periodically. Zhu et al. [53] also proposed a similar method with a different objective function, that is, maximizing the extensibility of the resulting system, i.e, how the system is robust to future upgrades. As a extensibility metric, they use the weighted sum of each task's execution time slack over its period. Although, above works are based on a practical execution model, they also assume a fixed HW topology and do not deal with minimizing the manufacturing cost. As the optimization method, they formulated their problems as linear programming problems and solved using solvers like CPLEX [73]. By using solvers, they can guarantee to find the optimal solution. However, if the optimization problem cannot be transformed to a linear programming problem, we cannot use solvers.

So far, system configuration optimization algorithms with various objectives and execution models are surveyed. Most researches assume a fixed HW topology, thus there is no room to optimize the system cost by their methods.

Chapter 3

System Model

In this chapter, we describe our system model, assumptions, terms and notations that are valid through the rest of this dissertation, unless otherwise stated.

3.1 High-level System Model

For building a new vehicle, the control engineers first design a set of end-to-end control transactions from sensors to actuators considering the control intention for the target vehicle dynamics. Each control transaction consists of a number of SW components, which is a logical unit of computation. When describing a control transaction, each transaction is represented by a DAG where each vertex is a SW component and edges are data dependencies among them. For each SW component, control engineers assume a fixed input-output delay when composing them such that the end-to-end delay of the transaction can be simply calculated by summing the delays of its SW components. Note that at this stage the actual hardware system configuration where the control transactions will be eventually implemented is not considered yet.

In order to implement such designed control transactions on networked ECUs, each SW component should be implemented on its corresponding

ECU. In case where SW components with direct data dependency reside within the same ECU, the data between them can be simply exchanged by a memory copy operation. On the other hand, an additional communication component, i.e., a CAN message, should be created between them to deliver the data between SW components on separate ECUs connected by a CAN bus. For the simplicity of explanation, we use the same terms and notations for such communication components as SW components. Thus, when mentioning SW components, we mean both SW components and communication components, unless otherwise stated.

After mapping the SW components to HW resources, i.e., ECUs and CAN buses, certain HW resources may have multiple SW components in it, thus there needs a resource sharing mechanism to handle concurrently running multiple SW components in a single HW resource. Although multiple SW components share a single HW resource, in order to respect the control engineer's design intention, each SW component's design time invariant delay property should be guaranteed as if it is running on its dedicated HW resource.

3.2 Assumptions

- All control transactions are periodically triggered with a fixed control period.
- All control transactions have an explicit deadline, which is less than or equal to the period.
- All control transactions are independent, that is, they have no data and

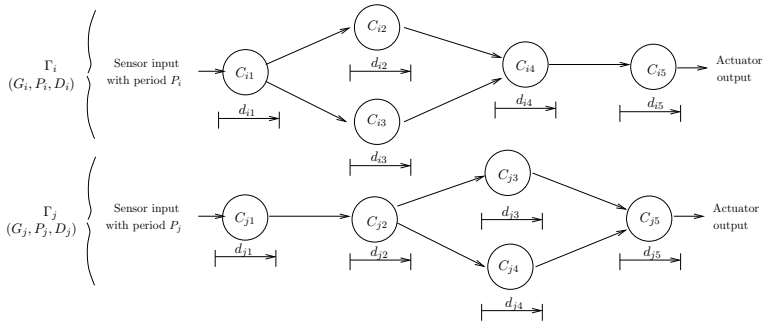
temporal dependency.

- All control transactions are offset-free meaning that we can arbitrarily adjust each control transaction's starting offset without hurting the control transaction's control performance.
- Each SW component's worst-case execution time for the target HW is known a priori.
- Each SW component has its own invariant delay given by the control engineers This assumption is similar to the LET of Giotto and Timed multitasking.
- Computation is fully preemptible whereas message delivery is non-preemptible.

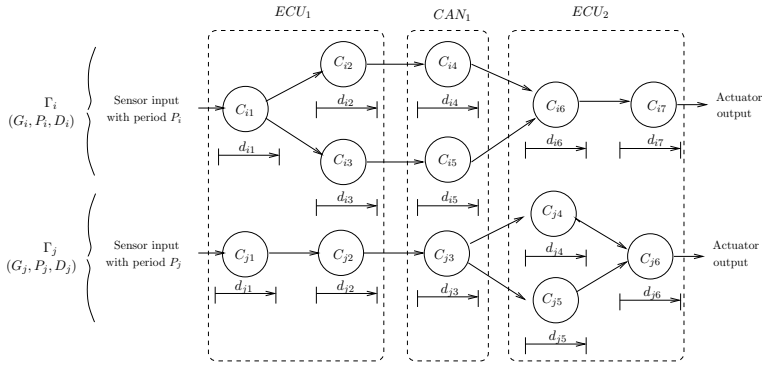
3.3 Models, Terms and Notations

Each end-to-end control transaction Γ_i is generally represented by a DAG $G_i = \{V_i, E_i\}$, where a set of vertices V_i is composed of $|\Gamma_i|$ SW components $\{C_{i1}, C_{i2}, \cdots, C_{i|\Gamma_i|}\}$ and a set of edges E_i is data dependencies among them $\{(C_{ik}, C_{il}) | C_{ik}, C_{il} \in V_i\}$. Each SW component C_{ik} is represented by a three-tuple (F_{ik}, d_{ik}, e_{ik}) where F_{ik} is its function, d_{ik} is the given invariant delay, and e_{ik} is the worst-case execution time. By d_{ik}, we specifically mean a constant time delay from inputs to outputs for C_{ik}. Figure 4(a) shows the DAGs for two example transactions Γ_i and Γ_j.

For each transaction Γ_i, the control engineers also determine its period P_i with which it should be periodically triggered to achieve its control



(a) Before mapping to a system configuration



(b) After mapping to a system configuration

Figure. 4: Example control transactions

objectives. Also, they determine the maximum tolerable delay D_i from sensing to actuation considering the physical dynamics of the intended control targets. As a consequence, a transaction Γ_i is represented by a three-tuple (G_i, P_i, D_i).

In this dissertation, we assume that we are given a set of N such end-to-end transactions, i.e., $\{\Gamma_1, \Gamma_2, \cdots, \Gamma_N\}$. For producing the final vehicle, these transactions should be implemented on ECUs connected by CAN buses. Thus, the automaker's system engineers need to determine the required ECUs, that is, $\{ECU_1, ECU_2, \cdots, ECU_M\}$, the required CAN buses, that is, $\{CAN_1, CAN_2, \cdots, CAN_L\}$, and also the mapping of each SW component C_{ik} to an ECU, which the auto industry calls the "system configuration problem". For now, just for the explanation purposes, we assume that such system configuration is already given. The system configuration problem will be addressed in Chapter 5.

Figure 4(b) shows an example system configuration where all the SW components of two transactions, i.e., $\{\Gamma_i, \Gamma_j\}$ are mapped to two ECUs, i.e., $\{ECU_1, ECU_2\}$, connected by a single CAN bus CAN_1. After the mapping, we need extra components for the CAN messages between SW components in different ECUs. For example, we have additional $\{C_{i4}, C_{i5}, C_{j3}\}$ on CAN_1 in Figure 4(b) compared to Figure 4(a). Please note that the indexes are reordered for the notational simplicity. More specifically, C_{i4} is a CAN message from C_{i2} in ECU_1 to C_{i6} in ECU_2. In the same way, C_{i5} and C_{j3} also represent CAN messages between SW components across ECU_1 and ECU_2.

Since the worst-case execution time of a CAN message can be obtained

by considering the message length and CAN's transmission rate, we use the same notation $C_{ik} = (F_{ik}, d_{ik}, e_{ik})$ where F_{ik} is the CAN message to be transmitted, d_{ik} is the required local delay, and e_{ik} is the transmission time of the message. When deciding the d_{ik}s of newly added SW components, we use a simple rule that the message delivery should be finished within its source SW component's local delay. Thus, the original SW component's local delay is split into two parts proportional to the worst-case execution times of the computation part and the communication part.

Chapter 4

Schedule Optimization

4.1 Introduction

When developing complex automotive control systems, control engineers first design a set of end-to-end control transactions from sensors to actuators by composing SW components' function F and invariant delay d. Then, for given set of control transactions, our objective is to realize them on networked ECUs guaranteeing each SW component's invariant delay property. In this chapter, we specifically deal with the SW component scheduling problem assuming that a fixed system configuration is given. More specifically, we are given a set of control transactions, a set of ECUs, a set of CANs, and SW/HW mapping. Then, our objective is to find the SW component scheduling which guarantees the component's delay property.

As our baseline scheduling algorithm, we make use of existing bandwidth reservation mechanisms. Nonetheless, since traditional bandwidth reservation mechanisms assign a permanent HW resource utilization to each SW component, it is not practical for the resource-constrained environment, as is such in the automotive industry. With the above motivation, this chapter presents a novel resource provisioning method based on our *active window based server scheduling*. Compared to the traditional bandwidth reservation mechanisms, our approach achieves a significantly higher resource ef-

ficiency by the following two ways [74]:

- **Exploiting intra-transaction component dependencies.** From the DAG for each control transaction, we extract the time windows during which each SW component actually needs the resource share. We call them *active windows* for each SW component. Instead of assigning a permanent resource utilization for the entire time domain, we assign the utilization only during the active windows. By this time-domain resource multiplexing, we can significantly increase the resource efficiency.

- **Exploiting inter-transaction component dependencies.** After considering the active windows, each control transaction's resource requirement (i.e., instantaneous utilization) is given as a function to time t. In order to check the schedulability of the entire system, we have to sum up every transaction's resource requirement at every t during the hyperperiod. When doing this, since each transactions' resource requirement may fairly fluctuate during the period, we can reduce the total resource requirement by adjusting the starting phases of control transactions. Intuitively, assuming two transactions, we can overlap a heavily loaded time duration of a transaction with a lightly loaded time duration of the other one.

The rest of this chapter is organized as follows: The next section gives the formal problem description. Section 4.3 describes the base scheduling algorithm and points out its limitation. Section 4.4 specifically presents our active window based server scheduling approach. Section 4.5 describes our

method for reducing algorithm complexity for the large hyperperiod case. Section 4.6 extends our approach to multicore ECUs. Section 4.7 describes how our approach can be applied to the CAN bus. Section 4.8 evaluates our optimization algorithms. Finally, Section 4.9 summarizes this chapter.

4.2 Problem Description

For given set of control transactions and given system configuration, our problem is to find SW component scheduling ensuring each SW component C_{ik}'s invariant delay property d_{ik}. More formally, the problem is described as follows:

Problem Description: Suppose that we are given a set of transactions $\{\Gamma_1, \Gamma_2, \cdots, \Gamma_N\}$, a set of ECUs $\{ECU_1, ECU_2, \cdots, ECU_M\}$, a set of CANs $\{CAN_1, CAN_2, \cdots, CAN_L\}$, a set of mappings from SW components to ECUs/CANs, and a set of per-component specifications (F_{ik}, d_{ik}, e_{ik})s for C_{ik}s. Our problem is how to guarantee invariant delay property to every SW component C_{ik} such that its function F_{ik} can be executed (or its message F_{ik} can be transmitted) and the outputs are produced after the constant delay d_{ik}.

4.3 Base Scheduling Algorithm

In order to schedule each SW component C_{ik} guaranteeing the constant input-output delay of d_{ik}, our baseline approach is to dedicate the required HW utilization to each C_{ik} permanently. Whenever each SW component C_{ik} arrives at time t, a dedicated server S_{ik} is created with its relative deadline equal to d_{ik} and its execution budget e_{ik}. Then, each server S_{ik} is sched-

uled under EDF with its absolute deadline $t + d_{ik}$ competing the underlying HW resource until its execution budget is exhausted. While S_{ik} is taking the chance of using the target HW resource, it serves C_{ik} with its execution budget. Since each server S_{ik}'s utilization $u_{ik} = e_{ik}/d_{ik}$, we can calculate each HW resource R's utilization as

$$\sum_{C_{ik} \in R} \frac{e_{ik}}{d_{ik}}. \tag{4.1}$$

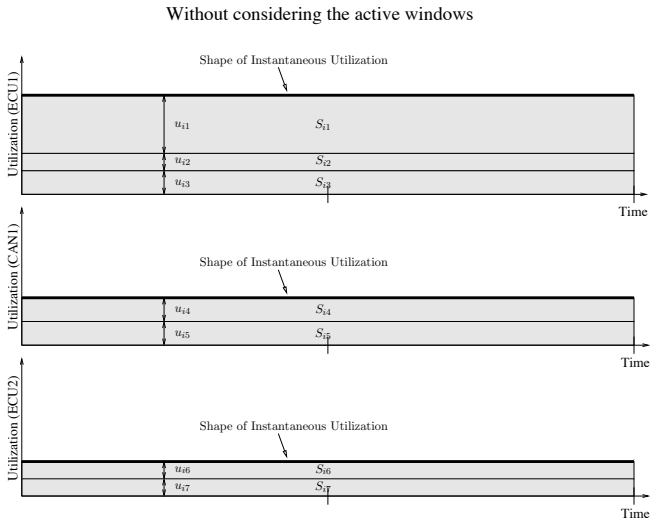
Then, every SW component C_{ik} in the system is provided with its required e_{ik} during its relative deadline d_{ik} whenever it arrives, if and only if every R's utilization $\leq 100\%$ assuming preemptible resources. For non-preemptive HW resources like CAN, a slight modification is required as presented in Section 4.7.

