Adaptive Dynamic Scheduling on Multifunctional Mixed-Criticality Automotive Cyber-Physical Systems

Guoqi Xie, *Member, IEEE*, Gang Zeng, *Member, IEEE*, Zhetao Li, Renfa Li, *Senior Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—A function model for the description of distributed end-to-end computations is called a task graph. Multiple functions with different criticality levels are supported by one electronic control unit (ECU), and one function is distributed over multiple ECUs in integrated automotive architecture. Considering the inherent heterogeneity, interaction, and diverse nature of such an architecture, automotive embedded systems have evolved to automotive cyber-physical systems (ACPS), which consist of multiple distributed automotive functions with different criticality levels. Efficient scheduling strategies can fully utilize ECUs in ACPS for high performance. However, ACPS should deal with joint challenges of heterogeneity, dynamics, parallelism, safety, and criticality, and these challenges are the key issues that will be solved in the next generation automotive open system architecture adaptive platform. This study first proposes a fairness-based dynamic scheduling algorithm FDS_MIMF to minimize the individual makespans (i.e., schedule lengths) of functions from a high performance perspective. FDS_MIMF can respond autonomously to the joint challenges of heterogeneity, dynamics, and parallelism of ACPS. To further respond autonomously to the joint challenges of heterogeneity, dynamics, parallelism, safety, and criticality of ACPS, we present an adaptive dynamic scheduling algorithm ADS_MIMF to achieve low deadline miss ratios (DMRs) of safety-critical functions from a timing constraint perspective while maintaining the acceptable overall makespan of ACPS from a high performance perspective. ADS_MIMF is implemented by changing up and down the criticality level of ACPS to adjust the execution of different functions on different criticality levels without increasing the time complexity. Experimental results indicate that FDS_MIMF can obtain short overall makespan, whereas ADS_MIMF can reduce the DMR values of high-criticality functions while still keeping satisfactory performance of ACPS.**

***Index Terms*—AUTOSAR adaptive platform, automotive cyber-physical systems (ACPS), functional safety, mixed-criticality, task graph.**

Manuscript received August 4, 2016; revised November 17, 2016, December 29, 2016, and February 8, 2017; accepted February 22, 2017. Date of publication February 23, 2017; date of current version August 11, 2017. This work was supported in part by the National Key Research and Development Plan of China under Grant 2016YFB0200405 and Grant 2012AA01A301-01, in part by the National Natural Science Foundation of China under Grant 61672217, Grant 61173036, Grant 61432005, Grant 61300037, Grant 61300039, Grant 61379115, Grant 61402170, Grant 61370097, Grant 61502162, and Grant 61502405, in part by the CERNET Innovation Project under Grant NGII20161003, and in part by the China Postdoctoral Science Foundation under Grant 2016M592422. The review of this paper was coordinated by Prof. W. Song. *(Corresponding author: Zhetao Li.)*

G. Xie, Z. Li, and R. Li are with the Key Laboratory for Embedded and Network Computing of Hunan Province, College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China (e-mail: xgqman@hnu.edu.cn; liztchina@hotmail.com; liztchina@hotmail.com).

G. Zeng is with the Graduate School of Engineering, Nagoya University, Nagoya 4648603, Japan (e-mail: sogo@ertl.jp).

K. Li is with the Key Laboratory for Embedded and Network Computing of Hunan Province, College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China, and also with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVT.2017.2674302

I. Introduction

A. Background

COST pressure, flexibility, and extensibility, as well as the need to cope with high complexity of functions, are changing the fundamental paradigms of automotive architecture to the integrated architecture, in which software components supplied by multiple sources are integrated in the same hardware platform [1]. Automotive architecture is a type of heterogeneous disturbed architecture, which consists of up to 100 heterogeneous electronic control units (ECUs), sensors, and actuators that communicate over a network of buses [2].

A heterogeneous distributed integrated architecture leads to multi-functional automotive embedded systems, where multiple functions can be supported by one ECU and one function can be distributed over multiple ECUs [1]. Premium cars have up to 70 ECUs, connected to five system busses, realizing over 800 functions [3]. Various functions are realized by a number of distributed tasks which communicate by exchanging messages over the shared buses [2]. A function model for the description of distributed end-to-end computations in automobiles is called a task graph [4], [5]. Given that a distributed automotive function is released by receiving collected data from the sensor and is completed by sending the performing action to the actuator in automotive embedded systems, the task graph is restricted to be directed and acyclic and is called a directed acyclic graph (DAG) [4], [5] where the nodes represent tasks and the edges represent the communication messages between tasks. Examples of active safety functions are brake-by-wire and adaptive cruise control [1]. Furthermore, multiple distributed functions represent multiple DAGs in heterogeneous distributed systems [6]–[8].

Different functions are developed using different design approaches by various levels of auto part suppliers and are deployed together in automotive systems. Distributed automotive functions are classified into three types: active safety, passive safety, and non-safety functions. As a result, the mixed-criticality concept, where certain functions are more significant (critical) than others, has been introduced as a core foundational idea [9]. Criticality is represented by the automotive safety integrity level (ASIL) in automobiles and is defined in the ISO 26262 (Road vehicles - Functional safety), which was formally issued in November 2011 [10].

In automotive embedded systems, the implementation of multiple distributed functions depends on the interaction, feedback, and coordination of multiple ECUs through networks [2]. Moreover, such systems support dynamically released (activated) distributed functions with end-to-end computations that collect and transfer physical world data from 360° sensors to the actuators. Examples of such functions are active cruise control, lane departure warning, and collision avoidance [1], [11]. The inherent heterogeneity, interaction, and diverse nature of integrated automotive architecture require the joint and tight interaction between the cyber (networked computational) and physical worlds [2], [12]. Thus, automotive embedded systems are also typical cyber-physical systems (CPS), and are called automotive cyber-physical systems (ACPS) [2], [13], [14].

B. Motivation

Efficient scheduling strategies are required to fully utilize the numerous ECUs and to achieve substantially high performance improvements. However, scheduling in ACPS faces several new challenges.

First, we know that automotive embedded systems rely on safety checking units and watchdogs to find out whether safety-critical functions are providing correct service or are failed [15], such that these safety-critical functions (e.g., control engine and gearbox) are strictly and periodically released and scheduled. Such periodic activation model is also supported by the automotive open system architecture (AUTOSAR) standard [4]. However, automobiles are required to support an ever increasing number of complex functions to meet higher safety requirement. These functions are distributed, interdependent, and dynamic released by event arrivals [1], [12]. Consider for example, active safety functions X-by-wires (e.g, steering-by-wire and brake-by-wire) use sensors (radars, camera, and ultrasound) to scan the environment surrounding the car. If they detect objects or situations that may endanger the passengers or the pedestrians, then the steering or brake actions should be performed [1]. When a fierce collision happens in the car, the passive safety function airbag will automatically pop up. That is, more safety-critical functions should interact with dynamic physical world and exhibit dynamic behavior. Hence, ACPS integrate both periodically and dynamically released safe-criticality functions together in a common platform. Usual static scheduling for periodic functions should be improved to be applied to dynamic released functions. Furthermore, as emphasized in the preface of [16], the design of CPS requires understanding the joint dynamics of computers, software, networks, and physical processes. Measuring and controlling the dynamics of these processes are the main tasks of CPS. In CPS, many things occur at once; physical processes are compositions of many parallel processes; parallelism (concurrence) is intrinsic in CPS. In addition to dealing with temporal dynamics, CPS designs invariably face challenging parallel issues [16]. In particular, dynamics and parallelism are also the inherent properties of ACPS, which should address these two properties at runtime in response to changes in environments or within themselves [17].

Second, as mentioned in the introduction of ISO 26262 [10]: "safety is one of the key issues of future automobile development. New functions are not only in the area of driver assistance but also in vehicle dynamics control and active and passive safety systems." Safety analysis is important when designing and developing cyber-physical systems (CPS) [18]. Given that automotive embedded systems provide safety-relevant functions, such systems must preserve the predefined requirements to guarantee the correct behavior at all times [17]. Hard deadline constraints are the core predefined timing constraints of automotive functions, but ACPS cannot meet the deadlines of all functions, particularly in large-scale ACPS. A high-criticality function (i.e., a function with high criticality level) has a considerably important and strict timing constraint for a given deadline. Missing the deadlines of high-criticality functions would result in fatal injuries to people. Therefore, the safety of these high-criticality functions must be guaranteed, that is, their potential safety risk should be controlled within an acceptable range.

To support dynamic scheduling and communication in automotive systems, the next generation AUTOSAR standard called AUTOSAR adaptive platform will be formally released in 2017 [19], [20]. The AUTOSAR spokesperson Simon Fürst pointed out that AUTOSAR adaptive platform will support "planned dynamics", which includes dynamic deployment of software components, planning of dynamic scheduling and communication. Meanwhile, the previous AUTOSAR standard has been renamed as the AUTOSAR classic platform. Currently, the AUTOSAR classic platform 1.1 is being accepted testing. Moreover, the AUTOSAR adaptive platform will also be an integrated heterogeneous platform that integrates parallel computing, real-time requirements, safety, criticality, and existing functions into a heterogeneous architecture [21], [22]. Therefore, the AUTOSAR adaptive platform actually reflects the current challenges of heterogeneity, dynamics, parallelism, safety, and criticality of ACPS and theses challenges should be addressed with an adaptive approach. Adaptive scheduling approach should be realized by adapting the architecture of ACPS at runtime to respond autonomously to changes in environments or within themselves.

C. Our Contributions

The main contributions of this study are as follows.

1) We use the task graph of DAG to represent distributed automotive functions and construct ACPS model from dynamics, parallelism, safety, and criticality perspectives according to the characteristics of ACPS.

2) We present the fairness-based dynamic scheduling algorithm FDS_MIMF on multiple-functional mixed-criticality ACPS. FDS_MIM aims to minimize individual makespans

(schedule lengths) of functions with short overall makespan of ACPS from a high performance perspective, and thereby can respond autonomously to the joint challenges of heterogeneity, dynamics, and parallelism of ACPS.

3) We present the adaptive dynamic scheduling algorithm ADS_MIMF on multiple-functional mixed-criticality ACPS. ADS_MIMF aims to achieve the low deadline miss ratios (DMRs) [23] of functions from a timing constraint perspective, whereas obtain a satisfactory overall makespan of ACPS from a high performance perspective, and thereby further respond autonomously to the joint challenges of heterogeneity, dynamics, parallelism, safety, and criticality of ACPS.

The rest of this study is organized as follows. Section II reviews the related literature. Section III constructs related models for ACPS. Section IV proposes the fairness-based dynamic scheduling approach. Section V proposes the adaptive dynamic scheduling approach. Section VI verifies the performance of all our proposed algorithms. Section VII concludes this study.

II. Related Work

Adaptive behavior is common in ACPS (e.g., engine control functions). Analysis, design, and scheduling based on the adaptive variable-rate (AVR) task model with variable WCETs, period, and deadlines, were studied recently in [24]–[26]. The mixed-criticality scheduling problem was first identified and formalized by Vestal [27], whose work has been extended and has inspired substantial investigations of mixed-criticality cyber-physical systems [28], [29]. The models of these studies are based on task model. In other words, these studies considered mixed-criticality from a "task level" perspective. The mixed-criticality scheduling for DAG-based tasks byusing federated scheduling was studied recently [30], [31]. The federated scheduling approach means assigning dedicated processors to high-utilization tasks and schedule them using a work-conserving scheduler. The main difference between [30], [31] and this study is that the former is about static single DAG-based mixed-criticality scheduling, whereas this study is about dynamic multiple DAGs-based mixed-criticality scheduling. Considering that this study is to investigate the dynamic multiple-functional scheduling where functions with individual criticality levels, we mainly review static multiple-functional (i.e., multiple DAGs-based) scheduling, and then review dynamic multiple-functional scheduling.

Static single-functional scheduling provides a research basis for multi-functional scheduling. The problem of scheduling tasks on multiprocessors is known to be NP-hard [32], and scheduling tasks for minimum makespan of a DAG-based function in heterogeneous parallel and distributed systems is a well-known NP-hard optimization problem [33]–[36]. Many heuristic list scheduling algorithms have been proposed to generate near-optimal solutions of single-functional scheduling [33]–[36]. The core idea of single-functional list scheduling includes two phases: the first phase involves ordering all tasks of the function in a list, and the second phase assigns each task to a proper processor (i.e., an ECU in this study) [33]. Multi-functional static scheduling means that multiple functions arrive at the same time instant. A composition approach to merge multiple distributed functions into one function for scheduling first proposed in [37]. Zhao and Sakellariou [38] first indicated the fairness issue in multi-functional scheduling; they proposed a fairness scheduling algorithm called Fairness with a slowdown-driven strategy by ensuring the fairness of different functions. In distributed embedded systems, some works [39], [40] studied the real-time scheduling of distributed functions. They commonly assume that a distributed function is periodic in terms of released time and deadlines. Hu *et al.* [41] and [42] investigated the scheduling of multiple periodic distributed functions for safety-critical time-triggered avionic and automotive systems, respectively. Tamas *et al.* [43], [44] proposed a series of investigations about multi-functional mixed-criticality design and optimization. In [45], we studied the high performance scheduling of multiple simultaneously released functions with different criticality levels on automotive embedded systems. The main limitations of the above works are that all distributed functions are released periodically and cannot be applied to dynamic ACPS, where any function can arrive at any time instant. Dynamics is an inherent property of ACPS and should be dealt with as mentioned in Introduction [16].

The aforementioned multi-functional static scheduling cannot deal with the dynamics of ACPS, where multiple functions may operate at different time instants. Similar to multi-functional static scheduling, achieving high fairness is still an effective approach to minimize the overall makespan of the system in multi-functional dynamic scheduling. The first-come first-served and serve-on-time strategies cannot achieve effective fairness. Yu *et al.* [6] proposed a multi-functional dynamic scheduling algorithm by using a planner-guided strategy. Hsu *et al.* [7] proposed a multi-functional dynamic scheduling called online function management (OWM). The main contribution of OWM is that it considers whether the selected processor is free. If the selected processor is free at that time, the OWM algorithm assigns the selected task to that processor; otherwise, the algorithm keeps the selected task in the common ready list to be scheduled later. Arabnejad *et al.* [8] proposed a new multi-functional dynamic scheduling algorithm called fairness dynamic workflow scheduling (FDWS). If the selected processor is not free at that time, FDWS employs a waiting queue for each processor and places the selected task into the waiting queue for scheduling when the processor is not idle. Therefore, FDWS is different from OWM, which keeps the selected task in the common ready list.

The main problems of the preceding investigations are that they merely minimize the overall makespan of the system [7] or individual makespans of functions [8] from a high performance perspective but ignore the safety and criticality from a timing constraint perspective. Safety is one of the key issues and each function has different criticality levels in ACPS. In this study, we first present the improved high performance scheduling approach and then present an adaptive scheduling algorithm to solve the above joint challenges of dynamics, parallelism, safety, and criticality. We aim to achieve satisfactory performance of the system and significantly reduce the DMRs for high-criticality functions by using the adaptive scheduling algorithm.

TABLE I
IMPORTANT NOTATIONS AND THEIR DEFINITIONS USED IN THIS STUDY

Notation	Definition
$\|X\|$	Size of the set X
$F_m.n_i$	Task n_i of the function G_m
$F_m.c_{i,j}$	WCRT between the tasks $F_m.n_i$ and $F_m.n_j$
$F_m.w_{i,k}$	WCET of the task $F_m.n_i$ on the processor u_k
$rank_u(F_m.n_i)$	Upward rank value of the task n_i
$MS.criticality$	Criticality level of ACPS MS
$MS.makespan$	Overall makespan of ACPS MS
$F_m.F_m.arrivaltime$	Arrival time of the function F_m
$F_m.criticality$	Criticality level of the function F_m
$F_m.lowerbound$	Lower bound of the function F_m
$F_m.deadline$	Relative deadline of the function F_m
$F_m.abs\_deadline$	Absolute deadline of the function F_m
$F_m.makespan$	Makespan of the function F_m
$abs\_deadline(F_m.n_i)$	Absolute deadline of the task $F_m.n_i$
$EFT(F_m.n_i, p_k)$	Earliest finish time the task $F_m.n_i$ on p_k
$DMR(S_x)$	DMR of the function with criticality level S_x
$F_m.task\_priority\_queue$	Task priority queue of the function F_m
$MS.common\_ready\_queue$	Common ready queue of ACPS MS
$p_k.task\_allocation\_queue$	Task allocation queue of the processor p_k

Table I lists important notations and their definitions used in this study.

III. MODELS

A. Architecture

In automotive embedded systems, the scheduling strategies can also be time-triggered (e.g., the static segment of FlexRay) or event-triggered (e.g., the dynamic segment of FlexRay, or controller area network (CAN)) [46]–[48]. Currently, CAN is the most widespread networking standard in automotive industries. CAN is ideally suited for dynamic real-time distributed systems because of its event-triggered, non-destructive, and strictly deterministic medium arbitration [49]. In this study, we consider an integrated automotive electrical and electronic (E/E) architecture as a CAN cluster (also called multi-domain CAN systems) where more than four or five CAN buses are integrated by a central gateway and several ECUs are mounted on each CAN bus [50]–[52]. Considering that physical processes are compositions of many parallel processes, we use the same configuration as [17] that some ECUs connect to several sensors and other ECUs connect to several actuators, as shown in Fig. 1. Such similar automotive E/E architecture can also be found in some ACPS design [2], [12] and is basically similar to the hardware requirement (i.e., sensor and actuators are redundant or accessible via network) of 1oo2D (1 out of 2 Diagnosis) solution using dynamic reconfiguration by Elektrobit [53]. In this situation, partial ECU can release the function by receiving the collected data from the sensor, and other partial can complete the function by sending the performing action to the actuator. That is, the entry task of a function can only be executed by specified ECUs that connect sensors, and the exit task of the function can only be executed by specified ECUs that connecting actuators.

We use $P = \{p_1, p_2, ..., p_{|P|}\}$ to represent a set of heterogeneous ECUs; in this equation, $|P|$ represents the size of set P. Notably, for any set X, this study uses $|X|$ to denote its size.

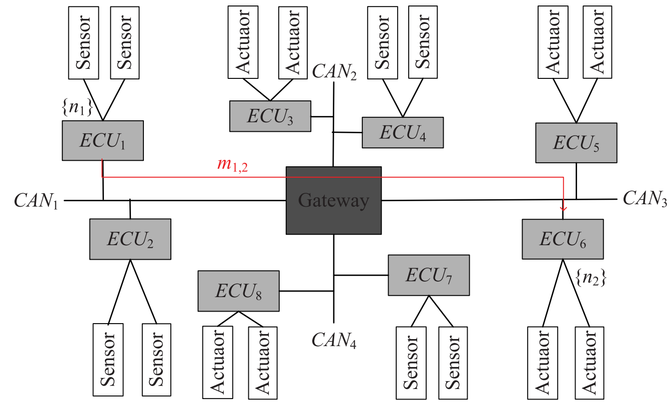

Fig. 1. Architecture of a CAN cluster with four buses interconnected by a central gateway.

When a task is executed completely in one ECU, the task sends messages to all of its successor tasks that may be located in the different ECUs of different buses. For example, in Fig. 1, task n_1 is executed on ECU_1 of CAN_1. When this task is executed completely, it then sends a message $m_{1,2}$ to its successor task n_2 located in the ECU_6 of CAN_3. The central gateway is a highly important node that connects the CAN cluster and allows messages to be passed from one bus to another.

B. Criticality Level

ISO 26262 identifies four criticality levels denoted by ASIL (i.e., A, B, C, and D) of automotive functions [10]. ASIL is established by analyzing the severity, exposure, and controllability of a vehicle under a hazard scenario [10]. Severity means the injury degree caused by accidents, such as missing the deadlines of functions with different severity levels will cause different injuries, and is usually evaluated by DMR. Exposure means the relative expected probability caused by random hardware failures in which the injury may happen and is evaluated by reliability [10]. In addition, as pointed out in ISO 26262, reliability (or exposure) is only related with random hardware failures, which occur unpredictably during the life time of a hardware, but follows a probability distribution [10]. Controllability depends on the states of drivers when running.

The safety requirement of an automotive function is actually the combination of the real-time requirement, the reliability requirement of the function, and the controllability requirements of drivers. Similar to [44], [45], we ignore the issue of reliability (which is orthogonal to our problem) and assume that the designer has developed functions that provide the required level of reliability. Controllability is related to drivers rather than systems and is a computer-human interaction problem; all developed functions are assumed to be uncontrollable by drivers. In other words, this study focuses on severity, which is one of the parts of ASIL. Severity also involves four levels, namely, S0, S1, S2, and S3, which represent no injuries, light to moderate injuries, severe to life-threatening injuries, and life-threatening to fatal injuries, respectively. Obviously, S0 and S3 represent the lowest criticality level and the highest criticality level, respectively, among these criticality levels [10]. Hence, $S = \{S_0, S_1, S_2, S_3\}$ is employed to represent a set of the severity levels in ACPS.

C. Mixed-Criticality Function Model

A distributed function is represented by a task graph DAG $F_m = (N, W, M, C)$ [33]–[36], where F_m represents the mth function in systems. The tasks, messages, and other attributes are described as follows:

(1) N represents a set of nodes in F_m, and each node $n_i \in N$ represents a task with different worst-case execution times (WCETs) on different ECUs. In general, a task has different WCET values in different criticality levels on mixed-criticality systems and the WCET in high-criticality level is larger than or equal to that in low-criticality level. Similar to [44], we also assume that each task has the same WCET value regardless of the criticality level for simplicity in this study. This assumption is a special case of the general case with varying WCETs and the proposed approaches can also be applied to the general case, as long as the deadlines of functions are certificated in the highest criticality level. $pred(n_i)$ represents the set of the immediate predecessor tasks of n_i. $succ(n_i)$ represents the set of the immediate successor tasks of n_i. The task which has no predecessor task is denoted as n_{entry}; and the task which has no successor task is denoted as n_{exit}. If a DAG-based function has multiple n_{entry} or multiple n_{exit} tasks, then a dummy entry or exit task with zero-weight dependencies is added to the graph. W is an $|N| \times |P|$ matrix where $w_{i,k}$ denotes the WCET of n_i runs on p_k.

(2) Communication between tasks mapped to different ECUs is performed by message passing over the bus; hence, M is a set of communication edges, and each edge $m_{i,j} \in M$ represents the communication message from n_i to n_j. Accordingly, C represents end-to-end worst case response time (WCRT) sets of messages. $c_{i,j} \in C$ represents the end-to-end WCRT of $m_{i,j}$, and it includes the gateway processing time of $m_{i,j}$ [45], [52], [54].

(3) The aforementioned parameters are the basic properties of the distributed functions in heterogeneous distributed systems and are used by several algorithms (e.g., heterogeneous earliest finish time (HEFT) [33] and heterogeneous selection value (HSV) [36]). For a distributed function in mixed-criticality ACPS, the remaining attributes ($arrivaltime$, $criticality$, $lowerbound$, $deadline$, and $makespan$) need to be used. $arrivaltime$ represents the arrival time (i.e., released time) of the function. $criticality \in S$ represents the criticality level of F_m. $lowerbound$ indicates the minimum makespan of a function when all ECUs are monopolized by the function using a standard DAG-based single-functional scheduling algorithm (e.g., HEFT [33]). $deadline$ means the relative deadline of the function and should be larger than or equal to $lowerbound$. Notably, the start time instant of the relative deadline is the arrival time of the function. $criticality$, $lowerbound$, and $deadline$ must be certificated by a certification authority (CA) (refer to Section IV-A for concrete certification). $makespan$ represents the actual makespan of F_m in multi-functional scheduling. Note that if a distributed function is periodically released, then we treat each instance of this function as a new dynamic function. In this way, all periodically released functions can also be considered special cases of dynamic functions. That is, the models and scheduling approaches presented in this study can also be applied to periodically released functions.