With the above mechanism, by serving each SW component C_{ik} with its dedicated server S_{ik}, each instance of C_{ik} triggered at t can be completed before $t+d_{ik}$. The output of C_{ik} that comes out earlier than $t+d_{ik}$ is buffered and issued exactly at $t + d_{ik}$ using the *timed-output* semantic. Thus, a S_{ik} combined with the timed-output semantic can guarantee the constant delay d_{ik} for each SW component C_{ik}. From now on, when mentioning a server, it refers to the above modified version of server with the the timed-output semantic.

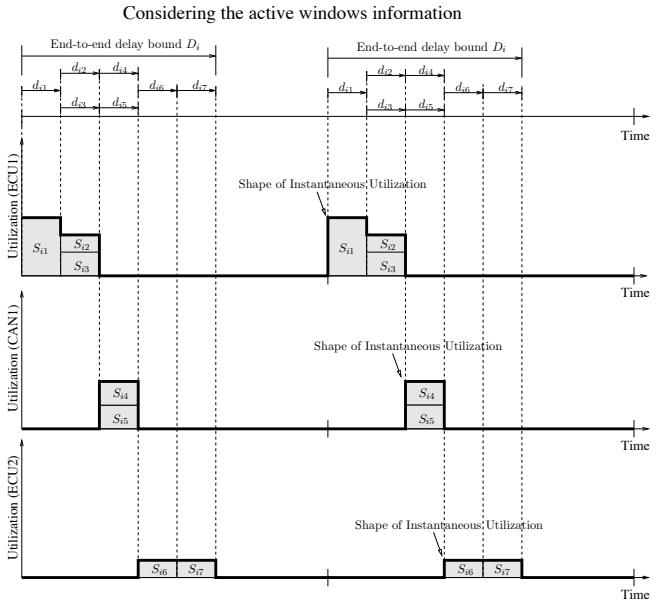
4.4 Schedule Optimization with Active Window based Server Scheduling

The base scheduling approach of assigning a permanent fractional utilization to each SW component can quickly use up the entire capacity of HW resources. We solve this problem by further dividing the HW capacity in the temporal domain using the notion of "active window" for each SW component. Due to the precedence relations among the SW components of the given transaction graphs, a SW component is active only for a short time window, called an active window. Thus, instead of assuming arbitrary arrival of C_{ik}s as in Section 4.3, we can create S_{ik}s at the start of its active windows in a time-triggered manner. Then, it is sufficient to account for the utilization u_{ik} to a SW component only for its active windows. This section explains our offline optimization process exercising active windows in order to determine when S_{ik} should be turn on and off. This offline computed on/off timetable is given to the kernel that actually realizes the on/off based active window server scheduling in runtime.

The active windows of SW components are exercised in two aspects: (1) intra-transaction relations and (2) inter-transaction relations. First, the intra-transaction relations are the relations among active windows of SW components belonging to the same transaction. For the example transaction of Γ_i in Figure 4(b), we can extract the active window relations among its SW components as in Figure 5(b). From this active window relations, we can spread out their required utilizations on their corresponding active windows as in Figure 5(b) rather than permanently assigning them as in



(a) Before intra-transaction optimization



(b) After intra-transaction optimization

Figure. 5: Intra-transaction optimization

Figure 5(a). By this way, we can reduce the total instantaneous utilization.

Second, inter-transaction relations are the relations among active windows of SW components belonging to different transactions. These relations can be exercised from the fact that we can adjust the relative phase between two independent transactions without affecting their control performance. For the example of two transactions Γ_i and Γ_j in Figure 4(b), we can shift the phase of Γ_j such that the utilization assignments can be more spread out as in Figure 6(b) rather than simply stacking up the utilizations of Γ_j on top of Γ_i as in Figure 6(a). By doing this inter-transaction optimization, the maximum peak utilization can be reduced as utilizations are more evenly distributed in the time domain after the optimization.

In order to formally exercise those intra-transaction and inter-transaction relations, let us introduce the following definitions. After spreading utilizations of Γ_i using the intra-transaction active window relations as in Figure 5(b), Γ_i's resource requirement at time $t > 0$ is represented as a timed-vector $L^{\Gamma_i}(t) = (L^{\Gamma_i}_{ECU_1}(t), L^{\Gamma_i}_{ECU_2}(t), \cdots, L^{\Gamma_i}_{ECU_M}(t), L^{\Gamma_i}_{CAN_1}(t), L^{\Gamma_i}_{CAN_2}(t), \cdots, L^{\Gamma_i}_{CAN_L}(t))$ where $L^{\Gamma_i}_R(t)$ is the sum of utilizations for all the Γ_i's SW components mapped to HW resource R that are active at t. For the example transaction Γ_i in Figure 4(b), $L^{\Gamma_i}(t)$ is depicted in Figure 7(a). $L^{\Gamma_j}(t)$ for Γ_j in Figure 4(b) can be similarly defined. If we shift the phase of Γ_j by θ_j, its resource requirement can be represented as $L^{\Gamma_j}(t - \theta_j)$ when $t \geq \theta_j$ and zero otherwise. This is depicted in Figure 7(b). Therefore, if we use $\{\theta_1, \theta_2, \cdots, \theta_N\}$ as their phases of $\{\Gamma_1, \Gamma_2, \cdots, \Gamma_N\}$, the system-wide resource requirement can be represented as $L(t) = \sum_{i=1}^{N} L^{\Gamma_i}(t - \theta_i)$. Figure 7(c) depicts $L(t) = L^{\Gamma_i}(t) + L^{\Gamma_j}(t - \theta_j)$ for the two example trans-

(a) Before inter-transaction optimization

(b) After inter-transaction optimization

Figure. 6: Inter-transaction optimization

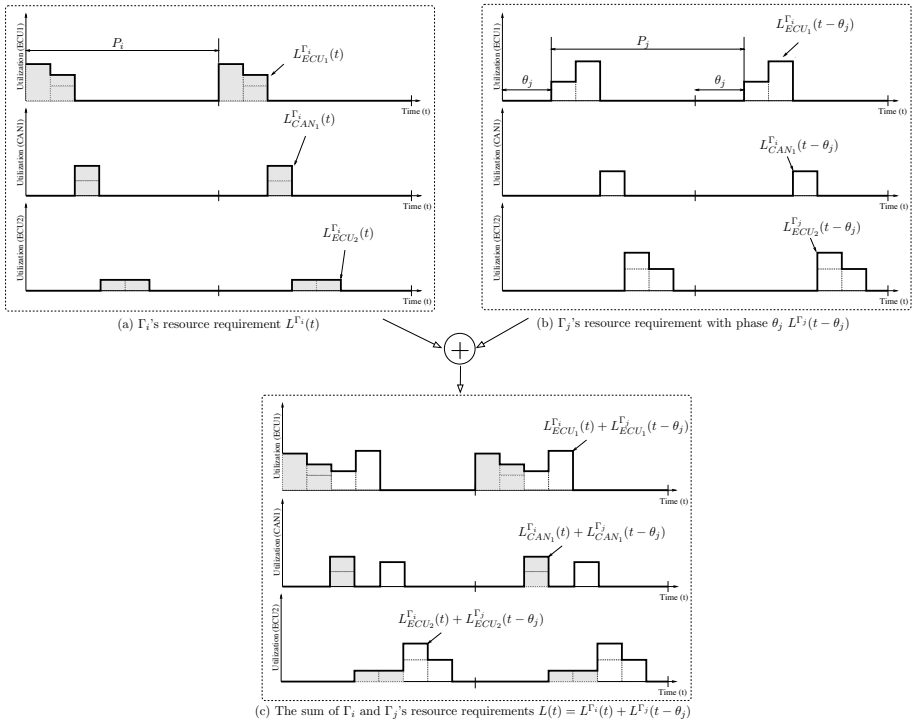
actions Γ_i and Γ_j in Figure 4(b).

With these definitions, we can formally describe our problem as finding $\{\theta_1, \theta_2, \cdots, \theta_N\}$ such that the $MaxPeak$ can be minimized, where $MaxPeak$ is the peak sum of utilizations along the timeline for the HW resource with the highest peak, that is,

$$MaxPeak = \max_{1 \le m \le M, 1 \le l \le L} \left\{ \max_{t>0} \left(\sum_{i=1}^{N} L_{ECU_m}^{\Gamma_i}(t - \theta_i) \right), \max_{t>0} \left(\sum_{i=1}^{N} L_{CAN_l}^{\Gamma_i}(t - \theta_i) \right) \right\}. \quad (4.2)$$

We are interested in minimizing $MaxPeak$ since the resulting solution is feasible if the sum of assigned utilizations is under the utilization bound at all times. For the ECU resource, the utilization bound is 100%. The utilization bound for the CAN bus also can be easily found by the method presented in Section 4.7. Note that this is a variation of the bin packing problem where a complex-shaped item, that is, a phase shiftable resource requirement of Γ_i, i.e., $L^{\Gamma_i}(t - \theta_i)$, needs to be placed on the timeline of multiple bins, that is, M ECUs and L CANs, such that $MaxPeak$ of Eq. 4.2 can be minimized. Since it is a combinatorial NP-hard problem [75], we use several heuristics to find a near optimal solution.

For the given N control transactions, our heuristic algorithm for the phase shifting optimization is described in Table 1. As its inputs, this algorithm takes the set of harmonic transactions $\{\Gamma_1, \Gamma_2, \cdots, \Gamma_N\}$, set of ECUs $\{ECU_1, ECU_2, \cdots, ECU_M\}$, set of CANs $\{CAN_1, CAN_2, \cdots, CAN_L\}$, and the mappings from SW components to ECUs and CANs. As the outputs,



(a) Γ_i's resource requirement $L^{\Gamma_i}(t)$

(b) Γ_j's resource requirement with phase θ_j $L^{\Gamma_j}(t-\theta_j)$

(c) The sum of Γ_i and Γ_j's resource requirements $L(t) = L^{\Gamma_i}(t) + L^{\Gamma_j}(t-\theta_j)$

Figure. 7: Resource requirement functions

Table. 1: Procedure for finding the near optimal phases for a set of transactions

FindOptimalPhases: Find the optimal phases $\{\theta_1, \theta_2, \cdots, \theta_N\}$ for the transactions in the same harmonic group $\{\Gamma_1, \Gamma_2, \cdots, \Gamma_N\}$

Input: Set of harmonic transactions $\{\Gamma_1, \Gamma_2, \cdots, \Gamma_N\}$,
Set of ECUs $\{ECU_1, ECU_2, \cdots, ECU_M\}$,
Set of CANs $\{CAN_1, CAN_2, \cdots, CAN_L\}$, and
Mapping from SW components to ECUs and CANs

Output: Near optimal phases $\{\theta_1, \theta_2, \cdots, \theta_N\}$

begin procedure

1. Sort $\{\Gamma_1, \Gamma_2, \cdots, \Gamma_N\}$ as the decreasing order of $Size(\Gamma)$
2. $\theta_1 = 0$
3. **for** $i = 2$ **to** N **do**
4. $\quad L(t) = \sum_{a=1}^{i-1} L^{\Gamma_a}(t - \theta_a)$
5. $\quad \theta = 0$
6. $\quad MaxPeak^{smallest} = \infty$
7. $\quad \theta^{best} = 0$
8. $\quad$ **while** $(\theta \leq P_i)$ **do**
9. $\quad\quad L'(t) = L(t) + L^{\Gamma_i}(t - \theta) = (L'_{ECU_1}(t), \cdots, L'_{ECU_M}(t), L'_{CAN_1}(t), \cdots, L'_{CAN_L}(t))$
10. $\quad\quad MaxPeak^{new} = \max_{1 \leq m \leq M, 1 \leq l \leq L} \{\max_{t>0} L'_{ECU_m}(t), \max_{t>0} L'_{CAN_l}(t)\}$
11. $\quad\quad$ **if** $MaxPeak^{new} < MaxPeak^{smallest}$ **then**
12. $\quad\quad\quad MaxPeak^{smallest} = MaxPeak^{new}$
13. $\quad\quad\quad \theta^{best} = \theta$
14. $\quad\quad$ **end if**
15. $\quad\quad \theta = \theta + 1$
16. $\quad$ **end while**
17. $\quad \theta_i = \theta^{best}$
18. **end for**

end procedure

it produces the near optimal phases $\{\theta_1, \theta_2, \cdots, \theta_N\}$. In order to efficiently find the near optimal solution, the algorithm uses a well known "largest item first packing" heuristic [76], which generally works well since a large item is easy to pack in the beginning but hard to pack at the end. With this regard, we define the size of a transaction Γ_i as follows:

$$Size(\Gamma_i) = \frac{GrossArea(\Gamma_i)}{P_i} \tag{4.3}$$

where $GrossArea(\Gamma_i)$ is $u_{i1}d_{i1} + u_{i2}d_{i2} + \cdots + u_{i|\Gamma_i|}d_{i|\Gamma_i|}$. With this $Size(\Gamma)$ definition, Line 1 sorts the transactions in the harmonic group as the decreasing order of their sizes. For such sorted transactions, let us use the same notation $\{\Gamma_1, \Gamma_2, \cdots, \Gamma_k\}$ where Γ_1 has the largest size, Γ_2 the second, and so on.