D. Mixed-Criticality System Model

A mixed-criticality ACPS comprises of multiple distributed functions with different criticality levels and is denoted as $MS = \{F_1, F_2, ..., F_{|MS|}\}$. Let $MS.criticality$ indicate the current criticality level of ACPS. In distinguishing the ambiguities, we use $MS.criticality$ to express the $criticality$ of MS and use $F_m.criticality$ to express the $criticality$ of the function F_m. Other attributes use the same expression. In mixed-criticality systems, $MS.criticality$ can be changed to high-criticality levels and back to low-criticality levels. A change in $MS.criticality$ indicates a switch in system mode. F_m can only be executed on the modes in which $F_m.criticality$ is higher than or equal to $MS.criticality$ [9]. Note that the number of functions in MS is dynamically increased with time.

According to the AUTOSAR standard, the scheduling of distributed functions in automobiles depends on the operating system (OS) running on ECUs, in which common automotive OSs can be either preemptive (e.g., OSEKTime) or non-preemptive (e.g., eCos) [17]. In this study, we consider non-preemptive scheduling for ECUs.

To implement the requirement of assigning tasks to different ECUs in a dynamic fashion, we emphasize the following conditions: 1) source code for each task needs to be presented on each ECU; 2) the data for the task needs to be available on the ECU which requires dynamic sending of data via messages; 3) all ECUs have to be certified to the highest criticality level; and 4) a TIER 1 supplier of one ECU has to allow the execution of a task possible designed by another TIER 1 supplier which raises questions regarding liability. Note that if all software developers and products comply with the AUTOSAR standard, then the last implication does not need to be considered, because the AUTOSAR standard allows the functions to run on different hardwares to improve development efficiency by AUTOSAR runtime environment (RTE). That is, AUTOSAR introduces the RTE to shield the details that are related to hardwares, such that the code portability in different hardwares is easy. The above requirement is basically similar to the software requirement (i.e., functions can be dynamically relocated) of 1oo2D salutation using dynamic reconfiguration by Elektrobit [53].

E. Motivating Example

Fig. 2 shows a motivating example of mixed-criticality ACPS with three functions, namely, F_1, F_2, and F_3, with $F_1.criticality = S_1$, $F_2.criticality = S_2$, and $F_3.criticality = S_3$. Table II shows the WCETs of tasks for F_1, F_2, and F_3 in Fig. 2. The example shows six tasks for F_1, five tasks for F_2, and six tasks for F_3. Three ECUs exists for ACPS in this motivating example. Although the example is simple, three ECUs, three functions, and three criticality levels are

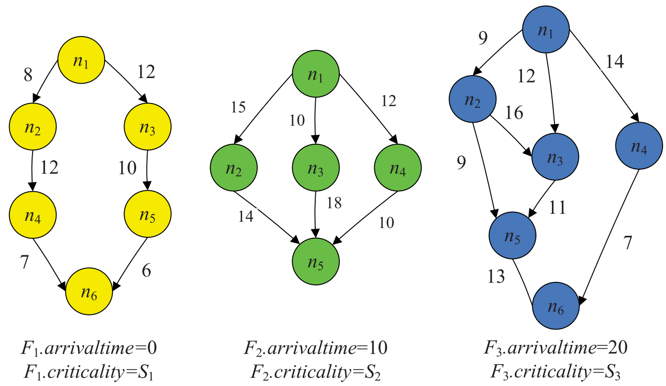

Fig. 2. Motivating example of ACPS containing three distributed functions with different criticality levels ($F_1.criticality = S_1$, $F_2.criticality = S_2$, and $F_3.criticality = S_3$).

involved. This example can reflect the characteristics of multiple ECUs, multiple functions, and multiple criticality levels in ACPS. The weight 8 of the edge between task $F_1.n_1$ and task $F_1.n_2$ represents the WCRT of $F_1.m_{1,2}$ if $F_1.n_1$ and $F_1.n_2$ are not assigned in the same ECU. The weight 12 of $F_1.n_1$ and p_1 in Table II(a) represents the WCET and is denoted as $F_1.w_{1,1} = 12$. We can see that the same task has different WCETs on different ECUs due to the heterogeneity of ACPS. To explain our scheduling algorithms clearly and intuitively, all ECUs can execute the entry and exit tasks of all functions in this example.

F. Problem Description

Given multiple functions $MS = \{F_1, F_2, ..., F_{|MS|}\}$ that would be executed on a heterogeneous multiple ECU set $P = \{p_1, p_2, ..., p_{|P|}\}$ and criticality level set $S = \{S_0, S_1, S_2, S_3\}$ in ACPS, the formal description is to simultaneously reduce the overall makespan of ACPS:

$$MS.makespan = \max(F_1.makespan, F_2.makespan, ..., F_{|MS|}.makespan), \tag{1}$$

and the DMRs of high-criticality functions:

$$DMR(S_x) = \frac{|MS^{\text{miss}}(S_x)|}{|MS(S_x)|}, \tag{2}$$

with a reasonable tradeoff. $|MS^{\text{miss}}(S_x)|$ represents the number of the functions with criticality level S_x missing their absolute deadlines, and $|MS(S_x)|$ represents the number of all the functions with criticality level S_x. Obviously, the problem is an NP-hard optimization problem.

IV. Fairness-Based Dynamic Scheduling

This section presents fairness-based dynamic scheduling on multi-functional mixed-criticality ACPS from a high performance perspective.

A. Lower Bound and Deadline

ISO 26262 requires designers to assess and eliminate all potential risks of automotive functions in advance and as soon

TABLE II
Computation Time Matrixes of the Example in Fig. 2

(a) Computation Time Matrix of F_1

Tasks	$F_1.n_1$	$F_1.n_2$	$F_1.n_3$	$F_1.n_4$	$F_1.n_5$	$F_1.n_6$
p_1	12	9	7	13	18	15
p_2	8	15	12	15	10	10
p_3	9	11	16	18	20	8
$rank_u$	77	58	55	34	33	11

(b) Computation Time Matrix of F_2

Tasks	$F_2.n_1$	$F_2.n_2$	$F_2.n_3$	$F_2.n_4$	$F_2.n_5$
p_1	14	9	18	21	7
p_2	5	10	17	15	6
p_3	6	11	16	19	15
$rank_u$	64	34	45	39	10

(c) Computation Time Matrix of F_3

Tasks	$F_3.n_1$	$F_3.n_2$	$F_3.n_3$	$F_3.n_4$	$F_3.n_5$	$F_3.n_6$
p_1	8	14	9	18	18	5
p_2	11	13	12	15	16	10
p_3	19	8	16	14	20	7
$rank_u$	110	91	63	31	39	8

as possible. Accordingly, the goals of safety-related automotive functions, particularly safety-related functions, can be achieved. The HEFT algorithm is the most popular DAG-based single-functional scheduling algorithm for reducing makespan to a minimum while achieving low complexity and high performance in heterogeneous distributed systems [33]. The two-phase HEFT algorithm has the following important steps.

First, the HEFT algorithm uses the upward rank value ($rank_{\text{u}}$) of a task (Eq. (3)) as the task priority standard. In this case, the tasks are ordered according to the descending order of $rank_{\text{u}}$. Table II shows the upward rank values of all tasks (Fig. 2), which are obtained by using Eq. (3):

$$rank_{\text{u}}(F_m.n_i) = F_m.\overline{w_i} + \max_{F_m.n_j \in succ(F_m.n_i)} \{F_m.c_{i,j} + rank_{\text{u}}(F_m.n_j)\}, \tag{3}$$

where $F_m.\overline{w_i}$ represents the average WCET of task $F_m.n_i$.

Second, the attributes $EST(F_m.n_j, p_k)$ and $EFT(F_m.n_j, p_k)$ represent the earliest start time (EST) and earliest finish time (EFT), respectively, of task $F_m.n_j$ on ECU p_k. $EFT(F_m.n_j, p_k)$ is considered the task allocation criterion because it can meet the local optimal of each task. The aforementioned attributes are calculated as given by (4), shown at the bottom of the next page, and

$$EFT(F_m.n_j, p_k) = EST(F_m.n_j, p_k) + F_m.w_{j,k}. \tag{5}$$

$avail[k]$ is the earliest available time when ECU p_k is ready for task execution. $AFT(F_m.n_i)$ is the actual finish time (AFT) of task $F_m.n_i$. $F_m.c'_{i,j}$ represents the WCRT between $F_m.n_i$ and $F_m.n_j$. If $F_m.n_i$ and $F_m.n_j$ are allocated to the same ECU, then $F_m.c'_{i,j} = 0$; otherwise, $F_m.c'_{i,j} = F_m.c_{i,j}$. $F_m.n_j$ is allocated to the ECU with the minimum EFT by using the insertion-based scheduling strategy, where $F_m.n_j$ can be inserted into the slack with the minimum EFT.

TABLE III
LOWER BOUND COMPUTATION OF F_1

Task	$EFT(F_1.n_i, p_1)$	$EFT(F_1.n_i, p_2)$	$EFT(F_1.n_i, p_3)$
$F_1.n_1$	12	8	9
$F_1.n_2$	25	23	27
$F_1.n_3$	27	35	36
$F_1.n_4$	48	38	53
$F_1.n_5$	45	48	57
$F_1.n_6$	60	61	59

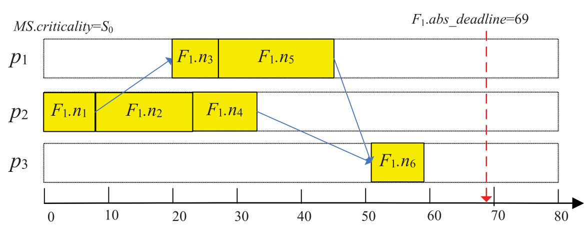

Fig. 3. Task scheduling for lower bound calculation.

Similar to [45], HEFT is employed by CAs to assess the lower bound of a distributed function. The lower bound refers to the minimum makespan of a function when all ECUs are monopolized by the function by using the standard DAG-based single-functional scheduling algorithm and is calculated as follows:

$$F_m.lowerbound = AFT(F_m.n_{\text{exit}}), \tag{6}$$

where $F_m.n_{\text{exit}}$ represents the exit task of F_m.

Table II(a) shows the upward rank values of all the tasks in Fig. 2(a). Note that only if all the predecessors of $F_m.n_i$ have been assigned to the processors, will $F_m.n_i$ prepare to be assigned. Assume that two tasks $F_m.n_i$ and $F_m.n_j$ satisfy $rank_{\text{u}}(F_m.n_i) > rank_{\text{u}}(F_m.n_j)$, if no precedence constraint exists between $F_m.n_i$ and $F_m.n_j$, then $F_m.n_i$ may not have higher priority than $F_m.n_j$. Finally, the task priorities in G is $\{F_1.n_1, F_1.n_2, F_1.n_3, F_1.n_4, F_1.n_5, F_1.n_6\}$.

As the task priorities have been obtained, we then explain the lower bound computation of F_1 using HEFT shown in Table III. First, $F_1.n_1$ is assigned to p_2 (denoted with red color) because it has the minimum WCET of 8. Then, $F_1.n_2$ is assigned to p_2 (denoted with red color) because it has the minimum EFT ($EFT(F_1.n_2, p_1) = 25$, $EFT(F_1.n_2, p_2) = 23$, $EFT(F_1.n_2, p_3) = 27$ calculated by Eq. (5)). $F_1.n_3$, $F_1.n_4$, $F_1.n_5$, and $F_1.n_6$ use the same pattern as $F_1.n_2$ to obtain the minimum EFT shown in Table III. Finally, lower bound of function F_1 is 59, which is the AFT of the exit task $F_1.n_6$. Fig. 3 shows the task scheduling for lower bound calculation of F_1.

TABLE IV
PROPERTIES OF FUNCTIONS IN FIG. 2

	F_1	F_2	F_3
Task priority	$F_1.n_1, F_1.n_2, F_1.n_3, F_1.n_4, F_1.n_5, F_1.n_6$	$F_2.n_1, F_2.n_3, F_2.n_4, F_2.n_2, F_2.n_5$	$F_3.n_1, F_3.n_2, F_3.n_3, F_3.n_5, F_3.n_4, F_3.n_6$
$arrivaltime$	0	10	20
$criticality$	S_1	S_2	S_3
$lowerbound$	59	52	54
$deadline$	69	62	64
$abs\_deadline$	69	72	84

A known relative deadline (i.e., $F_m.deadline$) is provided by CAs for each function on the basis of the actual physical time requirement after hazard analysis and risk assessment (Fig. 3). Considering the dynamics of ACPS, we should obtain the absolute deadline (the start time instant of F_m is 0) of each function, which is calculated as follows:

$$F_m.abs\_deadline = F_m.deadline + F_m.arrivaltime. \tag{7}$$

Table IV lists related properties of each function of the motivating example.

B. Dynamic Scheduling Framework

We propose a scheduling framework of dynamic multi-functional ACPS (Fig. 4). The framework has two main components: multi-functional pool, and heterogeneous distributed integrated architecture.

1) The multi-functional pool stores new dynamically arrived functions. Each function is submitted into the pool at any time instant.

2) The heterogeneous distributed integrated architecture contains different ECUs where allocated tasks can be executed.

The objective is to make all functions in the multi-functional pool be executed on the ECUs of the heterogeneous distributed integrated architecture. The scheduling framework introduces three types of priority queues: task priority, common ready, and task allocation queues.

1) The task priority queue ($F_m.task\_priority\_queue$) of each function and the tasks in $F_m.task\_priority\_queue$ are ordered according to descending $rank_{\text{u}}(F_m.n_i)$.

2) The common ready queue ($MS.common\_ready\_queue$) of ACPS for storing ready tasks and the tasks in $MS.common\_ready\_queue$ are also ordered according to descending $rank_{\text{u}}(F_m.n_i)$ as well.

$$\begin{cases} EST(F_m.n_{\text{entry}}, p_k) = 0; \\ EST(F_m.n_i, p_k) = \max\left\{ \begin{array}{c} avail[k], \\ \max\limits_{F_m.n_i \in pred(F_m.n_j)} \{AFT(F_m.n_i) + F_m.c'_{i,j}\} \end{array} \right\}; \end{cases} \tag{4}$$

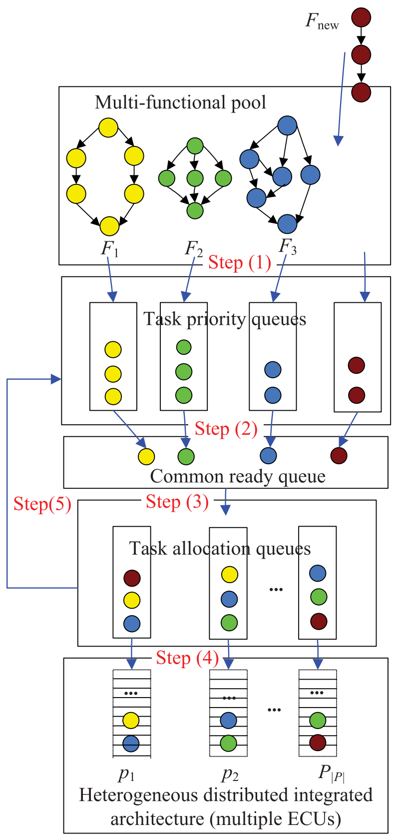

Fig. 4. Scheduling framework of dynamic multi-functional ACPS.

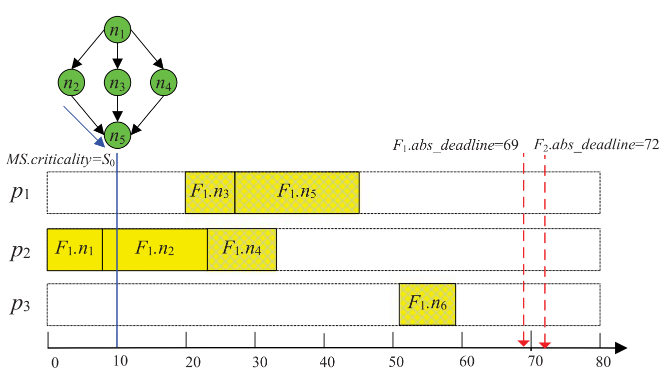

Fig. 5. Example of dynamic arrival, and the current time instant is 10 when F_2 arrives.

3) The task allocation queue ($p_k.task\_allocation\_queue$) of each ECU is for storing allocated tasks.

C. Fairness-Based Dynamic Scheduling Algorithm

Different functions may come at different time instants. As shown in Fig. 5, at time instant 0, the tasks of F_1 are assigned to the task allocation queues of ECUs. At time instant 10, F_2 arrives. The tasks in F_1 could then be divided into three task groups: (1) the group where tasks are executed ($F_1.n_1$); (2) the group where tasks are being executed ($F_1.n_2$); (3) the group where tasks are assigned to task allocation queues of ECUs but have not been started for execution in ECUs ($F_1.n_3$, $F_1.n_4$, $F_1.n_5$, and $F_1.n_6$).

Given that we use the non-preemptive scheduling strategy, $F_1.n_2$ cannot be interrupted by high-criticality functions. Considering that tasks $F_1.n_3$, $F_1.n_4$, $F_1.n_5$, and $F_1.n_6$ have not been started for execution in ECUs, their allocation can be canceled. On the basis of the above analysis, we optimize the objective of fairness from a high performance perspective. The fairness-based strategy of steps in this study is shown in Fig. 4. The steps are described as follows:

Step 1): Task priority: Place all the tasks of each function into corresponding task priority queue $F_m.task\_priority\_queue$ according to the descending order of $rank_{\text{u}}(F_m.n_i)$.

Step 2): Task readiness: Select the ready tasks with the maximum $rank_{\text{u}}(F_m.n_i)$ from each function, and place them into the common ready queue $MS.common\_ready\_queue$ according to the descending order of $rank_{\text{u}}(F_m.n_i)$, namely, $rank_{\text{u}}$ is also used to determine the priorities of the tasks in $MS.common\_ready\_queue$. We know that multiple functions can be merged into a new function by adding a common virtual entry task and an exit task. The $rank_{\text{u}}$ value for each task in the merged function is the same as in the original single function, so the $rank_{\text{u}}$ value can be used to determine the priority across multiple functions.

Step 3): Task allocation: Select a task with maximum $rank_{\text{u}}(F_m.n_i)$ from the $MS.commom\_ready\_queue$, and place it into the task allocation queue $p_k.task\_allocation\_queue$ of an ECU with minimum $EFT(F_m.n_i, p_k)$ using the insertion-based strategy and mark its actual start time (AST) and AFT in the task allocation queue, where $AFT(F_m.n_i)$ is equal to the minimum $EFT(F_m.n_i, p_k)$ and $AST(F_m.n_i) = AFT(F_m.n_i) - F_m.w_{i,k}$.

Note that each task is first allocated to the task allocation queue rather than ECU itself. Only when the ST of the task is equal to the current time instant, is the task assigned to the ECU and executed. Repeat Step (3) until the $MS.common\_reday\_queue$ is empty. Repeat Step (2) until no ready task can be selected.

Step 4): Task scheduling: Schedule all the tasks according to the assignment in task allocation queues.

Step 5): Arrival of a new function: If a new function F_{new} arrives (implemented by interrupt service routine (ISR)), then add it to MS and calculate $rank_{\text{u}}(F_{\text{new}}.n_i)$ for all the tasks of F_{new}, and place them into the $F_{\text{new}}.task\_priority\_queue$; cancel all the tasks that are waiting to be scheduled in the task allocation queues of all ECUs and place them into the corresponding task priority queues.

Compared with the latest typical multi-functional dynamic scheduling algorithm that aims to minimize individual makespans of functions (i.e., FDWS [8]), our scheduling strategy has the following improvements:

1) Each task is first allocated to the task allocation queue of the ECU regardless of whether the ECU is idle when using our strategy (Step (4)), whereas FWDS needs to determine whether the ECU is idle and the task should wait for scheduling when the ECU is not idle.

Algorithm 1: The FDS_MIMF Algorithm.

Input: $P = \{p_1, p_2, ..., p_{|P|}\}$, $S = \{S_0, S_1, S_2, S_3\}$, and $MS = \{F_1, F_2, ..., F_{|MS|}\}$
Output: Schedule results
for $(m \leftarrow 1; m \leqslant |MS|; m++)$ **do**
  $F_m.task\_priority\_queue(F_m).add(F_m.n_i)$;
end for//Step (1)
while (functions to be scheduled in MS exist) **do**
  for $(m \leftarrow 1; m \leqslant |MS|; m++)$ **do**
    $n_i \leftarrow F_m.task\_priority\_queue.out()$;
    $MS.common\_ready\_queue.put(n_i)$;
  end for//Step (2)
  while (!$common\_redaly\_queue.empty()$) **do**
    $n_i \leftarrow MS.common\_ready\_queue.out()$;
    Assign n_i to the $task\_allocation\_queue(p_k)$ of the ECU p_k with the EFT using the insertion-based strategy; //Step (3)
  end while
  Schedule all tasks according to the assignment in task allocation queues; //Step (4)
  if (a new function F_{new} arrives) **then**
    MS.add(F_{new});
    $task\_priority\_queue(F_{\text{new}}).add(F_{\text{new}}.n_i)$;
    Get unexecuted tasks $unexecuted\_tasks$ of other functions;
    $unexecuted\_tasks \xrightarrow{\text{back}} task\_priority\_queues$;
  end if//Step (5)
end while

2) If a new function arrives, our strategy can cancel all tasks that have not been started for execution (Step (5)) and these tasks can be fairly rescheduled with the tasks of new functions, whereas FDWS blocks tasks and leads to delays in waiting for the completion of other tasks.

On the basis of the above analysis, we propose round-robin *fairness-based dynamic scheduling with the objective of minimizing individual makespans of functions* (FDS_MIMF) for ACPS. The steps of FDS_MIMF are described in Algorithm 1. The time complexity of the FDS_MHEFT algorithm is $O(|MS| \times N_{\max}^2 \times |P|)$, where $N_{\max} = \max(|F_1.N|, |F_2.N|, ..., |F_{|MS|}.N|)$, which is the same as that of FDWS.

D. Example of the FDS_MIMF Algorithm

Figs. 3 and 5–8 show the Gantt charts of the scheduling, and Table V shows the corresponding steps of task operations of the motivating example using the FDS_MIMF algorithm. Notably, the actual execution time of each task should be less than or equal to its WCET in dynamic scheduling. To explain our proposed scheduling algorithm clearly, all tasks can be executed with their WCETs, and this assumption will not influence the effectiveness of the algorithm in the actual situation.

1) The current time instant of ACPS is 0 when F_1 arrives, and F_1 is scheduled using the HEFT algorithm (Fig. 3 of Section IV-A). As only one function in time instant 0, the results of HEFT are equivalent to that of FDS_MIMF.
2) The current time instant is changed to 10 when F_2 arrives (Fig. 5 of Section IV-C). To achieve fairness between F_1 and F_2, the tasks ($F_1.n_3$, $F_1.n_4$, $F_1.n_5$, $F_1.n_6$) of F_1 that have not been started for execution are canceled (denoted as shadowgraphs in Fig. 5).
3) In the current time instant 10, the tasks ($F_1.n_3$, $F_1.n_4$, $F_1.n_5$, $F_1.n_6$) of F_1 that have not been started for execution and all the tasks of F_2 are fairly scheduled (Fig. 6). Both F_1 and F_2 meet their absolute deadlines.
4) The current time instant is changed to 20 when F_3 arrives (Fig. 7). To achieve fairness among F_1, F_2, and F_3, the tasks of F_1 and F_2 that have not been started for execution are canceled (denoted as shadowgraphs in Fig. 7).
5) In the current time instant 20, the tasks of F_1 and F_2 that have not been started for execution and all the tasks of F_3 are fairly scheduled (Fig. 8).

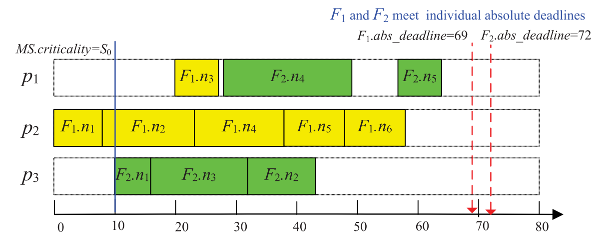

Fig. 6. In the current time instant 10, the tasks ($F_1.n_3$, $F_1.n_4$, $F_1.n_5$, $F_1.n_6$) of F_1 that have not been started for execution and all the tasks of F_2 are fairly scheduled.