After the sorting, Line 2 fixes the phase of Γ_1 to zero, that is, $\theta_1 = 0$. Then, the for loop from Line 3 to line 18 iteratively places the transactions, i.e., determines the phases of transactions from Γ_2 to Γ_k. For the iteration of Γ_i, the phases of the first $i-1$ largest size transactions are already determined as $\theta_1, \theta_2, \cdots, \theta_{i-1}$. Thus, the cumulated resource requirement so far is given in Line 4 as follows:

$$L(t) = \Bigg(\sum_{a=1}^{i-1} L_{ECU_1}^{\Gamma_a}(t-\theta_a), \quad \sum_{a=1}^{i-1} L_{ECU_2}^{\Gamma_a}(t-\theta_a), \cdots, \sum_{a=1}^{i-1} L_{ECU_M}^{\Gamma_a}(t-\theta_a), \quad \sum_{a=1}^{i-1} L_{CAN_1}^{\Gamma_a}(t-\theta_a), \sum_{a=1}^{i-1} L_{CAN_2}^{\Gamma_a}(t-\theta_a), \quad \cdots, \sum_{a=1}^{i-1} L_{CAN_L}^{\Gamma_a}(t-\theta_a) \Bigg). \tag{4.4}$$

Lines 5 to 7 initialize the variables for finding the best θ_i.

Then, the while loop from Line 8 to 16 repeatedly checks the phase θ_i of Γ_i in the range from 0 to P_i. For a specific $\theta_i = \theta$, Line 9 computes the new cumulative resource requirement as $L'(t) = L(t) + L^{\Gamma_i}(t-\theta)$ considering the addition of Γ_i. Then, Line 10 computes $MaxPeak^{new} = \max_{1\le m\le M, 1\le l\le L}\{\max_{t>0} L'_{ECU_m}(t), \max_{t>0} L'_{CAN_l}(t)\}$. If this $MaxPeak^{new}$ is smaller than the smallest $MaxPeak^{smallest}$ so far, Line 12 updates the

smallest $MaxPeak^{smallest}$ and Line 13 updates the current best θ^{best}. After exiting this while loop, in Line 17, θ^{best} is the best value for θ_i that minimizes the $MaxPeak$.

After completing this procedure, we can finally find the near optimal phases $\theta_1, \theta_2, \cdots, \theta_N$ for the given set of transactions. For the system-wide feasibility check, we find the system-wide maximum peak utilization by checking the peak of every HW resource R (ECU or CAN). If this maximum peak utilization is under the utilization bound for every HW resource, we can conclude that the resulting solution is feasible. Once the feasible solution is found, the active windows of all the SW components are finally fixed. This information is given to each ECU's kernel so that it can provide servers for the ECU and its connected CANs only for the given active windows in runtime.

4.5 Handling Algorithm Complexity of Workload with Large Hyperperiod

Since our schedule optimization algorithm in Section 4.4 should check the entire hyperperiod of given transactions to calculate the maximum peak utilization as varying the starting offset of each transaction, the complexity of our algorithm is directly affected by the length of the hyperperiod. In the worst case, the hyperperiod can become abnormally large especially when the periods are given as different prime numbers. Even though such case may not exists in realistic workloads, this section presents a method that can handle such abnormal cases in order to make our solution complete.

The problems that can be caused by an arbitrarily large hyperperiod can be identified as follows:

- *Offline time complexity.* In the offline schedule optimization process, the computation time can become intolerable since the phase shifting algorithm should always check the entire hyperperiod at every phase for every transaction. Moreover, as we will present in Chapter 5, our system optimization process repeatedly calls the schedule optimization process, thus in order to use the system optimization method practically, the time for the schedule optimization process should be minimized.

- *Online space complexity.* The resulting timetable produced by the schedule optimization process may become large according to the hyperperiod. Since this timetable should be transferred to the component execution kernel in ECUs with small memory, the timetable should be kept as small as possible.

The basic idea for solving the above problems is to conduct the phase shift optimization only for the transactions whose periods are divisible, that is, the ones with harmonic periods. Based on this idea, we first divide the given set of transactions $\{\Gamma_1, \Gamma_2, \cdots, \Gamma_N\}$ into a number of harmonic groups, which is a set of transactions with harmonic periods. After the grouping, we independently conduct the phase shifting optimization for each harmonic group. Then, the system-wide maximum peak utilization is conservatively computed by adding peaks obtained for all the harmonic groups. This is based on the worst case scenario where the peaks of all the harmonic groups

can occur at the same time. If this conservatively added peak is under the utilization bound for every HW resource, we can conclude that the resulting solution is feasible.

Due to the conservative estimation by the above harmonic grouping method, the estimated maximum peak utilization becomes pessimistic since the worst case scenario may hot happen. However, it can effectively bound both the offline time complexity and the online space complexity by the longest period. Thus, we can practically handle arbitrarily workloads with very large hyperperiods.

4.6 Supporting Multicore ECUs

Regarding the case of a multicore ECU with two or more identical cores, in addition to the active window scheduling problem in Section 4.4, we also have to decide the core on which the server S_{ik} of C_{ik} should be placed. This combined problem of active window scheduling and core mapping can be addressed as a "nested bin packing" where the outer bin packing solves the active window scheduling, that is, phases of all the transactions using the algorithm in Table 1 and the inner bin packing solves the core mapping for S_{ik} of SW components.

For this nested bin packing, we can slightly modify the algorithm in Table 1 by calling the inner bin packing procedure right after Line 17 that determines the best θ_i of Γ_i. In order to explain the inner bin packing procedure, let us introduce two more notations: (1) The resource requirement of C_{ik} on its mapped ECU_m when Γ_i's phase is θ_i is represented by $L_{ECU_m}^{C_{ik}}(t-$

θ_i). It is similarly defined as $L^{\Gamma_i}_{ECU_m}(t-\theta_i)$ except that $L^{C_{ik}}_{ECU_m}(t-\theta_i)$ represents the resource requirement only by the specific component C_{ik} rather than the transaction Γ_i. (2) The cumulated resource requirement by all the SW components mapped to the g-th core of ECU_m, i.e., $core_{mg}$, is represented by $L_{core_{mg}}(t)$. It is similarly defined as $L_{ECU_m}(t)$ except that $L_{core_{mg}}(t)$ represents the requirement only for $core_{mg}$ rather than the entire ECU_m.

With these notations, suppose that the algorithm in Table 1 addressed $\Gamma_1, \Gamma_2, \cdots, \Gamma_{i-1}$ so far and the the current cumulated resource requirement for each core $core_{mg}$ of ECU_m is given as $L_{core_{mg}}(t)$. Then, the iteration of the for loop for Γ_i determines θ_i at Line 17 and then calls the inner bin packing procedure. The inner bin packing procedure handles all the SW components of Γ_i one by one according to their indexing order. For a SW component C_{ik}, the inner bin packing procedure tries to pack it in a way of evenly distributing the workload to all the cores as much as possible. More specifically, for C_{ik}, the inner bin packing procedure tries every core $core_{mg}$ as a candidate core and computes its updated resource requirement $L'_{core_{mg}}(t)$ as follows:

$$L'_{core_{mg}}(t) = L_{core_{mg}}(t) + L^{C_{ik}}_{ECU_m}(t-\theta_i).$$

Then, the procedure actually maps C_{ik} to the core, say $core_{mg}$ whose peak, i.e., $\max_{t>0} L'_{core_{mg}}(t)$, is the smallest. The inner bin packing procedure continues this core mapping for all the SW components of Γ_i.

This modified algorithm for the multicore ECU case is applied to all the

harmonic groups. For the system-wide feasibility check, for each core, the peaks obtained for all the harmonic groups are conservatively cumulated as before. If this conservatively cumulated peak is under 100% for every core of every ECU, we can conclude that the resulting solution is feasible.

4.7 Supporting the CAN Bus

For our schedule optimization mechanism to be used for HW resources as shared communication buses like CAN, the above mechanism need to be slightly tuned due to two reasons: (1) EDF scheduling and (2) non-preemptiveness of a message transmission. Since CAN is originally designed for the fixed-priority scheduling, the utilization bound is as low as 25% [22] [1]. Due to this low utilization bound, we cannot effectively utilize a CAN bus. Thus, we try to use the EDF scheduling instead of the fixed-priority scheduling on a CAN bus. Fortunately, there are many previous researches for using the EDF scheduling on a CAN bus [23, 24, 25, 29]. There are also more general approaches [77, 78] that are applicable to various communication buses including CAN, but not limited to CAN. However, since a CAN message transmission is not preemptible, we cannot simply use the preemptive EDF scheduling's 100% utilization bound even when we are using the EDF scheduling on a CAN bus. Thus, in the remaining part of this section, we focus on deriving the utilization bound of non-preemptive EDF scheduling on a CAN bus, which is not clearly addressed in the literature

[1] The utilization bound of the fixed-priority scheduling is about 69.3% for the preemptive case and 0% for the non-preemptive case [22]. However, [22] found a more exact utilization bound (i.e., 25%) considering the maximum message length limit of CAN.

yet.

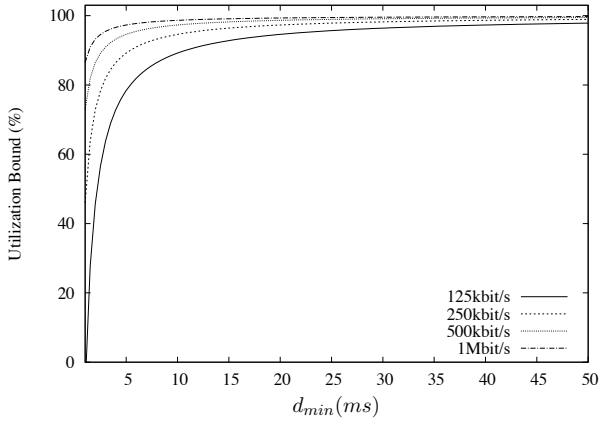
According to the ISO-11898 standard, the maximum user data length of CAN is defined as 8 bytes. That means that the maximum message transmission time of a CAN message can be bounded. More specifically, considering both user data and protocol overheads, the maximum message transmission time C_m can be calculated by the following equation [21]:

$$C_m = \left(g + 8s_m + 13 + \left\lfloor \frac{g + 8s_m - 1}{4} \right\rfloor \right) \tau_{bit} \tag{4.5}$$

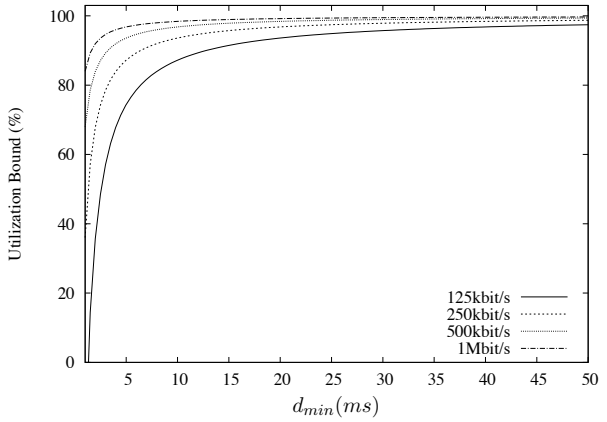
where g is 34 for CAN 2.0A or 54 for CAN 2.0B specification, s_m is the maximum length of user data in bytes, that is, 8 bytes for both CAN 2.0A and CAN 2.0B, and τ_{bit} is the transmission time for a single bit. Once the CAN protocol version and the transmission rate are decided, C_m is simply given as a constant by this equation. Then, the utilization bound of non-preemptive EDF scheduling U can be calculated by the following equation:

$$U = 1 - \frac{C_m}{d_{min}} \tag{4.6}$$

where d_{min} is the smallest relative deadline for all the tasks, which is a workload characteristic. For the detailed proof, readers are referred to [79], which extended the preemptive EDF utilization bound to more general task models with non-preemptive code sections. By this equation, the utilization bound of CAN is given as a function of d_{min}. Figure 8 shows the utilization bound as varying d_{min} and transmission rate for both CAN2.0A and CAN2.0B specifications. For example, for CAN2.0A, when d_{min} is 1 ms, the utilization bound is 74%, which is significantly higher than the fixed-



(a) CAN2.0A



(b) CAN2.0B

Figure. 8: CAN bus utilization bound for the non-preemptive EDF scheduling

priority scheduling case, i.e., 25%. In case that d_{min} is 5 ms, the utilization bound reaches 94.6%.