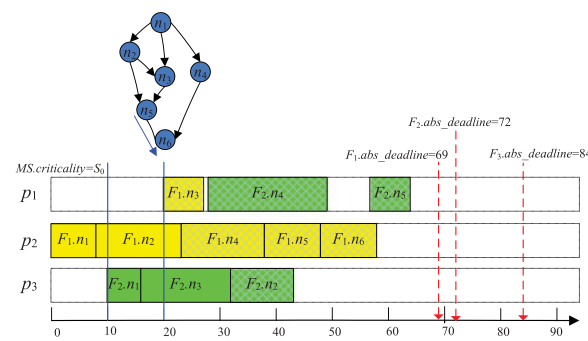

Fig. 7. Current time instant is changed to 20 when F_3 arrives; the tasks ($F_1.n_3$, $F_1.n_4$, $F_1.n_5$, $F_1.n_6$) have not been started for execution of F_1 and the tasks ($F_2.n_2$, $F_2.n_4$, $F_2.n_5$) have not been started for execution of F_2 are canceled.

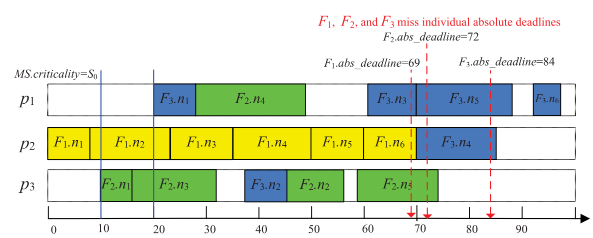

Fig. 8. In the current time instant 20, the tasks ($F_1.n_3$, $F_1.n_4$, $F_1.n_5$, $F_1.n_6$) of F_1 that have not been started for execution, the tasks ($F_2.n_2$, $F_2.n_4$, $F_2.n_5$) of F_2 that have not been started for execution, and all the tasks of F_3 are fairly scheduled.

TABLE V
TASK ALLOCATION STEPS OF THE MOTIVATING EXAMPLE USING THE FDS_MIMF ALGORITHM

Step	Figure	Current instant	System's criticality	Operation	Operated tasks and orders
1	Fig. 3	0	S_0	Allocation	$F_1.n_1, F_1.n_2, F_1.n_3, F_1.n_4, F_1.n_5, F_1.n_6$
2	Fig. 5	10	S_0	Cancel	$F_1.n_3, F_1.n_4, F_1.n_5, F_1.n_6$
3	Fig. 6	10	S_0	Allocation	$F_2.n_1, F_1.n_3, F_2.n_3, F_1.n_4, F_2.n_4, F_1.n_5, F_2.n_2, F_1.n_6, F_2.n_5$
4	Fig. 7	20	S_0	Cancel	$F_1.n_3, F_1.n_4, F_1.n_5, F_1.n_6, F_2.n_2, F_2.n_4, F_2.n_5$
5	Fig. 8	20	S_0	Allocation	$F_3.n_1, F_1.n_3, F_2.n_4, F_3.n_2, F_1.n_4, F_2.n_2, F_3.n_3, F_1.n_5, F_2.n_5, F_3.n_5, F_1.n_6, F_3.n_4, F_3.n_6$

Finally, the obtained makespans of functions are as follows: $F_1.makespan = 70$, $F_2.makespan = 74$, and $F_3.makespan = 97$. However, all functions miss their absolute deadlines because $F_1.makespan > F_1.abs\_deadline = 69$, $F_2.makespan > F_2.abs\_deadline = 72$, and $F_3.makespan > F_3.abs\_deadline = 84$. In other words, the results show high performance with short overall makespan of ACPS, but shows high DMR value of 1 that all functions miss their absolute deadlines.

V. ADAPTIVE DYNAMIC SCHEDULING

In mixed-criticality ACPS, an important concept is that a function can be scheduled only when its criticality is higher than or equal to the system criticality as mentioned earlier. We can use the FDS_MIMF algorithm (Algorithm 1) to schedule all functions with different criticality levels and to achieve a short overall makespan of ACPS; however, the absolute deadlines of several high-criticality functions may be missed. To meet the absolute deadlines of high-criticality functions and to reduce their DMRs, a novel solution is proposed and discussed in this section.

A. Deadline-Slack

Definition 1(Deadline-slack): The deadline-slack of a function represents the slack between the relative deadline and the lower bound of the function, that is,

$$F_m.deadlineslack = F_m.deadline - F_m.lowerbound. \tag{8}$$

Considering that we have used the HEFT algorithm to calculate the $F_m.lowerbound$, $F_m.deadlineslack$ is actually determined by $F_m.deadline$. As the precedence constraints between tasks in the function F_m, each task should also have an individual absolute deadline. In fact, the absolute deadline of F_m is the absolute deadline of the exit task $F_m.n_{\text{exit}}$. Thereafter, the absolute deadline of task n_i ($n_i \in F_m$) can be generated. Thus,

$$cabs\_deadline(F_m.n_i) = F_m.arrivaltime + lowerbound(F_m.n_i) + F_m.deadlineslack, \tag{9}$$

where $lowerbound(F_m.n_i) = AFT(F_m.n_i)$ represents the lower bound of $F_m.n_i$. That is, all tasks have individual lower bounds and absolute deadlines. The deadline-slacks of all functions are obtained using Eq. (8) ($F_1.deadlineslack = 10$, $F_2.deadlineslack = 10$, and $F_3.deadlineslack = 10$) of the motivating example. The absolute deadlines of all tasks are calculated using Eq. (9). Table VI shows all the absolute deadlines of tasks in the motivating example. Note that Table IV in Section IV-A shows the related values of functions.

TABLE VI
ABSOLUTE DEADLINES OF TASKS OF THE MOTIVATING EXAMPLE IN FIG. 2

(a) Deadlines of Tasks of F_1

Task	$F_1.n_1$	$F_1.n_2$	$F_1.n_3$	$F_1.n_4$	$F_1.n_5$	$F_1.n_6$
Absolute deadline	18	33	37	48	55	69

(b) Deadlines of Tasks of F_2

Task	$F_2.n_1$	$F_2.n_2$	$F_2.n_3$	$F_2.n_4$	$F_2.n_5$
Absolute deadline	25	49	42	56	72

(c) Deadlines of Tasks of F_3

Task	$F_3.n_1$	$F_3.n_2$	$F_3.n_3$	$F_3.n_4$	$F_3.n_5$	$F_3.n_6$
Absolute deadline	38	52	61	66	79	84

B. The ADS_MIMF Algorithm

As observed in Table V, the system criticality using the FDS_MIMF algorithm always stays at S_0, which is the lowest criticality. Thus, all functions can be scheduled fairly for high performance but the criticality levels of functions are completely ignored. To meet the absolute deadlines of more high-criticality functions and maintain the satisfactory performance of ACPS, we propose an adaptive dynamic scheduling strategy that is driven by the change in the system criticality. The proposed algorithm is called the *adaptive dynamic scheduling on the basis of minimizing individual makespans of functions* (ADS_MIMF), and the steps are described in Algorithm 2.

The main idea of ADS_MIMF is that when the fairness-based strategy cannot meet the absolute deadline of a task belonging to a high-criticality function F_m, the system criticality is changed up to the criticality of the function F_m. Thereafter, only the tasks of functions whose criticality levels are equal to or larger than the system criticality are scheduled fairly. After all the tasks of the function F_m are scheduled, the system criticality is changed down to S_0. Finally, the remaining tasks of functions are fairly scheduled. The concrete steps are explained as follows:

1) Initialize the criticality of ACPS as S_0 (i.e., $MS.criticality \leftarrow S_0$) (Line 4 of Algorithm 2), and prepare to schedule all functions fairly in the mode of $MS.criticality = S_0$.
2) If $n_i \in F_m$ meets $makespan(F_m.n_i) > abs\_deadline(F_m.n_i)$ and $F_m.criticality > MS.criticality$ (Line 16 of Algorithm 2), conduct the following works: a) cancel the allocation of tasks of the current and previous

Algorithm 2: The ADS_MIMF Algorithm.

Input: $P = \{p_1, p_2, ..., p_{|P|}\}$, $S = \{S_0, S_1, S_2, S_3\}$, and $MS = \{F_1, F_2, ..., F_{|MS|}\}$
Output: Schedule results
1: **for** $(m \leftarrow 1; m \leqslant |MS|; m++)$ **do**
2: $F_m.task\_priority\_queue(F_m).add(F_m.n_i)$;
3: **end for**
4: $MS.criticality \leftarrow S_0$;
5: **while** (there are tasks to be allocated) **do**
6: **for** $(m \leftarrow 1; m \leqslant |MS|; m++)$ **do**
7: **if** $(F_m.criticality < MS.criticality)$ **then**
8: **continue**;
9: **end if**
10: $n_i \leftarrow task\_priority\_queue(F_m).out()$;
11: $common\_ready\_queue.put(n_i)$;
12: **end for**
13: **while** (!$common\_ready\_queue.empty()$) **do**
14: $F_m.n_i \leftarrow common\_ready\_queue.out()$;
15: Assign $F_m.n_i$ to $task\_allocation\_queue(p_k)$ with the minimum EFT using the insertion-based scheduling strategy;
16: **if** $(makespan(F_m.n_i) > abs\_deadline(F_m.n_i)$ && $F_m.criticality > MS.criticality)$ **then**
17: Cancel the task allocations $canceled\_tasks$ of current and previous rounds of unscheduled completed functions;
18: $canceled\_tasks \xrightarrow{\text{back}} task\_priority\_queues$;
19: $cleared\_tasks \leftarrow common\_ready\_queue.clear()$;
20: $cleared\_tasks \xrightarrow{\text{back}} task\_priority\_queues$;
21: $MS.criticality \leftarrow F_m.criticality$;
22: **end if**
23: **if** (F_m, which is the function causing the system criticality to be changed up, is scheduled completed) **then**
24: $cleared\_tasks \leftarrow common\_ready\_queue.clear()$;
25: $cleared\_tasks \xrightarrow{\text{back}} task\_priority\_queues$;
26: $MS.criticality \leftarrow S_0$;
27: **end if**
28: **end while**
29: **if** (a new function F_{new} arrives) **then**
30: MS.add(F_{new});
31: $task\_priority\_queue(F_{\text{new}}).add(F_{\text{new}}.n_i)$;
32: Get unexecuted tasks $unexecuted\_tasks$ of other functions;
33: $unexecuted\_tasks \xrightarrow{\text{back}} task\_priority\_queues$;
34: **end if**
35: **end while**

rounds except for the tasks in scheduled completed functions (one round means the allocation that the tasks placed into the common ready queue together; given that the current round is not completed, the next round would still be the current round allocation if the current round allocation is merely canceled. In this case, the two rounds need to be canceled); b) clear the tasks in the common ready queue and place them back to individual task priority queues; c) change the system criticality to the criticality of F_m, namely, $MS.criticality \leftarrow F_m.criticality$;

3) Fairly schedule the functions whose criticality level is larger than or equal to $MS.criticality$ until all tasks of F_m is scheduled completely.
4) If F_m is the function causing the system criticality to be changed up and F_m is scheduled completed, change the system criticality to S_0, namely, $MS.criticality = S_0$ (Lines 23-27 of Algorithm 2);
5) If a new function arrives, cancel all tasks that have not been started for execution (implemented by ISR). These tasks can be fairly scheduled with the tasks of the new functions (Lines 29-34 of Algorithm 2).

Last, an important advantage is that ADS_MIMF achieves lower DMRs of safety-critical functions while maintaining the satisfactory overall makespan of ACPS without increasing the time complexity. The time complexity of the ADS_MIMF algorithm should be O($|MS| \times N_{\text{max}}^2 \times |P|$), which is equal to that of the FDS_MIMF algorithm. That is, changing the system criticality to implement adaptive dynamic scheduling does not increase the time complexity.

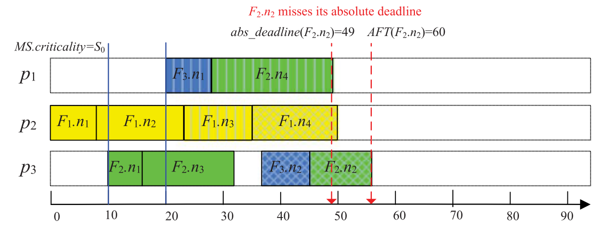

Fig. 9. Current time instant is 20; the makespan of $F_2.n_2$ is 56, which is larger than the absolute deadline 49 of $F_2.n_2$; then, the tasks of the current round $\{F_3.n_2, F_1.n_4, F_2.n_2\}$ and the previous round $\{F_3.n_1, F_1.n_3, F_2.n_4\}$ should be canceled.

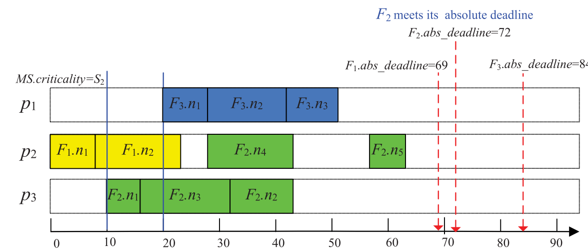

Fig. 10. Change the system criticality up to S_2; the tasks of F_2 and F_3 that have not been started for execution are then fairly scheduled until all the tasks of F_2 are allocated.

C. Example of the ADS_MIMF Algorithm

Figs. 9–13 show the Gantt charts and Table VII shows the corresponding steps of task operations of the motivating example using the ADS_MIMF algorithm. Note that the first four steps

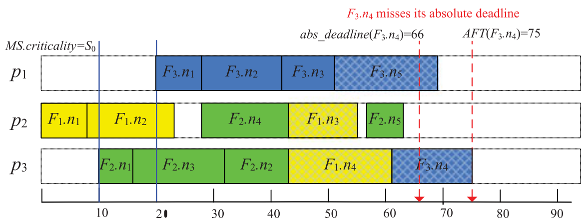

Fig. 11. Change the system criticality down to S_0; the makespan of $F_3.n_4$ is 75, which is larger than the absolute deadline 66 of $F_3.n_4$; then, the tasks of the current round $\{F_1.n_4, F_3.n_4\}$ and the previous round $\{F_1.n_3, F_3.n_5\}$ should be canceled.

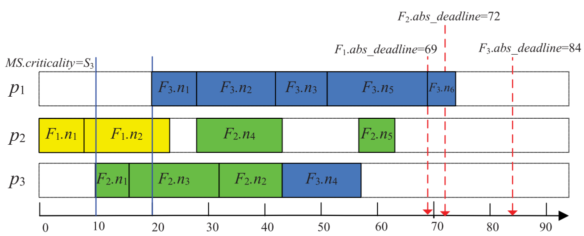

Fig. 12. Change the system criticality up to S_3; the tasks of F_3 that have not been started for execution are then scheduled until all the tasks of F_3 are allocated.

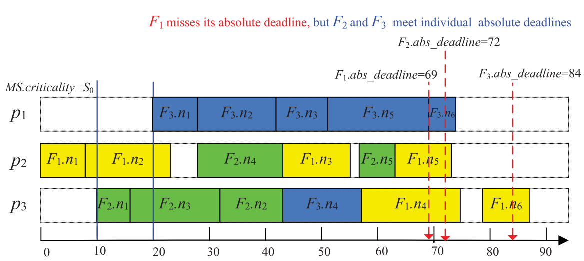

Fig. 13. The system criticality is changed down to S_0; the tasks of F_1 that have not been started for execution are to be scheduled.

(Figs. 3 and 5–7) are the same as those of the FDS_MIMF algorithm because no task $n_i \in F_m$ meets $makespan(F_m.n_i) > abs\_deadline(F_m.n_i)$ in $MS.criticality = S_0$ until F_3 arrives at time instant 20.

1) In current time instant 20, the tasks of F_1, F_2 that have not been started for execution and all the tasks of F_3 are to be fairly scheduled until allocating $F_2.n_2$ (Fig. 9). In the current round of allocating $F_2.n_2$, the tasks required to be allocated are $\{F_3.n_2, F_1.n_4, F_2.n_2\}$. The makespan of $F_2.n_2$ is 56, which is larger than the absolute deadline of $F_2.n_2$ of 49, that is, $AFT(F_2.n_2) > abs\_deadline(F_2.n_2)$ (Fig. 9). Then, the tasks of the current round $\{F_3.n_2, F_1.n_4, F_2.n_2\}$ and the previous round $\{F_3.n_1, F_1.n_3, F_2.n_4\}$ should be canceled (denoted as shadowgraphs Fig. 9).
2) Change the system criticality up to S_2 (Fig. 10), which is the criticality of F_2. The tasks of F_2 and F_3 that have not been started for execution are then fairly scheduled because $F_2.criticality = S_2$ and $F_3.criticality = S_3$. In this process, no task $n_i \in F_m$ meets $AFT(F_m.n_i) > abs\_deadline(F_m.n_i)$ in $MS.criticality = S_2$ until all the tasks of F_2 are allocated.
3) Considering that all the tasks of F_2 are allocated, then change the system criticality down to S_0, and the tasks of F_1 and F_3 that have not been started for execution are to be fairly scheduled until allocating $F_3.n_4$ (Fig. 11). In the current round of allocating $F_3.n_4$, the tasks required to be allocated are $\{F_1.n_4, F_3.n_4\}$. The makespan of $F_3.n_4$ is 75, which is larger than the absolute deadline of 66 of $F_3.n_4$, that is, $AFT(F_3.n_4) > abs\_deadline(F_3.n_4)$ (Fig. 11). Then, the tasks of the current round $\{F_1.n_4, F_3.n_4\}$ and the previous round $\{F_1.n_3, F_3.n_5\}$ should be canceled (denoted as shadowgraphs Fig. 11).
4) Change the system criticality up to S_3 (Fig. 12), which is the criticality of F_3. The tasks F_3 that have not been started for execution are then scheduled because $F_3.criticality = S_3$. In this process, no task $n_i \in F_m$ meets $AFT(F_m.n_i) > abs\_deadline(F_m.n_i)$ in $MS.criticality = S_3$ until all the tasks of F_3 are allocated.
5) Considering that all the tasks of F_3 are allocated, the system criticality is changed down to S_0 (Fig. 13), and the tasks of F_1 that have not been started for execution are to be scheduled. Finally, $F_1.makespan = 87$, which is larger than $F_1.abs\_deadline = 69$. Hence, F_1 misses its absolute deadline. However, $F_2.makespan = 63$ and $F_3.makespan = 74$, which are less than $F_2.abs\_deadline = 72$ and $F_3.abs\_deadline = 84$, respectively. Therefore, high-criticality functions F_2 and F_3 meet their individual absolute deadlines. Although F_1 misses its absolute deadline, it is a low-criticality function and will not cause fatal injuries in this situation.

We then summarize the following observations on the FDS_MIMF and ADS_MIMF algorithms.

1) The FDS_MIMF algorithm aims to minimize the individual makespans of functions with satisfactory overall makespan of ACPS from a high performance perspective and ignores the real-time properties of all functions. FDS_MIMF can respond autonomously to the joint challenges of heterogeneity, dynamics, and parallelism of ACPS.
2) The ADS_MIMF algorithm aims to meet the absolute deadlines of more high-criticality functions while still keeping satisfactory overall makespan of ACPS. That is, the ADS_MIMF algorithm achieves low DMR and satisfactory system performance by changing the system criticality. ADS_MIMF can respond autonomously to the joint challenges of heterogeneity, dynamics, parallelism, safety, and criticality of ACPS.

VI. Experiments

A. Experimental Metrics

The performance metrics selected for comparison are the overall makespan of ACPS (Eq. (1)) and the DMR of the functions (Eq. (2)). We implemented a simulated heterogeneous CAN cluster with five buses by using Java on a standard desktop computer. This platform can generate a variety of function samples (including active safety, passive-safety, and

TABLE VII
TASK ALLOCATION STEPS OF THE MOTIVATING EXAMPLE USING THE ADS_MIMF ALGORITHM

Step	Figure	Current instant	System's criticality	Operation	Operated tasks and orders
1	Fig. 3	0	S_0	Allocation	$F_1.n_1, F_1.n_2, F_1.n_3, F_1.n_4, F_1.n_5, F_1.n_6$
2	Fig. 5	10	S_0	Cancel	$F_1.n_3, F_1.n_4, F_1.n_5, F_1.n_6$
3	Fig. 6	10	S_0	Allocation	$F_2.n_1, F_1.n_3, F_2.n_3, F_1.n_4, F_2.n_4, F_1.n_5, F_2.n_2, F_1.n_6, F_2.n_5$
4	Fig. 7	20	S_0	Cancel	$F_1.n_3, F_1.n_4, F_1.n_5, F_1.n_6, F_2.n_2, F_2.n_4, F_2.n_5$
5	Fig. 9	20	S_0	Allocation	$F_3.n_1, F_1.n_3, F_2.n_4, F_3.n_2, F_1.n_4, F_2.n_2$
6	Fig. 9	20	S_0	Cancel	$F_3.n_1, F_1.n_3, F_2.n_4, F_3.n_2, F_1.n_4, F_2.n_2$
7	Fig. 10	20	S_2	Allocation	$F_3.n_1, F_2.n_4, F_3.n_2, F_2.n_2, F_3.n_3, F_2.n_5$
8	Fig. 11	20	S_0	Allocation	$F_1.n_3, F_3.n_5, F_1.n_4, F_3.n_4$
9	Fig. 11	20	S_0	Cancel	$F_1.n_3, F_3.n_5, F_1.n_4, F_3.n_4$
9	Fig. 12	20	S_3	Allocation	$F_3.n_5, F_3.n_4, F_3.n_6$
10	Fig. 13	20	S_0	Allocation	$F_1.n_4, F_1.n_5, F_1.n_6$

TABLE VIII
OVERALL MAKESPANS (μS) FOR VARYING NUMBER OF FUNCTIONS

Algorithm	FDWS	FDS_MIMF	ADS_MIMF
$\|MS\| = 100$	23067	22545	22918
$\|MS\| = 200$	24040	23361	25870
$\|MS\| = 300$	23865	21843	24066
$\|MS\| = 400$	31541	26078	33320
$\|MS\| = 500$	33391	26689	34725
$\|MS\| = 600$	40529	31105	40977
$\|MS\| = 700$	43405	33154	42208
$\|MS\| = 800$	47755	34255	45111

non-safety functions). Function samples are generated depending on the following realistic parameters: 100 μs $\leqslant w_{i,k} \leqslant$ 400 μs, 100 μs $\leqslant c_{i,j} \leqslant$ 400 μs, $8 \leqslant |N| \leqslant 23$. To meet the increasing complexity and requirement of ACPS, these experiments consider a maximum number of 800 functions running on 100 ECUs distributed on the CAN cluster.

B. Experimental Analysis

Experiment 1: This experiment is conducted to compare the overall makespan and DMRs on different scale function sets. Function samples are selected from the sample space. We limit the interval between the first and the last arrival functions to 10000 μs of each function set. The number of functions is changed from 100 to 800 to reflect the workload of ACPS. The criticality subscript of a function is calculated by $m\%4$, where m represents the mth function of ACPS. The deadline-slack of each function F_m is calculated as $F_m.deadlineslack = F_m.lowerbound/40$. Three algorithms (i.e., FDWS [8], FDS_MIMF, and ADS_MIMF) are used for the experiment and are then compared for verification. The FDWS algorithm is chosen because it is the latest typical multi-functional dynamic scheduling algorithms with the objective of minimizing individual makespans of functions for short over makespan of ACPS.

Table VIII shows the overall makespans for varying numbers of functions using the FDWS, FDS_MIMF, and ADS_MIMF algorithms. The FDS_MIMF algorithm exhibits a shorter makespan than the FDWS and ADS_MIMF algorithms in all cases. With the increase in the number of functions, the advantage of FDS_MIMF is apparent. For example, when $|MS| = 100$, the difference of the makespan between FDWS and FDS_MIMF is 1062 μs; however, when $|MS| = 800$, the difference reaches 13500 μs. Such results indicate that our proposed FDS_MIMF is effective in generating overall makespan from a high performance perspective.