By employing the EDF scheduling to the CAN bus and knowing the utilization bound of a given CAN bus which considers the non-preemptiveness of a CAN message delivery and a workload characteristic d_{min}, the only difference when applying our technique to the CAN bus is that the CAN bus utilization bound should be used instead of using 100% utilization bound of preemptive EDF scheduling.

4.8 Experiment

This section validates the proposed approach. We first conduct extensive simulations with synthesized workload in order to see the improvement by the proposed approach in wide spectrums of workload parameters. For this simulation study, we consider a future automotive system equipped with three powerful ECUs, i.e., $ECU1$, $ECU2$, and $ECU3$, which are connected with a CAN bus, i.e., CAN_1. On top of this hardware configuration, we make a set of synthesized transactions with the following parameters:

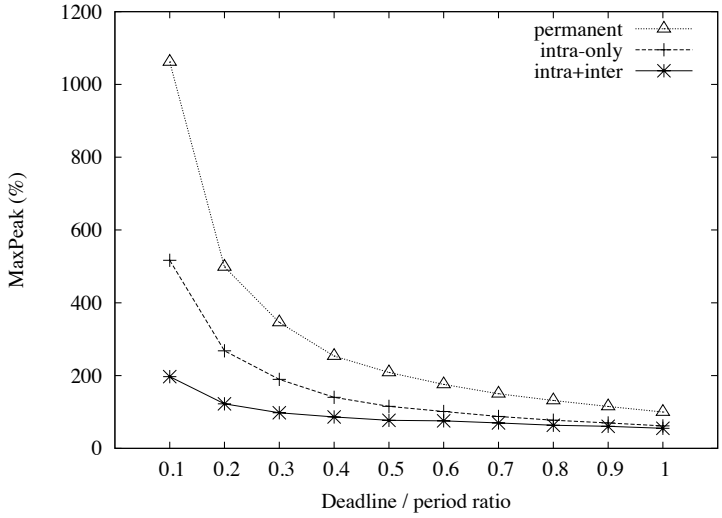
- The number of transactions N whose default value is 16 if not otherwise mentioned.
- The deadline to period ratio D_i/P_i whose default value is 0.5 if not otherwise mentioned. With this ratio, the end-to-end deadline D_i of each transaction Γ_i is simply determined from its period P_i.

For each transaction generated with the above parameters, we create a randomly generated DAG consisting of three to nine SW components. The execution time e_{ik} and delay bound d_{ik} of each SW component C_{ik} are determined such that its required instantaneous utilization $u_{ik} = \frac{e_{ik}}{d_{ik}}$ ranges from 3% to 10% for SW components in ECUs and from 1% to 5% for message transmissions in CANs and also the sum of delay bounds $\sum_{j=1}^{|\Gamma_i|} d_{ik}$ of all the SW components of Γ_i is equal to the end-to-end deadline D_i.

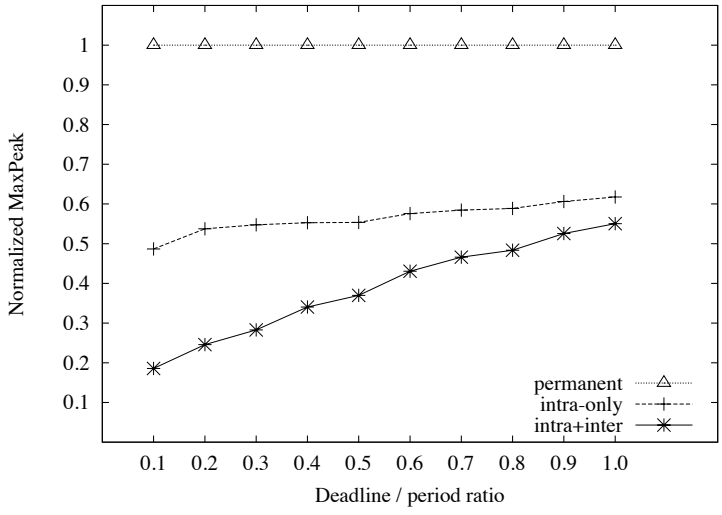
With this synthesized workload, we compare the following three approaches:

- The approach that assigns a server to each SW component without considering its active window. This approach is denoted by *permanent*.
- The approach that spreads out the server assignments in the temporal domain using only intra-transaction relations of active windows. This approach is denoted by *intra-only*.
- The approach that further spreads out the server assignments using both intra-transaction and inter-transaction relations of active windows. This approach is denoted by *intra+inter*.

Figure 9(a) compares the three approaches in terms of $MaxPeak$ as varying the deadline to period ratio D_i/P_i from 0.1 to 1. The $MaxPeak$ is the peak sum of the assigned server utilizations along the timeline for the ECU (or CAN) with the largest peak as defined in Eq. 4.2. The graph shows the average of 100 randomly synthesized transaction sets. From this



(a) $MaxPeak$



(b) Normalized $MaxPeak$

Figure. 9: Comparison of $MaxPeak$ with varying D_i/P_i

graph, for the entire range of D_i/P_i, we can observe significant reduction of $MaxPeak$ by *intra-only* and further significant reduction by *intra+inter* compared to *permanent*. This reduction is because of spreading out the server assignment along the timeline using the intra-transaction and inter-transaction relations of active windows. Another interesting observation is that the reduction of $MaxPeak$ by *intra+inter* becomes more significant as D_i/P_i gets smaller. This is because we can enjoy more room of inter-transaction based temporal multiplexing of servers when D_i is much smaller than P_i. This trend can be more clearly observed in Figure 9(b) that shows the normalized $MaxPeak$ values relative to *permanent*. In the most favorable case to *intra+inter*, that is, when D_i/P_i is 0.1, the *intra+inter* shows 5 times smaller $MaxPeak$ than *permanent*. Even in the most unfavorable case, that is, when $D_i/P_i = 1.0$, the *intra+inter* approach still shows under 60% $MaxPeak$ of *permanent*.

The reduction of $MaxPeak$ by *intra-only* and *intra+inter* as observed so far allows us to accommodate a significantly larger number of transactions keeping the system feasibility. In order to show this, we add transactions into the system one by one until none of the ECUs' and CANs' peak becomes over their utilization bound. Figure 10 shows the number of accepted transactions by the three approaches. The numbers are the average of 100 experiments conducted as changing the random seed. By taking advantage of only intra-transaction relations, *intra-only* can accept two times more transactions than *permanent*. Furthermore, by taking advantage of both intra-transaction and inter-transaction relations, *intra+inter* can accept about four times more transactions than *permanent*. If we take a de-

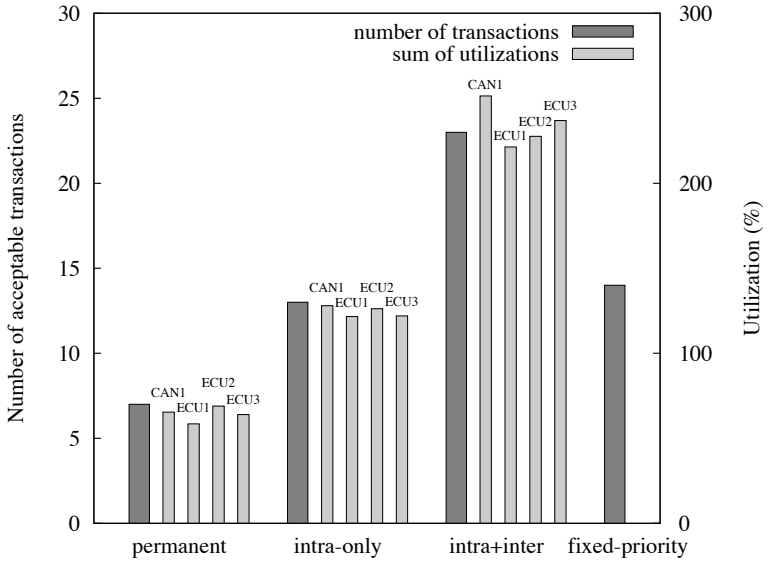


Figure. 10: Comparison of the number of acceptable transactions

tailed look on the sum of utilizations of the accepted SW components on each ECU (or CAN), *permanent* can accept only up to about 70% share requirement while *intra+inter* can effectively accept up to about 250% share requirement by the multiplexing in the temporal domain. As a reference, Figure 10 also shows the number of accepted transactions when all the ECUs and CANs run tasks with the *fixed-priority* scheduling. For the acceptance test of the *fixed-priority* case, the well known holistic end-to-end schedulability analysis [4] is used. The results show that our *intra+inter* approach even beats the fixed-priority scheduling framework in terms of the number of accepted transactions. This is because of the pessimism of the holistic end-to-end schedulability analysis. More importantly, such analysis has to view the entire system as a whole and hence is limited in handling the ever

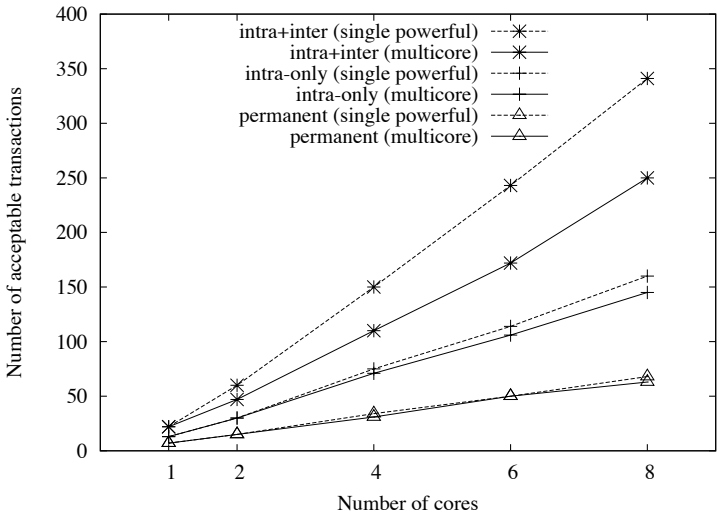


Figure. 11: Comparison of the number of acceptable transactions with varying number of cores per ECU

increasing complexity of the future automotive systems.

In order to see how our technique efficiently utilize the multicore systems, Figure 11 compares the number of acceptable transactions as varying the number of cores from one to eight. To see the fragmentation overhead caused by the multicore bin packing, we compare the three approaches' results with the results of using two, four, six, eight-times faster single powerful ECU instead of two, four, six, eight-cores ECU respectively. Although the fragmentation overhead of *intra+inter* is relatively larger than other approaches, the number of acceptable transactions linearly scales up to eight cores. The main reason of the fragmentation overhead is the two-step bin packing mechanism which do not consider the phase and the core placement at the same time. However, this is our design choice by observing that if we

consider the phase and the core placement at the same time, the run-time overhead becomes too high to be used practically.

4.9 Summary

For given set of transactions and given system configuration, this chapter presents a SW component scheduling algorithm that guarantees the component's invariant delay property. Starting from the base scheduling algorithm that assigns permanent HW resource utilization to each SW component, our approach employs the notion of active windows for each SW component during which each SW component actually needs the HW resource utilization. By accounting for the utilizations only during the active windows of SW components, our SW component scheduling algorithms can achive significantly higher resource efficiency. Furthremore, we optimize the resource efficiency again by controlling the starting phases of control transactions.

Since our assumed transaction model consists of control transactions with its SW components sequentially executing one after the other and each SW component has its fixed input-output delay, we can easily extract each SW component's active windows from the control transactions. Then, exploiting those intra-transaction component dependencies, we assign servers to SW components exactly at the starting points of active windows. For independent transactions, we also exploit the inter-transaction phase dependencies in order to furthre optimize the maximum peak utilization in order to accept as many transactions as possible with the given system configuration.

For handling workloads with abnormally large hyperperiod, we also propose a harmonic grouping method that can practically bound the complexity of our schedule optimization algorithm. We also extend our approach for handling multicore ECUs and CAN buses.

Chapter 5

System Optimization

5.1 Introduction

So far, we have presented our SW component scheduling algorithm assuming that the system configuration of the target automotive system is given as an input. This chapter presents our system optimization method that finds the minimal cost system configuration with a feasible SW component scheduling for given set of control transactions. Please note that by system configuration we refer to a set of required ECUs, a set of required CANs, and SW/HW mapping.

Since our system optimization is a combinatorial NP-hard problem, solution space search algorithms should be employed. One possible solution is to formulate our system optimization problem as a linear programming problem and use solvers [73] to find the optimal solution. In the literature, many optimization problems are solved this way [52, 54, 53]. However, in order to formulate our problem as a linear optimization problem, every equations including the constraints and optimization objective should be represented as linear equations. Unfortunately, our offline SW component scheduling optimization itself is a combinatorial optimization problem that cannot be represented as a linear programming problem, thus our system optimization problem also cannot be represented as a linear programming

problem.

Under the above limitation, we devise a heuristic algorithm that efficiently searches the problem space to find the near-optimal solution. Our heuristic algorithm starts with a initial system configuration where each SW component is dedicated its own ECU, thus having the most remaining capacity to accept more transactions and also the most expensive system cost. Then, our optimization process iteratively merges HW resources, thus reducing system cost, until the remaining capacity reaches zero. By our heuristic algorithm, we can find a near-optimal solution in a reasonable time bound as is shown in Section 5.4.

The rest of this chapter is organized as follows: Section 5.2 presents the formal problem description. Section 5.3 gives our solution approach. Section 5.4 evaluates our approach. Finally, Section 5.5 summarizes this chapter.

5.2 Problem Description

For given set of control transactions, our problem is to find a minimal cost system configuration that has a feasible SW component scheduling under the scheduling method described in Chapter 4. Figure 12 shows our system optimization problem illustratively. In our problem, we have a solution space where each solution represents a system configuration. For each solution, we can conduct our scheduling optimization process in Chapter 4 with the given set of transactions. As results, the scheduling optimization process yields (1) schedulability room that represents the remaining capac-

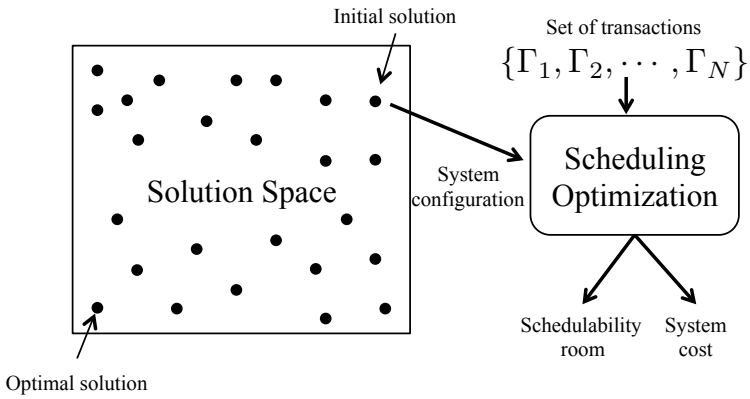


Figure. 12: Solution space search with scheduling optimization

ity to accept more transactions after accepting the given transactions and (2) system cost that is the sum of every HW's cost that constitute the system configuration. Then, our problem is to search the solution space using the schedulability room and system cost to find the optimal solution.

Our system configuration problem can be formally described as follows:

Problem Description: For a given set of control transactions $\{\Gamma_1, \Gamma_2, \cdots, \Gamma_N\}$, the problem is to find the optimal system configuration with the minimal system cost. By the system configuration, we specifically mean

- the set of required ECUs $\{ECU_1, ECU_2, \cdots, ECU_M\}$,
- the set of required CANs $\{CAN_1, CAN_2, \cdots, CAN_L\}$, and
- the mappings from SW components to ECUs/CANs.

The constraint is that the given set of control transactions should be acceptable with the system configuration under our SW component scheduling

technique presented in Chapter 4. For each SW component C_{ik}, we assume that it can be placed on a specific type of HW resource. For example, a vision component should be placed on an ECU with a camera sensor. Thus, for each C_{ik}, it is associated with a specific resource type. The cost of a resource is also determined by the resource type it belongs to.

5.3 Heuristic Iterative Search

In order to make a practical solution to solve such design space exploration problem, we propose to use a heuristic algorithm that finds a near-optimal solution in a reasonable time bound. First, let us define the following notations for a specific system configuration SC, that is, a set of ECUs, CANs, and SW/HW mappings:

- $SystemCost(SC)$: the total manufacturing cost for ECUs and CANs that constitute the system configuration SC.
- $SchedulabilityRoom(SC)$: the minimum margin between the utilization bound and the peak utilization of each HW resource after conducting the schedule optimization process explained in Chapter 4 assuming the system configuration SC. This can be thought as a pessimistic estimation of SC's remaining HW capacity to accept more control transactions.

Based on the above notations, the basic idea is to trade the schedulability room to reduce the system cost. To do so, we start by the system configuration SC_0 as shown in Figure 13(a) where every SW component

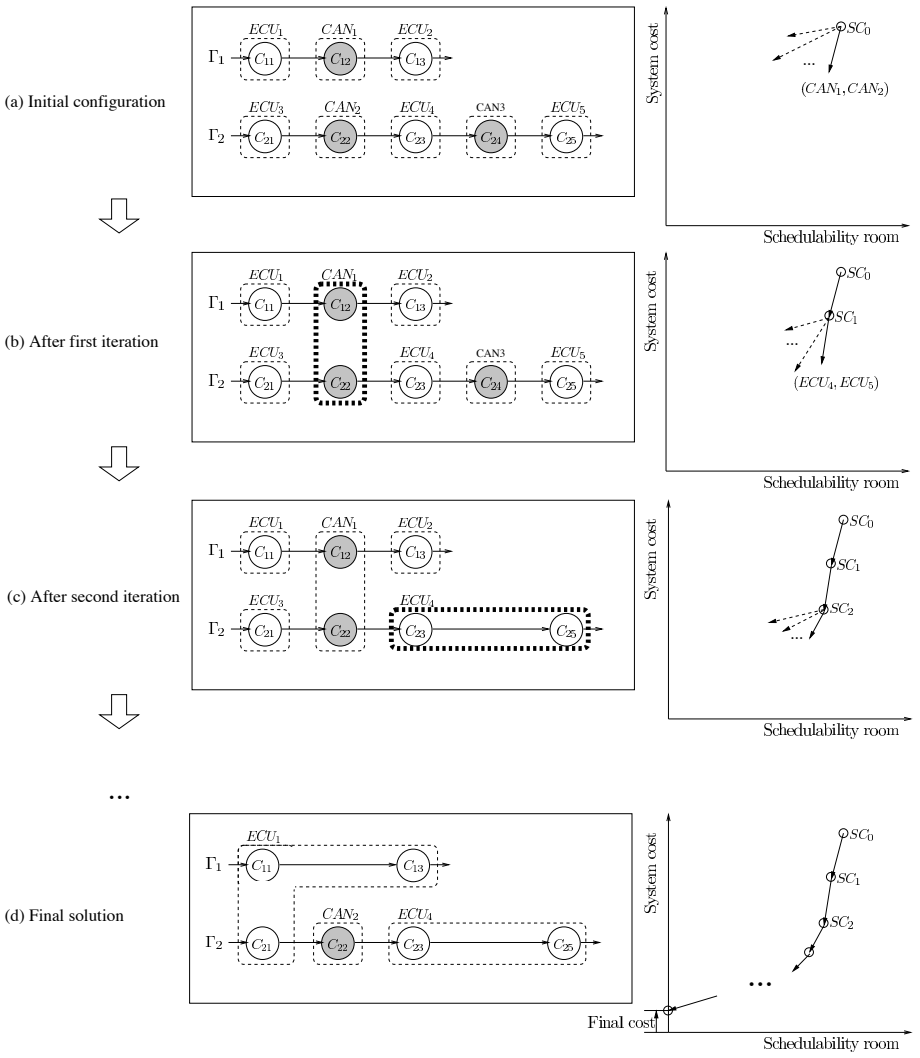


Figure. 13: Heuristic iterative search

is provided with a dedicated HW resource. In the example, we have two transactions $\{\Gamma_1, \Gamma_2\}$. Γ_1 has three SW components where C_{12} is a CAN message since C_{11} and C_{13} are on different ECUs. Γ_2 has five SW components where two of them are CAN messages. Please note that the SW components representing CAN messages are distinguished as gray circles in the figure. At SC_0, the system cost is the highest in the entire solution space. However, note that this worst solution SC_0 also has the maximum schedulability room as shown in Figure 13(a). Then, the next step is to merge a pair of HW resources, that is, moving SW components in one HW resource to the other HW resource and discard the former. Note that following the resource type constraint, not every pair of HW resources are mergeable. Only HW resources with the same resource type can be merged. After this merge process, the system cost is reduced and at the same time the schedulability room also shrinks. The arrows going out of SC_0 depict the possible pairs to merge where the destination of each arrow represents the resulting system cost and schedulability room after merging each candidate pair. Generally, at each SC_i, there are a number of choices to pick a pair of HW resources to merge. Among them, we pick SC_{i+1} with the sharpest

$$\frac{\Delta SystemCost(SC_i)}{\Delta SchedulabilityRoom(SC_i)} =$$

$$\frac{|SystemCost(SC_{i+1}) - SystemCost(SC_i)}{|SchedulabilityRoom(SC_{i+1}) - SchedulabilityRoom(SC_i)|}$$

. The intuition behind this is that we try to trade the schedulability room to reduce the system cost at the best price-performance ratio. By iteratively

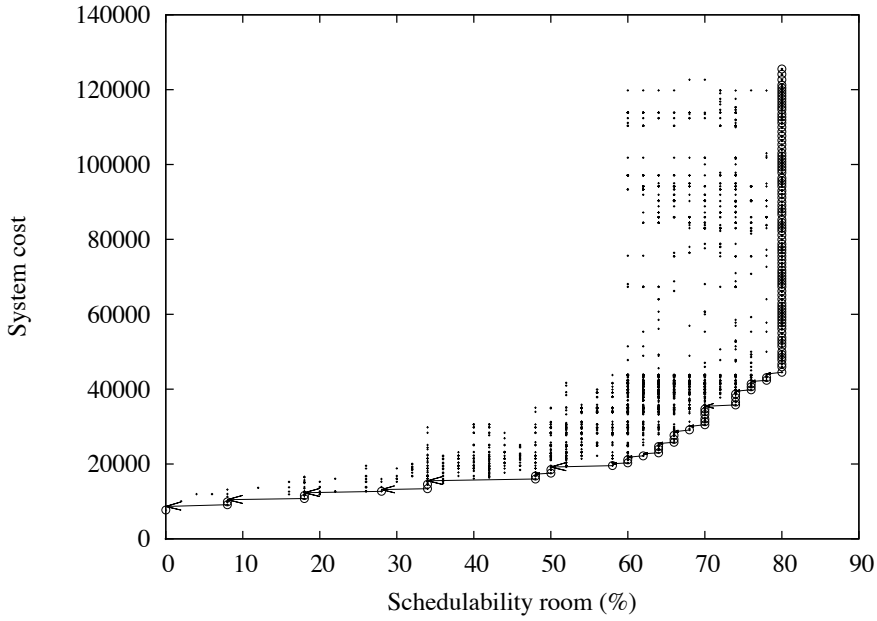


Figure. 14: System configuration optimization process

merging HW resources (or trading the schedulability room) as shown in Figure 13(b)(c) until there remains no schedulability room to trade, the solution with the minimum system cost can be found as in Figure 13(d).

5.4 Experiment

In order to validate our heuristic algorithm for the system configuration problem, we first generate 32 synthesized control transactions with 142 SW components. We assume six types of HW resources including five different types of ECUs and a single type of CAN bus. For each type of HW resource, the relative cost value is uniquely assigned from 100 to 500. Figure 14 shows

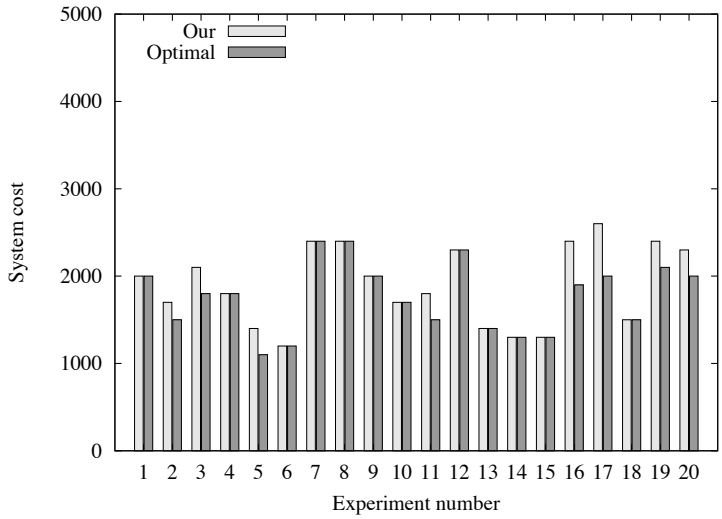


Figure. 15: Comparison with the optimal results

the optimization process. The x-axis represents the schedulability room and the y-axis represents the system cost for each system configuration. Circles in the figure represent the iteratively chosen system configurations during the optimization process, while the points are the candidate solutions considered during the optimization process but discarded. Starting from the upper right corner, our heuristic algorithm tries to reduce the system cost as much as possible while at the same time keeping the schedulability room also as much as possible. As the result of the optimization process, the final system cost is about 7700. Note that the vertical slope in the early phase of the optimization process is caused by the fact the in the early phase there are still too many ECUs and CANs so that eliminating ECUs and CANs does not cause any visible reduction of the schedulability room. It takes 24 minutes to find the solution on a 2.66GHz Intel Core i7 system, which is a

reasonable time bound for finding the optimal system configuration of such complex workloads.

We also check the optimality of our heuristic algorithm by comparing our results with the optimal results as shown in Figure 15. Note that when the number of control transactions gets larger, it becomes impossible to find the optimal solution, thus, we generate only four control transactions for each experiment. Out of such 20 iterations, 12 cases successfully find the optimal result. The average over-estimation is 8% compared to the found optimal results. Thus, we can conclude that our heuristic algorithm can practically find a near-optimal solution in a reasonable time bound even for complex systems.

For the above experiments, we generate transactions with their periods harmonic, thus the hyperperiod is always equal to the longest period. In the remainder of this section, we validate our framework with transactions with non-harmonic periods. For the comparison, we compare two approaches in the following:

- When calculating the maximum peak utilization in the scheduling optimization process, all the transactions are considered at once, thus investigating the entire hyperperiod. This approach is denoted by *No harmonic grouping.*

- When calculating the maximum peak utilization, the given transactions are grouped into a number of harmonic groups. Then, the maximum peak utilization is pessimistically estimated by adding the peaks of every harmonic group. This approach is denoted by *Harmonic*

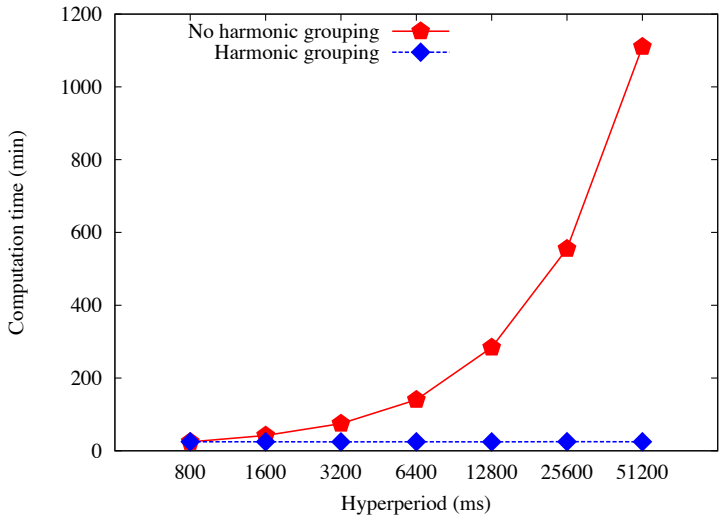


Figure. 16: Computation times with varying hyperperiod

grouping.

Figure 16 compares the two approaches in terms of the computation time for the system optimization as varying the hyperperiod from 800 ms to 51200 ms. In the graph, we can observe that the computation time grows according to the hyperperiod by *No harmonic grouping*. When the hyperperiod is 51200 ms, the computation time is almost 20 hours. However, by *Harmonic grouping*, the computation times are almost constant. In order to measure the estimation overhead of *Harmonic grouping*, we compares the two approaches in terms of the overestimated system cost of the system optimization. Figure 17 shows the normalized resulting system cost for both approaches as varying the number of harmonic groups with 100 randomly generated transaction sets with 32 control transactions. As shown in

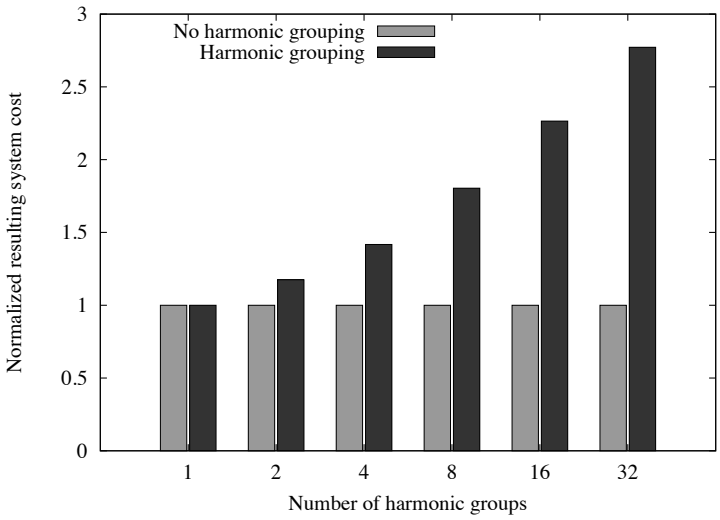


Figure. 17: Estimation overhead of harmonic grouping

the graph, as the number of harmonic groups increases, the estimation overhead also grows. When the number of harmonic groups is 2, the estimation overhead is about 17%. The estimation overhead becomes 170% when the number of harmonic groups is 32, that is, every transaction is non-harmonic. However, since most automotive workloads are composed of at most two or three harmonic groups, the estimation overhead is not significant in that case.

5.5 Summary

In this chapter, we presents the system configuration method that minimizes the system cost with the constraint that the system configuration should accept the given control transactions still guaranteeing each SW

component's invariant delay under the mechanism in Chapter 4. In order to find the near-optimal solution in a reasonable time bound, we propose a heuristic algorithm that searches the design space efficiently guided by a metric $SchedulabilityRoom(SC)$ that approximates a system configuration SC's remaining capacity to accept more control transactions. Under our proposed algorithm, it is shown that we can find the near-optimal solution within a reasonable time bound for a realistic workload.

Chapter 6

End-to-end Toolchain and Component Execution Kernel

In this chapter, we demonstrate our end-to-end toolchain and component execution kernel to show the usability of our framework.

Our end-to-end toolchain and component exeuction kernel consist of the following components:

- *Control transaction designer.* Control engineers can design complex control transactions using our control transaction designer by composing SW component's function and delay in a graphical user environment.

- *System configuration and schedule optimizer.* For the given control transactions, our system configuration and schedule optimizer finds the optimal system configuration and schedule following our methods in Chapter 4 and 5.

- *Real-time simulator.* The entire system configuration is validated by simulating the functional and temporal behaviors of SW components assuming the found optimal system configuration and schedule. Our real-time simulator is useful since we can validate the system in the early stage of development where actual ECU hardwares are not ready

yet.

- *ECU code generator.* After validating the system through real-time simulation, our ECU code generator actually generates the source codes for the target ECUs. When generating source codes for each ECU, it automatically generates source codes for SW components allocated to that ECU in the system optimization. Also, the optimized schedule timetable is generated during the source code generation.

- *Component execution kernel.* Our component execution kernel actually implements the active window based server scheduling. According to the schedule timetable, it creates servers for each SW component and schedules them under EDF scheduling algorithm.

For the remainder of this chapter, we demonstrate each component of our end-to-end toolchain and component execution kernel for the purpose of validating the usability of our framework.

6.1 Control Transaction Designer

Figure 18 shows an example screenshot of our control transaction designer. Control engineers compose control transactions from sensing to actuation through connecting input and output ports of SW components. In the figure, each rounded rectangle represents a SW component where each SW component has a number of input ports and output ports represented by squared rectangles. Between input ports and output ports, narrow lines represent the messages between SW components. The car under the SW com-

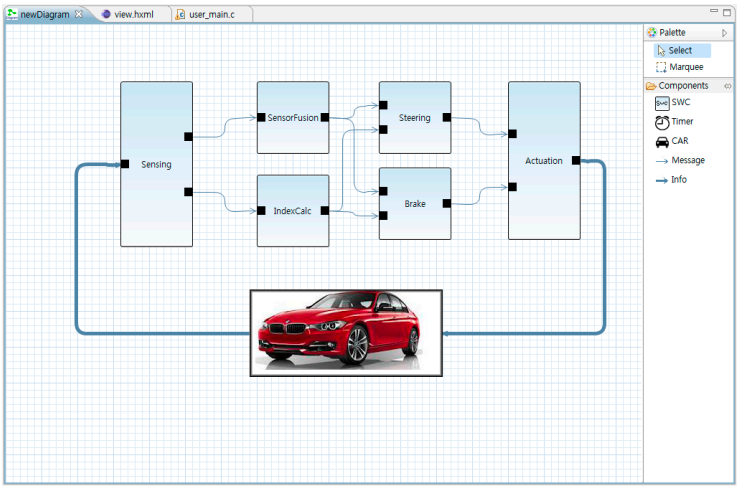


Figure. 18: An example screenshot of our control transaction designer

ponents depicts the physical sensors and actuators attached to the vehicle. The thick lines are sensing and actuation signals between SW components and sensors/actuators. Control engineers can design a complex control algorithm by composing already-existing blocks (i.e., SW components) as well as making a new SW component and editing the algorithm inside the block. Note that, at this stage, control engineers do not care about the system configuration upon which the control transactions will be implemented.

6.2 System Configuration and Schedule Optimizer

Figure 19 shows an example screenshot of our system configuration and schedule optimizer. The figure specifically shows the resulting optimal

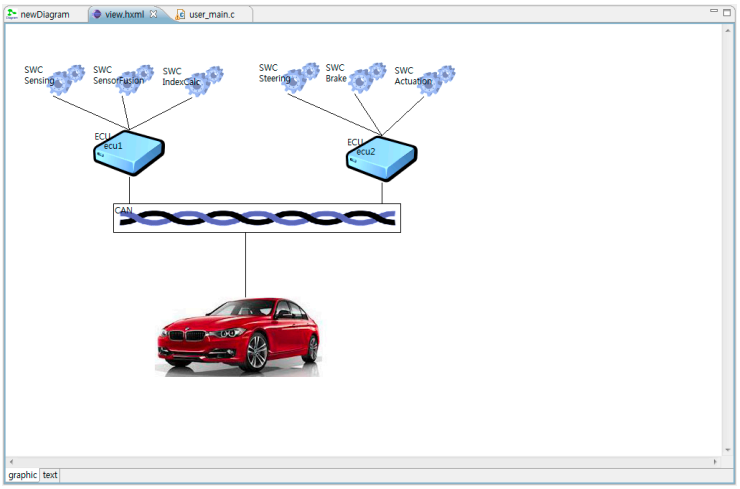


Figure. 19: An example screenshot of our system configuration and schedule optimizer showing the resulting system configuration

system configuration of the control algorithms given in Figure 18. As a result, it says that we need two ECUs connected by a single CAN bus. It also shows the SW/HW mapping. Additionally, the optimized schedule timetable is also generated by our system configuration and schedule optimizer. This tool is the actual implementation of our schedule optimziation and system optimization methods in Chapter 4 and 5.

6.3 Real-Time Simulator

Figure 20 shows an example screenshot of our PC-based real-time simulator. Our real-time simulator actually executes the control algorithms considering the given system configuration and schedule and mimics the exact timing behavior of each SW component running in the ECUs. The left-hand

Figure. 20: An example screenshot of our real-time simulator

side of the figure shows the output information of our real-time simulator. It gives the internal scheduling diagram as well as the real-time plotting of internal variables of the control algorithms. The right-hand side shows the vehicle dynamics simulator, which is connected via CAN bus to the PC running our real-time simulator. As the vehicle dynamics simulator, we use CarSim [80] which is the most popular commercial vehicle dynamics simulator in the automotive industry.

6.4 ECU Code Generator

We implement our ECU code generator for the purpose of generating source codes to implement SW components ready to run on our component execution kernel. Thus, our ECU code generator generates source

codes using our kernel APIs. Please note that the generated source codes are not portable to other operating systems. Generating platform-independent source codes is out of this dissertation's scope. The generated source codes are compiled and linked with our component execution kernel to create the binary image file for each ECU.

6.5 Component Execution Kernel

In order to validate the invariant delay property in terms of the actual control performance, we actually implement an automotive testbed as in Figure 21. As a target platform, we use an ECU board equipped with a TriCore TC1797 32-bit microprocessor. On top of it. we implement our component execution kernel including the active window based server scheduling module, delayed-output module, and CAN driver module. We use CarSim as the plant.

As a target application, we implement the LKAS which automatically keeps the vehicle following the lane without the driver's intervention. To implement the LKAS control transaction, we decompose the entire control transaction into three sequential SW components, i.e., sensing, decision, and actuation SW components. The period for the control transaction is 100 ms and the end-to-end deadline is equal to the period. For the simplicity, the end-to-end deadline is unformly decomposed into local delay bounds. Besides the LKAS control transaction, we also add three dummy SW components to generate overload. For the comparison purpose, we run the application with the existing fixed-priority scheduling kernel and our component

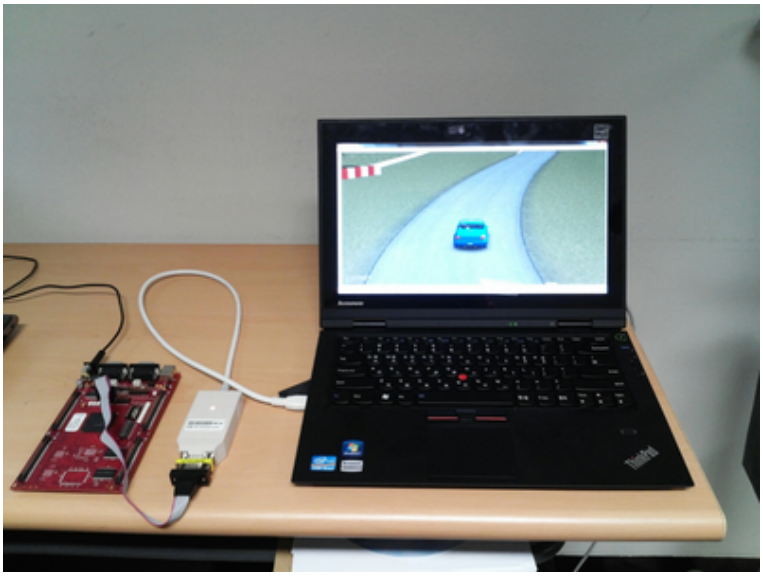
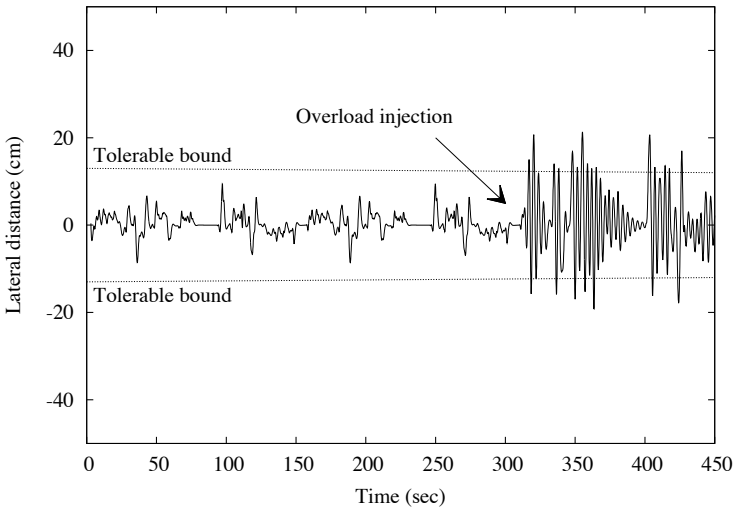
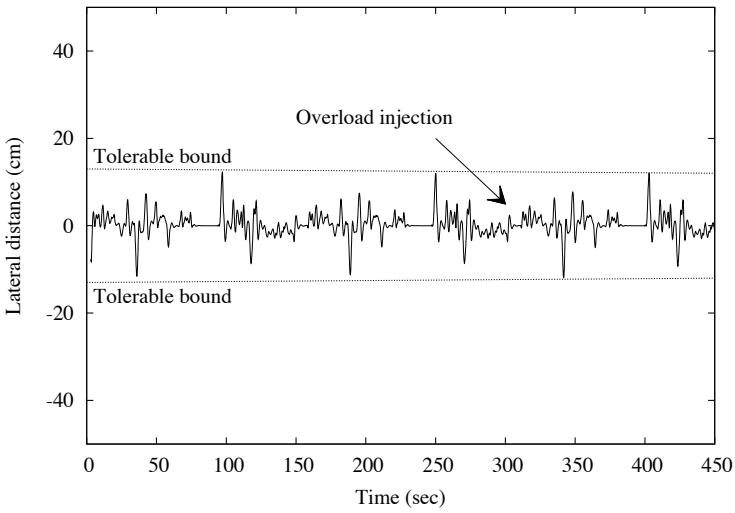


Figure. 21: Automotive testbed system

execution kernel. Figure 22(a) plots the fixed-priority scheduler's actually measured control performance, that is, the lateral distance between the middle of the lane and the middle of the vehicle. As you can see in Figure 22(a), from 0 sec to 300 sec, the control performance is good enough. However, after injecting overload at time 300 sec by increasing the execution times of the three dummy SW components, the control performance is immediately degraded. This performance degradation is due to the unexpected delay caused by the overloaded dummy SW components. Eventually, it goes out of the tolerable bound area. Figure 22(b) shows the control performance of our component execution kernel with the same workload. Note that in the normal situation, it shows a slightly lower control performance due to the delayed-output semantic of our kernel compared to the fixed-priority



(a) Fixed-priority scheduler without temporal isolation



(b) Our control kernel with temporal isolation

Figure. 22: Implementation results

scheduler which emits actuation commands as early as possible. However, even after injecting the overload, our kernel shows the same control performance as if nothing happens, which shows the benefit of our mechanism in the control performance perspective by guaranteeing invariant delay to each SW component.

Chapter 7

Conclusion

Targeting for complex automotive software systems, this dissertation proposes a novel framework for realizing automotive control transactions on networked ECUs guaranteeing invariant delay property for each SW component. Under our framework, control engineers design their control transactions by composing SW components with its function and also its invariant delay, thus each SW component becomes a complete building block not only in the functional perspective but also in the temporal perspective. Then, our framework systematically realizes the given control transactions on networked ECUs in the following ways:

- For given control transactions and given system configuration, our schedule optimization method finds the optimal SW component scheduling that guarantees component's delay property with the maximum resource efficiency by our active window based server scheduling.
- For given control transactions with no given system configuration, our system optimization method finds the minimal cost system configuration with a feasible SW component scheduling by our heuristic iterative search algorithm.

Based on the above methods, we also implement our toolchain and component execution kernel to demonstrate the usability of our framework.

Using our toolchain, control engineers can easily design complex control transactions with a component-based design method. Our toolchain then automatically finds the optimal system configuration and schedule. After validating the system design through our real-time simulator that precisely mimics the functional and temporal behavior of the control transactions in a PC environment, source codes for ECU are automatically generated and downloaded to ECUs with our component execution kernel that actually implements the active window based server scheduling.

7.1 Outstanding Issues

Although our proposed framework provides a promising solution for the design and implementation of complex automotive control transactions, the following issues should be addressed to practically use our framework in the automotive systems:

- *Handling aperiodic workloads.* We assume that all the control transactions are periodic. The automotive system however also has interrupt-like workloads such as firing the ignition when a crankshaft reaches a specific position. In order to handle such workloads with a immediate response requirement, our framework also should account for the aperiodic workloads besides the periodic workloads.

- *Non-preemptible code sections.* We assume that the workloads are fully preemptible. In a practical sense, however, there are non-preemptible code sections like HW register accessing and operating system services. For our framework to be used practically, our schedule opti-

mization framework should handle such issues.

- *Workload changes.* Since our framework is based on offline optimization process, workload changes may lead to another offline optimization steps. In case that the workload change is not significant, there needs a way to dynamically adapt the system to workload change.

- *Mode changes.* Automotive systems have multiple operating modes. For instance, the software for a engine controller has several modes like (1) from power on to engine start, (2) after engine start to driving, (3) normal driving, and (4) after engine stop. Different jobs should be performed according to the mode. Thus, we need a mode change protocol to prepare for multi mode systems.

7.2 Future Work

In this section, we identify the future research directions our proposed framework is aiming at as follows:

- *Intermediate delay assignment.* Although having invariant delay for each SW component greatly simplifies the contract between control engineers and software engineers, it may hurt the schedulability compared to the case where only each transaction's end-to-end delay is given as a constraint. In such case, the intermediate delays can be chosen freely on condition that they don't break the end-to-end delay constraint. With this motivation, we plan to extend our algorithm such that it can find the optimal intermediate delay assignment also.

- *Supporting parallel workloads.* Although manycore processors and general-purpose computing on graphics processing units (GPGPU) are not commonly used in the automotive industry now, they have an enormous amount of opportunity to accelerate the performance of control systems by parallelizing workloads. Our framework however assumes that each SW component does not have any parallel workload. Thus, we are planning to extend our framework such that it can support SW components with parallel workload by efficiently utilizing the underlying manycore processors and GPGPUs.

References

[1] I. Shin and I. Lee, "Periodic Resource Model for Compositional Real-Time Guarantees," in *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS)*, pp. 2–13, Dec. 2003.

[2] M. Broy, "Challenges in Automotive Software Engineering," in *Proceedings of the 28th International Conference on Software Engineering (ICSE)*, pp. 33–42, 2006.

[3] 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*, vol. 8, no. 5, pp. 284–292, 1993.

[4] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming*, vol. 40, pp. 117–134, April 1994.

[5] P. Derler, E. Lee, M. Törngren, and S. Tripakis, "Cyber-Physical System Design Contracts," in *Proceedings of the ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, April 2013.

[6] Simulink, MathWorks, Inc. `http://www.mathworks.com/products/simulink/`.

[7] Ascet, ETAS. `http://www.etas.com/en/products/ascet_software_products.php`.

[8] LabView, National Instruments Corporation. `http://www.ni.com/labview`.

[9] AutoBox, dSPACE GmbH. `https://www.dspace.com/en/ltd/home/products/hw/accessories/autobox.cfm`.

[10] N. Navet, A. Monot, B. Bavoux, and F. Simonot-Lion, "Multi-source and multicore automotive ECUs - OS protection mechanisms and scheduling," in *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE)*, pp. 3734–3741, Jul. 2010.

[11] H. Kopetz, "On the Design of Distributed Time-Triggered Embedded Systems," *Journal of Computing Science and Engineering*, vol. 2, pp. 340–356, Dec. 2008.

[12] AUTOSAR, AUTOSAR Consortium. `http://www.autosar.org`.

[13] OSEK. `http://www.osek-vdx.org`.

[14] M. Monchiero, R. Canal, and A. González, "Design space exploration for multicore architectures: a power/performance/thermal view," in *Proceedings of the 20th annual international conference on Supercomputing*, pp. 177–186, 2006.

[15] P. Chaparro, J. Gonzalez, G. Magklis, C. Qiong, and A. Gonzalez, "Understanding the Thermal Implications of Multi-Core Architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 8, pp. 1055–1065, 2007.

[16] Aurix, Infineon Technologies AG. `http://www.infineon.com/aurix`.

[17] Qorivva, Freescale Semiconductor, Inc. `http://www.freescale.com/qorivva`.

[18] SPEAr, STMicroelectronics. `http://www.st.com/web/catalog/mmc/FM169/SC1156`.

[19] M. Farsi, K. Ratcliff, and M. Barbosa, "An overview of Controller Area Network," *Computing & Control Engineering Journal*, vol. 10, no. 3, pp. 113–120, 1999.

[20] K. Tindell, A. Burns, and A. J. Wellings, "Calculating Controller Area Network (Can) Message Response Times," *Control Engineering Practice*, vol. 3, pp. 1163–1169, Aug. 1995.

[21] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised," *Real-Time Systems Journal*, vol. 35, pp. 239–272, April 2007.

[22] B. Andersson and E. Tovar, "The Utilization Bound of Non-Preemptive Rate-Monotonic Scheduling in Controller Area Networks is 25%," in *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pp. 11–18, July 2009.

[23] K. M. Zuberi and K. G. Shin, "Non-Preemptive Scheduling of Messages on Controller Area Network for Real-Time Control Applications," in *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 240–249, May 1995.

[24] M. D. Natale, "Scheduling the CAN Bus with Earliest Deadline Techniques," in *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pp. 259–268, Nov. 2000.

[25] P. Pedreiras and L. Almeida, "EDF message scheduling on controller area network," *Computing & Control Engineering Journal*, vol. 13, pp. 163–170, Aug. 2002.

[26] T. Führer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel, and M. Walther, "Time triggered communication on CAN (Time Triggered CAN-TTCAN)," in *Proceedings of the 7th International CAN Conference*, 2000.

[27] L. Almeida, J. A. Fonseca, and P. Fonseca, "Flexible Time-Triggered Communication on a Controller Area Network," in *Proceedings of the Work-In-Progress Session of the IEEE Real-Time Systems Symposium (RTSS)*, Dec. 1998.

[28] L. Almeida, J. A. Fonseca, and P. Fonseca, "A Flexible Time-Triggered Communication System Based on the Controller Area Network: Experimental Results," in *Proceedings of the Fieldbus Technology International Conference*, Sep. 1999.

[29] T. Nolte, M. Sjodin, and H. Hansson, "Server-Based Scheduling of the CAN Bus," in *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 169–176, Sep. 2003.

[30] C. Hewitt, P. Bishop, and R. Steiger, "A universal modular ACTOR formalism for artificial intelligence," in *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, IJCAI'73, pp. 235–245, 1973.

[31] S. Tripakis, C. Sofronis, P. Caspi, and A. Curic, "Translating discrete-time Simulink to Lustre," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 4, no. 4, pp. 779–818, 2005.

[32] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. De Simone, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64–83, 2003.

[33] R. Arthan, P. Caseley, C. O'Halloran, and A. Smith, "ClawZ: Control laws in Z," in *Proceedings of the 3rd IEEE International Conference on Formal Engineering Methods (ICFEM)*, pp. 169–176, 2000.

[34] M. D. Natale and V. Pappalardo, "Buffer optimization in multitasking implementations of Simulink models," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, 2008.

[35] A. Benveniste and G. Berry, "The synchronous approach to reactive and real-time systems," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1270–1282, 1991.

[36] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis–the SymTA/S approach," *IEE*

Proceedings-Computers and Digital Techniques, vol. 152, no. 2, pp. 148–166, 2005.

[37] SymTA/S, Symtavision. https://www.symtavision.com/symtas.html.

[38] 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," in *Proceedings of the 3rd IEEE/ACM International Conference on Cyber-Physical Systems (ICCPS)*, pp. 208–208, 2012.

[39] Simulink Coder, MathWorks, Inc. http://www.mathworks.com/products/simulink-coder/.

[40] TargetLink, dSPACE GmbH. http://www.dspace.com/en/inc/home/products/sw/pcgs/targetli.cfm.

[41] S. Wang and K. G. Shin, "Task construction for model-based design of embedded control software," *IEEE Transactions on Software Engineering*, vol. 32, no. 4, pp. 254–264, 2006.

[42] T. Henzinger, B. Horowitz, and C. Kirsch, "Giotto: a time-triggered language for embedded programming," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 84–99, 2003.

[43] J. Liu and E. Lee, "Timed multitasking for real-time embedded software," *IEEE Control Systems*, vol. 23, no. 1, pp. 65–75, 2003.

[44] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[45] S. K. Baruah, L. E. Rosier, and R. R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Systems*, vol. 2, no. 4, pp. 301–324, 1990.

[46] J. K. Strosnider, J. P. Lehoczky, and L. Sha, "The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments," *IEEE Transactions on Computers (TC)*, vol. 44, no. 1, pp. 73–91, 1995.

[47] J. P. Lehoczky, L. Sha, J. K. Strosnider, *et al.*, "Enhanced aperiodic responsiveness in hard real-time environments," in *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pp. 261–270, 1987.

[48] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for hard-real-time systems," *Real-Time Systems*, vol. 1, no. 1, pp. 27–60, 1989.

[49] J. P. Lehoczky and S. Ramos-Thuel, "Scheduling periodic and aperiodic tasks using the slack stealing algorithm," *Advances in Real-Time Systems*, pp. 175–198, 1995.

[50] M. Spuri and G. Buttazzo, "Scheduling aperiodic tasks in dynamic priority systems," *Real-Time Systems*, vol. 10, no. 2, pp. 179–210, 1996.

[51] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS)*, pp. 4–13, 1998.

[52] W. Zheng, Q. Zhu, M. Di Natale, and A. S. Vincentelli, "Definition of task allocation and priority assignment in hard real-time distributed systems," in *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pp. 161–170, 2007.

[53] Q. Zhu, Y. Yang, M. Natale, E. Scholte, and A. Sangiovanni-Vincentelli, "Optimizing the software architecture for extensibility in hard real-time distributed systems," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 621–636, 2010.

[54] Q. Zhu, H. Zeng, W. Zheng, M. D. Natale, and A. Sangiovanni-Vincentelli, "Optimization of task allocation and priority assignment

in hard real-time distributed systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 11, no. 4, p. 85, 2012.

[55] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, "Period optimization for hard real-time distributed automotive systems," in *Proceedings of the 44th Design Automation Conference (DAC)*, pp. 278–283, 2007.

[56] L. Coyle, M. Hinchey, B. Nuseibeh, and J. L. Fiadeiro, "Guest Editors' Introduction: Evolving Critical Systems," *Computer*, pp. 28–33, 2010.

[57] A. Hattendorf, A. Raabe, and A. Knoll, "Shared Memory Protection for Spatial Separation in Multicore Architectures," in *Proceedings of the 7th IEEE International Symposium on Industrial Embedded Systems (SIES). Karlsruhe, Germany*, 2012.

[58] J.-C. Kim, D. Lee, C.-G. Lee, and K. Kim, "RT-PLRU: A New Paging Scheme for Real–Time Execution of Program Codes on NAND Flash Memory for Portable Media Players," *IEEE Transactions on Computers (TC)*, vol. 60, no. 8, pp. 1126–1141, 2011.

[59] D. Lee, J. Kim, C. Lee, and K. Kim, "mRT-PLRU: A General Framework for Real-Time Multi-Task Executions on NAND Flash Memory," *IEEE Transactions on Computers (TC)*, vol. 62, no. 4, pp. 758–771, 2013.

[60] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-core Platforms," in *Proceedings of the 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2013.

[61] C. Ficek, N. Feiertag, and K. Richter, "Applying the AUTOSAR timing protection to build safe and efficient ISO 26262 mixed-criticality systems," in *Proceedings of the ERTS2*, 2012.

[62] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Transactions on Networking (TON)*, vol. 1, no. 3, pp. 344–357, 1993.

[63] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. Gehrke, and C. Plaxton, "A Proportional Share Resource Allocation Algorithm For Real-Time, Time-Shared Systems," in *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS)*, pp. 288–299, Dec. 1996.

[64] C. W. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves: Operating system support for multimedia applications," in *Proceedings of the International Conference on Multimedia Computing and Systems*, pp. 90–99, 1994.

[65] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa, "Resource Kernels: A Resource-Centric Approach to Real-Time and Multimedia Systems," in *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, pp. 150–164, Jan. 1998.

[66] S. S. Craciunas, C. M. Kirsch, H. Payer, H. Rock, and A. Sokolova, "Programmable Temporal Isolation through Variable-Bandwidth Servers," in *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pp. 171–180, 2009.

[67] S. S. Craciunas, C. M. Kirsch, H. Payer, H. Röck, and A. Sokolova, "Temporal isolation in real-time systems: the VBS approach," *International Journal on Software Tools for Technology Transfer*, pp. 1–21, 2012.

[68] G. Lipari and G. Buttazzo, "Scheduling real-time multi-task applications in an open system," in *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pp. 234–241, 1999.

[69] G. Lipari and S. K. Baruah, "Efficient scheduling of real-time multi-task applications in dynamic systems," in *Proceedings of the 6th IEEE*

Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 166–175, 2000.

[70] G. Lipari, J. Carpenter, and S. Baruah, "A framework for achieving inter-application isolation in multiprogrammed, hard real-time environments," in *Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS)*, pp. 217–226, 2000.

[71] P. Kumar, J.-J. Chen, and L. Thiele, "Demand bound Server: Generalized Resource Reservation for Hard Real-Time Systems," in *Proceedings of the 9th ACM International Conference on Embedded Software (EMSOFT)*, pp. 233–242, 2011.

[72] S. Wang, J. R. Merrick, and K. G. Shin, "Component Allocation with Multiple Resource Constraints for Large Embedded Real-Time Software Design," in *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 219–226, 2004.

[73] CPLEX, IBM. `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`.

[74] J.-C. Kim, K.-S. We, C.-G. Lee, K.-J. Lin, and Y. S. Lee, "HW Resource Componentizing for Smooth Migration from Single-function ECU to Multi-function ECU," in *Proceedings of the 27th International Symposium on Applied Computing (SAC)*, 2012.

[75] D. Johnson, A. Demers, D. Ullman, M. Garey, and R. Graham, "Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms," *SIAM Journal of Computing (SICOMP)*, no. 3, pp. 299 – 325, 1974.

[76] D. Johnson, *Near-Optimal Bin Packing Algorithms*. PhD thesis, MIT, 1973.

[77] M. Caccamo, L. Y. Zhang, L. Sha, and G. Buttazzo, "An Implicit Prioritized Access Protocol for Wireless Sensor Networks," in *Proceedings*

of the IEEE Real-Time Systems Symposium (RTSS), pp. 39–48, Dec. 2002.

[78] T. L. Crenshaw, A. Tirumala, S. Hoke, and M. Caccamo, "A Robust Implicit Access Protocol for Real-Time Wireless Collaboration," in *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 177–186, July 2005.

[79] J. W. S. Liu, *Real-Time Systems*, pp. 224–225.

[80] CarSim, Mechanical Simulation Corporation. `http://www.carsim.com`.

Abstract

Component-based Scheduling and System Optimization for Automotive Control Systems

Jong-Chan Kim
School of Computer Science and Engineering
The Graduate School
Seoul National University

Automotive software systems are getting larger and more complex as new advanced features are added. Thus, such complex software systems should be designed by composing less complex SW components to manage the complexity. Although the functional correctness of the entire system can be easily verified by validating each SW component's invariant functional behavior, since each SW component's temporal behavior changes due to the underlying HWs and surrounding SW components, it is difficult to validate the temporal correctness of the resulting system. In order to solve this problem, this dissertation proposes a novel framework that realizes the given control transactions on networked ECUs guaranteeing SW component's invariant input-output delay property. By guaranteeing SW component's invariant delay property, the entire system's temporal correctness can be easily

validated from the constructing SW components' delays. For the component scheduling method, we make use of existing bandwidth reservation mechanisms. However, since traditional bandwidth reservation mechanisms assign a permanent utilization to each SW component for the entire time duration, it wastes away the underlying HW capacity especially when the transaction is given as sequentially executing SW components. With this motivation, we present a new resource provisioning mechanism that provides the HW resource utilization only during each SW component actually needs it. The experimental result shows that our framework can handle three times more control transactions than the traditional bandwidth reservation mechanisms that assign a permanent utilization to each SW component. Based on that, we also propose a system configuration optimization method that finds the minimal cost HW architecture and SW/HW mapping (or system configuration) with a feasible SW component scheduling. The performance of the proposed framework is validated through both simulation and actual implementation. Our implementation study also demonstrates the usability of our approach by realizing our framework as end-to-end toolchain and component execution kernel.

Keywords : Automotive Control Software, Scheduling, System Optimization

Student Number : 2008-30219