Table IX shows the DMRs grouped by criticality levels (i.e., S_0, S_1, S_2, and S_3) for varying numbers of functions using the three algorithms. In general, FDS_MIMF generates considerably higher DMRs than FDWS and ADS_MIMF in all cases. When the number of functions reaches or exceeds 400 (i.e., $|MS| \geqslant 400$), the DMRs of all different criticality levels generated by FDS_MIMF are always 1. Although FDS_MIMF implements a low overall makespan for ACPS, this algorithm has the highest DMR. These results indicate that a short overall makespan of ACPS does not mean low DMRs; on the contrary, a high DMR exists in such a situation. In general, FDWS has lower DMRs than ADS_MIMF with criticality levels of S_0, S_1, and S_2 when the number of functions reaches or exceeds 400. However, ADS_MIMF has lower DMR than FDWS with the high-criticality level of S_3. Considering that missing the absolute deadlines of a function with criticality level S_3 would cause fatal injuries to people, the objective of ADS_MIMF is to reduce the DMR of high-criticality functions by scarifying the safety of partial low-criticality functions.

Meanwhile, as shown in Tables VIII, ADS_MIMF shows satisfactory performance compared with FDWS, especially on large-scale function sets (i.e., $MS = 700$ and $MS = 800$). According to the results of Tables VIII and IX, ADS_MIMF can significantly reduce the DMR of the functions with high-criticality level while maintaining satisfactory performance.

Experiment 2: Given that missing the deadlines of functions with S_3 will cause severity of life-threatening to fatal injuries and ACPS cannot meet the absolute deadlines of all functions with S_3 in Experiment 1, the number of such functions must be reduced. In this experiment, the total number of functions is fixed to 200. These functions are first evenly distributed to four criticality levels (S_0, S_1, S_2, and S_3), and then partial functions with criticality level S_3 are changed to S_0. The deadline-slack of each function F_m is still fixed as $F_m.deadlineslack = F_m.lowerbound/40$.

Table X shows the overall makespans for varying numbers of functions whose criticality levels are S_0 and S_3. The overall

TABLE IX
DMRs for Varying Numbers of Functions

Algorithm	FDWS				FDS_MIMF				ADS_MIMF			
Criticality	S_0	S_1	S_2	S_3	S_0	S_1	S_2	S_3	S_0	S_1	S_2	S_3
$\|MS\| = 100$	0.48	0.24	0.6	0.56	0.44	0.6	0.64	0.52	0.52	0.64	0.56	0.24
$\|MS\| = 200$	0.64	0.74	0.58	0.74	0.8	0.78	0.96	0.82	1.0	0.84	0.7	0.42
$\|MS\| = 300$	0.84	0.82	0.81	0.77	1.0	0.98	0.98	1.0	1.0	0.81	0.57	0.42
$\|MS\| = 400$	0.91	0.9	0.87	0.85	1.0	1.0	1.0	1.0	1.0	1.0	0.96	0.57
$\|MS\| = 500$	0.88	0.86	0.94	0.95	1.0	1.0	1.0	1.0	1.0	1.0	0.98	0.60
$\|MS\| = 600$	0.94	0.96	0.88	0.90	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.72
$\|MS\| = 700$	0.93	0.96	0.96	0.93	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.72
$\|MS\| = 800$	0.96	0.95	0.95	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.88

TABLE X
Overall Makespans (μs) for Varying Numbers of Functions Whose Criticality Levels are S_0 and S_3

Algorithms	FDWS	FDS_MIMF	ADS_MIMF
$\|MS(S_0)\| = 50$, $\|MS(S_1)\| = 50$, $\|MS(S_2) = \|50$, $\|MS(S_3)\| = 50$	21850	21332	21199
$\|MS(S_0)\| = 60$, $\|MS(S_1)\| = 50$, $\|MS(S_2) = \|50$, $\|MS(S_3)\| = 40$	21850	21332	21320
$\|MS(S_0)\| = 70$, $\|MS(S_1)\| = 50$, $\|MS(S_2) = \|50$, $\|MS(S_3)\| = 30$	21850	21332	21741
$\|MS(S_0)\| = 80$, $\|MS(S_1)\| = 50$, $\|MS(S_2) = \|50$, $\|MS(S_3)\| = 20$	21850	21332	20853
$\|MS(S_0)\| = 90$, $\|MS(S_1)\| = 50$, $\|MS(S_2) = \|50$, $\|MS(S_3)\| = 10$	21850	21332	21641

TABLE XI
DMRs for Varying Numbers of Functions With S_0 and S_3

Algorithm	FDWS				FDS_MIMF				ADS_MIMF			
Criticality	S_0	S_1	S_2	S_3	S_0	S_1	S_2	S_3	S_0	S_1	S_2	S_3
$\|MS(S_0)\| = 50$, $\|MS(S_1)\| = 50$, $\|MS(S_2)\| = 50$, $\|MS(S_3)\| = 50$	0.82	0.72	0.68	0.76	0.86	0.78	0.8	0.78	1.0	0.9	0.52	0.44
$\|MS(S_0)\| = 60$, $\|MS(S_1)\| = 50$, $\|MS(S_2)\| = 50$, $\|MS(S_3)\| = 40$	0.83	0.72	0.68	0.72	0.85	0.78	0.8	0.77	0.91	0.8	0.52	0.325
$\|MS(S_0)\| = 70$, $\|MS(S_1)\| = 50$, $\|MS(S_2)\| = 50$, $\|MS(S_3)\| = 30$	0.82	0.72	0.68	0.70	0.82	0.78	0.8	0.8	0.95	0.8	0.52	0.36
$\|MS(S_0)\| = 80$, $\|MS(S_1)\| = 50$, $\|MS(S_2)\| = 50$, $\|MS(S_3)\| = 20$	0.8	0.72	0.68	0.75	0.82	0.78	0.8	0.8	0.93	0.78	0.4	0.3
$\|MS(S_0)\| = 90$, $\|MS(S_1)\| = 50$, $\|MS(S_2)\| = 50$, $\|MS(S_3)\| = 10$	0.8	0.72	0.68	0.7	0.82	0.78	0.8	0.8	0.94	0.76	0.4	0.0

makespans generated by FDWS and FDS_MIMF are always fixed to 21850 and 21332 μs, respectively. The reason is that the total number of functions is not changed and the fairness-based strategy ignores the criticality levels. The overall makespans generated by ADS_MIMF are changed in the scope of 20853 μs and 21741 μs, and the differences are relatively stable.

Table XI shows the DMRs for varying numbers of functions with S_0 and S_3 using the three algorithms. The DMRs are high for FDWS and FDS_MIMF even when we reduce the number of the highest functions (S_3). Such results further indicate that FDWS and FDS_MIMF are not sensitive to the quantity of the highest functions if the total number of all functions is not changed. However, the DMR of functions with S_3 using ADS_MIMF is gradually reduced when we reduce the number of the highest functions. Particularly, when the number is reduced to 10, the DMR of functions with S_3 is 0. The DMR of functions with S_2 using ADS_MIMF are also reduced from 0.52 to 0.4. By this treatment, we implement the objective that ACPS meet the absolute deadlines of all the functions with S_3.

Experiment 3: Considering that scheduling tasks for minimum makespan is a well-known NP-hard optimization problem in dynamic multi-functional scheduling, it would be much more interesting to understand the quality of the generated schedules. In particular, it would be interesting to see a comparison to exact schedule results by exhausting all ECUs of each task to minimize the overall makespan of a small function set. In this study, we name the exact approach as the "exact FDS_MIMF" for FDS_MIMF. We limit the number of function set is fixed with 4, and the interval between the first and the last arrival functions to 1000 μs of the function set. The deadline-slack of each function F_m is still $F_m.deadlineslack = F_m.lowerbound/40$. The number of the tasks for each function is equal to 8. The CAN cluster contains 4 ECUs. We can derive that the number of the combination for the exact FDS_MIMF algorithm is $(4 \times 8)^4 = 1,048,576$.

TABLE XII
Overall Makespans (μs) of the Small Function Set

Algorithm	FDS_MIMF	ADS_MIMF	exact FDS_MIMF
$\|MS\| = 4$	2479.0	2213	2045

Tables XII and XIII show overall makespans and DMRs of the small function set using different algorithms. We can see

TABLE XIII
DMRs of the Small Function Set

Algorithm	FDS_MIMF				ADS_MIMF				exact FDS_MIMF			
Criticality	S_0	S_1	S_2	S_3	S_0	S_1	S_2	S_3	S_0	S_1	S_2	S_3
$\|MS\| = 4$	0.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0

from Table XII that the overall makespan of ACPS using the exact FDS_MIMF algorithm is about 82.5% and 92.4% of the overall makespan using the FDS_MIMF and ADS_MIMF algorithms, respectively. As shown in Table XIII, the total DMRs using the exact FDS_MIMF algorithm is similar to that using ADS_MIMF. However, ADS_MIMF satisfies the deadlines of two high-criticality (S_2 and S_3) functions, whereas the exact FDS_MIMF algorithm just satisfies the deadlines of functions with criticality levels of S_0 and S_2. FDS_MIMF merely satisfies the deadline of the lowest criticality (S_0) function. Such results demonstrate that the exact FDS_MIMF algorithm only reduce the overall makespan of ACPS and cannot ensure the reduction the DMRs of high-criticality functions.

Another approach to meet the absolute deadlines of all high-criticality (S_3) functions is to modify the deadline-slack or arrival interval. However, deadline-slack is determined by the relative deadline and the arrival interval between two functions is actually determined by the physical world. Therefore, both relative deadline and arrival interval cannot be changed in actual situation. Adding more ECUs in ACPS in the design phase is feasible but would be costly.

VII. Conclusions

We develop fairness-based and adaptive dynamic scheduling algorithms FDS_MIMF and ADS_MIMF on multi-functional mixed-criticality ACPS, respectively. Each distributed functions is described as a task graph with representation of a DAG. The FDS_MIMF algorithm aims to minimize individual makespans of functions with short overall makespan of ACPS from a high performance perspective. FDS_MIMF can respond autonomously to the joint challenges of heterogeneity, dynamics, and parallelism of ACPS. The ADS_MIMF algorithm aims to meet the absolute deadlines of more high-criticality functions whereas still keep satisfactory overall makespan of ACPS. ADS_MIMF can respond autonomously to the joint challenges of heterogeneity, dynamics, parallelism, safety, and criticality of ACPS. Experimental results indicate that both FDS_MIMF and ADS_MIMF are effective in individual objectives. We believe that the ADS_MIMF algorithm in this paper could provide a valuable reference design for adaptive scheduling in the next generation AUTOSAR adaptive platform. We will implement the proposed algorithms in the upcoming AUTOSAR adaptive platform and consider the evaluation of the real implementation as our future work.

Acknowledgment

The authors would like to express their gratitude to the anonymous reviewers whose constructive comments have helped to improve the manuscript.

References

[1] M. D. Natale and A. Sangiovanni-Vincentelli, "Moving from federated to integrated architectures in automotive: The role of standards, methods and tools," *Proc. IEEE*, vol. 98, no. 4, pp. 603–620, Mar. 2010.
[2] D. Goswami *et al.*, "Challenges in automotive cyber-physical systems design," in *Proc. IEEE Int. Conf. Embedded Comput. Syst.*, 2012, pp. 346–354.
[3] S. Fürst, "Challenges in the design of automotive software," in *Proc. Conf. Des., Autom. Test Eur.*, 2010, pp. 256–258.
[4] H. Zeng, M. Di Natale, P. Giusto, and A. Sangiovanni-Vincentelli , "Stochastic analysis of CAN-based real-time automotive systems," *IEEE Trans. Ind. Informat.*, vol. 5, no. 4, pp. 388–401, Sep. 2009.
[5] H. Zeng, M. D. Natale, A. Ghosal, and A. Sangiovanni-Vincentelli , "Schedule optimization of time-triggered systems communicating over the FlexRay static segment," *IEEE Trans. Ind. Informat.*, vol. 7, no. 1, pp. 1–17, Feb. 2011.
[6] Z. Yu and W. Shi, "A planner-guided scheduling strategy for multiple workflow applications," in *Proc. IEEE Int. Conf. Parallel Process. Workshops*, 2008, pp. 1–8.
[7] C.-C. Hsu, K.-C. Huang, and F.-J. Wang, "Online scheduling of workflow applications in grid environments," *Future Gener. Comp. Syst.*, vol. 27, no. 6, pp. 860–870, Jun. 2011.
[8] H. Arabnejad and J. Barbosa, "Fairness resource sharing for dynamic workflow scheduling on heterogeneous systems," in *Proc. IEEE 10th Int. Symp. Parallel Distrib. Process. Appl.*, 2012, pp. 633–639.
[9] A. Burns and R. Davis, "Mixed criticality systems—A review," Dept. Comput. Sci., Univ. York, York, U.K., Tech. Rep., pp. 1–64, 2016. [Online]. Available: http://www-users.cs.york.ac.uk/burns/review.pdf
[10] *Road Vehicles-Functional Safety, ISO 26262*, 2011.
[11] J. Nilsson, A. C. Ödblom, and J. Fredriksson, "Worst-case analysis of automotive collision avoidance systems," *IEEE Trans. Veh. Technol.*, vol. 65, no. 4, pp. 1899–1911, Apr. 2016.
[12] P. Kumar, D. Goswami, S. Chakraborty, A. Annaswamy, K. Lampka, and L. Thiele, "A hybrid approach to cyber-physical systems verification," in *Proc. 49th ACM/EDAC/IEEE Des. Autom. Conf.*, 2012, pp. 688–696.
[13] A. Wasicek, P. Derler, and E. A. Lee, "Aspect-oriented modeling of attacks in automotive cyber-physical systems," in *Proc. 51st ACM/EDAC/IEEE Des. Autom. Conf.*, 2014, pp. 1–6.
[14] S. Chakraborty, M. A. A. Faruque, W. Chang, and D. Goswami, "Automotive cyber-physical systems: A tutorial introduction," *IEEE Des. Test*, vol. 33, no. 4, pp. 92–108, May 2016.
[15] X. Chen, J. Feng, M. Hiller, and V. Lauer, "Application of software watchdog as a dependability software service for automotive safety relevant systems," in *Proc. 37th IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2007, pp. 618–624.
[16] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA, USA: MIT, 2011.
[17] M. Zeller, C. Prehofer, G. Weiss, D. Eilers, and R. Knorr, "Towards self-adaptation in real-time, networked systems: Efficient solving of system constraints for automotive embedded systems," in *Proc. 15th IEEE Int. Conf. Self-Adapt. Self-Organizing Syst.*, 2011, pp. 79–88.
[18] S. Dai and X. Koutsoukos, "Safety analysis of automotive control systems using multi-modal port-hamiltonian systems," in *Proc. 19th Int. Conf. Hybrid Syst., Comput. Control*, 2016, pp. 105–114.
[19] S. Fürst, "AUTOSAR the next generation—The adaptive platform," in *Proc. Conf., CARS@EDCC*, Paris, 8 Sep. 2015, 2015. [Online]. Available: http://conf.laas.fr/cars2015/CARS/CARS@EDCC2015_files/AUTOSAR_CARS@EDCC%202015.pdf
[20] S. Fürst, "AUTOSAR adaptive platform for connected and autonomous vehicles," in *Proc. conf., 8th Vector Congress, Alte Stuttgarter Reithalle*, Stuttgart, Germany, 29 Nov. 2016, 2016. [Online]. Available: https://vector.com/congress/files/presentations/VeCo16_06_29Nov_Reithalle_Fuerst_BMW.pdf

[21] S. Fürst and M. Bechter, "AUTOSAR for connected and autonomous vehicles: The AUTOSAR adaptive platform," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshop*, 2016, pp. 215–217.
[22] M. Bechter and M. Wille, "Future of AUTOSAR integrating heterogeneous platforms," 2015.
[23] S. Manolache, P. Eles, and Z. Peng, "Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 2, pp. 421–434, Feb. 2008.
[24] A. Biondi, M. Di Natale, and G. Buttazzo, "Response-time analysis for real-time tasks in engine control applications," in *Proc. ACM/IEEE 6th Int. Conf. Cyber-Phys. Syst.*, 2015, pp. 120–129.
[25] A. Biondi, M. Di Natale, and G. Buttazzo, "Performance-driven design of engine control tasks," in *Proc. ACM/IEEE 7th Int. Conf. Cyber-Phys. Syst.*, 2016, pp. 1–10.
[26] Z. Guo and S. K. Baruah, "Uniprocessor EDF scheduling of AVR task systems," in *Proc. ACM/IEEE 6th Int. Conf. Cyber-Phys. Syst.*, 2015, pp. 159–168.
[27] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proc. 28th IEEE Int. Real-Time Syst. Symp.*, 2007, pp. 239–243.
[28] K. Lakshmanan, D. De Niz, R. Rajkumar, and G. Moreno, "Overload provisioning in mixed-criticality cyber-physical systems," *ACM Trans. Embedded Comput. Syst.*, vol. 11, no. 4, pp. 1–24, Dec. 2012.
[29] R. Schneider, D. Goswami, A. Masrur, M. Becker, and S. Chakraborty, "Multi-layered scheduling of mixed-criticality cyber-physical systems," *J. Syst. Archit.*, vol. 59, no. 10, pp. 1215–1230, Nov. 2013.
[30] J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill, and C. Lu, "Mixed-criticality federated scheduling for parallel real-time tasks," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2016, pp. 1–12.
[31] S. Baruah, "The federated scheduling of systems of mixed-criticality sporadic DAG tasks," in *Proc. IEEE Real-Time Syst. Symp.*, 2016, pp. 1–10.
[32] J. D. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, Jun. 1975.
[33] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Aug. 2002.
[34] M. A. Khan, "Scheduling for heterogeneous systems using constrained critical paths," *Parallel Comput.*, vol. 38, no. 4, pp. 175–193, Apr. 2012.
[35] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 682–694, Mar. 2014.
[36] G. Xie, R. Li, and K. Li, "Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems," *J. Parallel Distrib. Comput.*, vol. 83, pp. 1–12, Sep. 2015.
[37] U. Hönig and W. Schiffmann, "A meta-algorithm for scheduling multiple dags in homogeneous system environments," in *Proc. 8th IASTED Int. Conf. Parallel Distrib. Comput. Syst.*, 2006, pp. 147–152.
[38] H. Zhao and R. Sakellariou, "Scheduling multiple dags onto heterogeneous systems," in *Proc. IEEE 20th Int. Parallel Distrib. Process. Symp.*, 2006, pp. 159–172.
[39] T. F. Abdelzaher and K. G. Shin, "Combined task and message scheduling in distributed real-time systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 11, pp. 1179–1191, Aug. 1999.
[40] B. P. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 7, no. 1, pp. 92–104, Aug. 2002.
[41] M. Hu, J. Luo, Y. Wang, and B. Veeravalli, "Scheduling periodic task graphs for safety-critical time-triggered avionic systems," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 51, no. 3, pp. 2294–2304, Sep. 2015.
[42] M. Hu, J. Luo, Y. Wang, M. Lukasiewycz, and Z. Zeng, "Holistic scheduling of real-time applications in time-triggered in-vehicle networks," *IEEE Trans. Ind. Informat.*, vol. 10, no. 3, pp. 1817–1828, May 2014.
[43] D. Tămaş-Selicean, P. Pop, and W. Steiner, "Design optimization of TTEthernet-based distributed real-time systems," *Real-Time Syst.*, vol. 51, no. 1, pp. 1–35, 2015.
[44] D. Tămaş-Selicean and P. Pop, "Design optimization of mixed-criticality real-time embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 14, no. 3, pp. 1–29, May 2015.
[45] G. Xie, G. Zeng, L. Liu, R. Li, and K. Li, "High performance real-time scheduling of multiple mixed-criticality functions in heterogeneous distributed embedded systems," *J. Syst. Archit.*, vol. 70, pp. 3–14, Oct. 2016.
[46] S. W. Kim, E. Lee, M. Choi, H. Jeong, and S. W. Seo, "Design optimization of vehicle control networks," *IEEE Trans. Veh. Technol.*, vol. 60, no. 7, pp. 3002–3016, Jul. 2011.
[47] S. Shreejith and S. A. Fahmy, "Extensible FlexRay communication controller for FPGA-based automotive systems," *IEEE Trans. Veh. Technol.*, vol. 64, no. 2, pp. 453–465, Feb. 2015.
[48] J. H. Kim, S. Seo, N. T. Hai, and B. M. Cheon, "Gateway framework for in-vehicle networks based on CAN, FlexRay, and Ethernet," *IEEE Trans. Veh. Technol.*, vol. 64, no. 10, pp. 4472–4486, Nov. 2015.
[49] M. Sojka, P. Písa, O. Spinka, and Z. Hanzálek, "Measurement automation and result processing in timing analysis of a Linux-based CAN-to-CAN gateway," in *Proc. IEEE 6th Int. Conf. Intell. Data Acquisition Adv. Comput. Syst.*, 2011, vol. 2, pp. 963–968.
[50] F. C. Braescu, L. Ferariu, and A. Nacu, "OSEK-based gateway algorithm for multi-domain can systems," in *Proc. IEEE Int. Conf. Intell. Comput. Commun. Process.*, 2011, pp. 423–428.
[51] N. Navet, S. Louvart, J. Villanueva, S. Campoy-Martinez, and J. Migge, "Timing verification of automotive communication architectures using quantile estimation," in *Proc. Eur. Congr. Embedded Real-Time Softw. Syst.*, 2014, pp. 1–10.
[52] G. Xie, G. Zeng, R. Kurachi, H. Takada, and R. Li, "Gateway modeling and response time analysis on CAN clusters of automobiles," in *Proc. IEEE 17th Int. Conf. High Perform. Comput. Commun.*, 2015, pp. 1147–1153.
[53] R. Leibinger, "Software architectures for advanced driver assistance systems (ADAS)," *Agenda: Short overview of Elektrobit automotive*, 2015.
[54] Y. Xie, G. Zeng, Y. Chen, R. Kurachi, H. Takada, and R. Li, "Worst case response time analysis for messages in controller area network with gateway," *IEICE Trans. Inf. Syst.*, vol. 96, no. 7, pp. 1467–1477, 2013.

Guoqi Xie (M'15) received the Ph.D. degree in computer science and engineering from Hunan University, Changsha, China, in 2014.

From 2014 to 2015, he was a Postdoctoral Researcher with Nagoya University, Nagoya, Japan. Since 2015, he has been a Postdoctoral Researcher with Hunan University. His research interests include automotive systems, embedded and real-time systems, parallel and distributed systems, software engineering, and methodology.

Dr. Xie is a member of ACM and CCF. He received the Best Paper Award from ISPA 2016.

Gang Zeng (M'03) received the Ph.D. degree in information science from Chiba University, Chiba, Japan, in 2006.

He is an Associate Professor with the Graduate School of Engineering, Nagoya University, Nagoya, Japan. From 2006 to 2010, he was a Researcher and then Assistant Professor with the Center for Embedded Computing Systems (NCES), Graduate School of Information Science, Nagoya University. His research interests include power-aware computing and real-time embedded system design.

Dr. Zeng is a member of IPSJ.

Zhetao Li received the Ph.D. degree in computer science from Hunan University, Changsha, China, in 2010.

He is an Associate Professor with Hunan University and Xiangtan University, Xiangtan, China. His research interests include Internet of Things, compressive sensing, and social computing.

Renfa Li (M'05–SM'10) received the Ph.D. degree in electronic engineering from Huazhong University of Science and Technology, Wuhan, China, in 2003. He is a Professor of computer science and electronic engineering and the Dean of College of Computer Science and Electronic Engineering, Hunan University, Changsha, China. He is the Director of the Key Laboratory for Embedded and Network Computing of Hunan Province, Changsha, China. His research interests include computer architectures, embedded computing systems, cyber-physical systems, and Internet of Things.

Prof. Li is a member of the council of CCF and a senior member of ACM.

Keqin Li (M'90–SM'96–F'15) received the Ph.D. degree in computer science from the University of Houston, Houston, TX, USA, in 1990. He is a SUNY Distinguished Professor of computer science. His research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU–GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of Things, and cyber-physical systems. He has published more than 460 journal articles, book chapters, and refereed conference papers, and has received several best paper awards.

Prof. Li is currently or has served on the editorial boards of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, theIEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON CLOUD COMPUTING, the IEEE TRANSACTIONS ON SERVICES COMPUTING, and the IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING.