Minimizing Development Cost With Reliability Goal for Automotive Functional Safety During Design Phase

Guoqi Xie, *Member, IEEE*, Yuekun Chen, Yan Liu, Renfa Li, *Senior Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—ISO 26262 is a functional safety standard specifically made for automotive systems, in which the automotive safety integrity level (ASIL) is the representation of the criticality level. Recently, most studies have used ASIL decomposition to reduce the development cost of automotive functions. However, these studies have not paid special attention to the problem that the reliability goal may not be satisfied when ASIL decomposition is performed. In this study, we solve the problem of minimizing the development cost of a distributed automotive function while satisfying its reliability goal during the design phase by presenting two heuristic algorithms, reliabilitycalculation of scheme (RCS) and minimizing development cost with reliability goal (MDCRG). We first use RCS to calculate the reliability value of each ASIL decomposition scheme; then, the MDCRG is used to select the scheme with the minimum development cost while satisfying the reliability goal. Real-life benchmark and simulated functions based on real parameter values are used in experiments, and results show the effectiveness of the proposed algorithms.**

***Index Terms*—Automotive functional safety, automotive safety integrity level (ASIL) decomposition, development cost, ISO 26262, reliability goal.**

NOMENCLATURE

Acronyms and Abbreviations

ASIL	Automotive safety integrity level.
DAG	Directed acyclic graph.
DAL	Design assurance level.
ECU	Electronic control unit.
MDCRG	Minimizing development cost with reliability goal.
RCS	Reliability calculation of scheme.
WCET	Worst case execution time.
WCRT	Worst case response time.

NOTATIONS

$\|X\|$	Size of the set X.
$w_{i,k,h}$	WCET of the task n_i on the ECU u_k and the ASIL L_h.
$c_{i,j}$	WCRT time of the message $m_{i,j.}$
$pred(n_i)$	Set of n_i's immediate predecessor tasks.
$succ(n_i)$	Set of n_i's immediate successor tasks.
λ_k	Failure rate per time unit of the ECU u_k.
$R(n_i, u_k, L_h)$	Reliability value of the task n_i on the ECU u_k and the ASIL L_h.
$DC(n_i, L_h)$	Reliability value of the task n_i on the ASIL L_h.
$R(n_i, scheme_g)$	Reliability value of the task n_i with the scheme $scheme_g$.
$R_{\text{goal}}(G)$	Reliability goal of the function G.
$R(G)$	Reliability of the function G.
$DC(G)$	Development cost of the function G.
$u_{pr(i)}$	Allocated ECU of the task n_i.
$DC_{\min}(n_i)$	Minimum development cost (MDC) of the task n_i.
$DC_{\max}(n_i)$	MDC of the task n_i.
$DC_{\min}(G)$	MDC of the function G.
$DC_{\max}(G)$	Maximum development cost of the function G.
$R_{\min}(n_i)$	Minimum reliability of the task n_i.
$R_{\max}(n_i)$	Maximum reliability of the task n_i.
$R_{\min}(G)$	Minimum reliability of the function G.
$R_{\max}(G)$	Maximum reliability of the function G.

Manuscript received November 13, 2016; revised July 8, 2017 and October 10, 2017; accepted November 24, 2017. Date of publication December 14, 2017; date of current version March 1, 2018. This work was supported in part by the National Key Research and Development Plan of China under Grant 2016YFB0200405, in part by the National Natural Science Foundation of China under Grant 61702172, Grant 61672217, Grant 61173036, Grant 61379115, Grant 61402170, Grant 61370097, Grant 61502162, and Grant 61502405, in part by the CCF-Venustech Open Research Fund under Grant CCF-VenustechRP2017012, in part by the CERNET Innovation Project under Grant NGII20161003, and in part by the China Postdoctoral Science Foundation under Grant 2016M592422. Associate Editor: Y. Le Traon. *(Corresponding author: Yan Liu.)*

G. Xie, Y. Chen, Y. Liu, 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; chenyuekun@126.com; liuyan@hnu.edu.cn; lirenfa@hnu.edu.cn).

K. Li is with the 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 at New Paltz, 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/TR.2017.2778070

I. INTRODUCTION

A. Background

SINCE the invention of the automobile, people have pursued the goal of safe driving. The earliest safety belts, later airbags, and other passive safety measures have saved millions

of lives. The later-developed antilock braking system and other active safety functions greatly enhance the safety of automobiles. However, hazardous events, including systemic failures and random hardware failures in automotive systems, caused by malfunctioning behavior may lead to risks [1]. Risk means the likelihood of injury or damage resulting from these failures and the severity of the consequences of injury or damage [1]. Examples include acceleration, deceleration, and steering; abnormal bounce of the airbag; and doors suddenly opening in high-speed driving. Functional safety is proposed to deal with the above-mentioned risks, and it refers to the absence of unreasonable risk due to hazards caused by malfunctioning behavior of electrical and electronic (E/E) systems [1]. The road vehicles functional safety standard ISO 26262 was officially released in 2011 to adapt to the functional safety of automotive E/E part [1]. ISO 26262 identifies four criticality levels denoted by automotive safety integrity levels (ASILs) (i.e., A, B, C, and D). ASIL refers to a classification of inherent safety goals used to ensure the accomplishment of the goals in the system. ASIL A and ASIL D represent the lowest and highest criticality levels, respectively. ISO 26262 requires designers to evaluate all potential risks in advance and take appropriate measures to maintain the risks in an acceptable scope to ensure the functional safety, particularly active safety functions. Therefore, the core task of modern automotive embedded system design is to ensure the safety-critical function of automotive embedded systems, that is, to maintain normal operations under all kinds of severe conditions and the functional safety of occupants and pedestrians.

B. Motivation

ISO 26262 defines the exposure for random hardware failures. Exposure means the relative expected frequency of the operational conditions, in which a hazardous event may happen and cause hazard and injury [1]. Exposure thus illustrates the possibility of the occurrence of harm and could affect the reliability of systems. Exposure involves five levels, namely E0, E1, E2, E3, and E4, where E0 represents the lowest level (i.e., incredibly unlikely), and E4 represents the highest level (i.e., high probability, injury could happen under most operating conditions) [1]. Reliability is defined as the probability that the execution is successful [2]–[5]. That is, reliability is just the reverse expression of exposure. If a function satisfies its reliability goal, then this function is considered to be reliable. However, reliability goal is not explicitly defined in ISO 26262, but it can be derived according to the probability values of exposures (please refer to Section III for more details).

The development of safety-critical functions is a highly structured and systematic process dictated by ISO 26262, thereby development cost is generated. Generally, development cost refers to the labor used in the development life cycle of systems, and it is different from resource cost and hardware cost. Increasing development cost is not an option because the automotive industry is a highly cost-sensitive industry for the mass market. ASIL decomposition means that a high-ASIL task can be decomposed into one or more redundant low-ASIL tasks [1], [6], [7]. ISO 26262, Part 9, Section V, provides the guide shown in Fig. 1 for

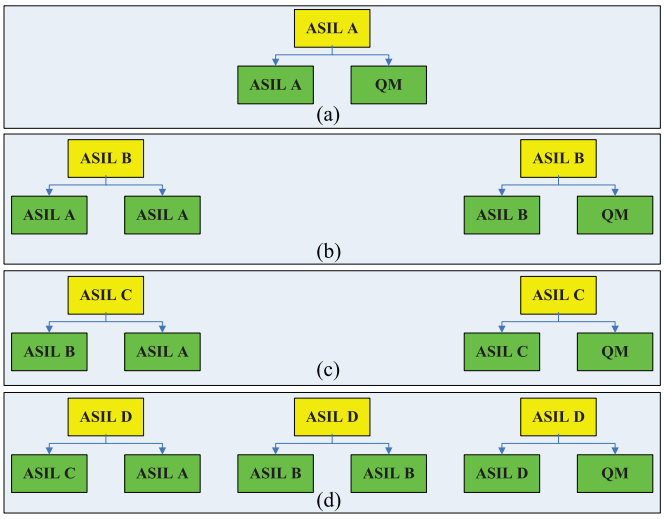


Fig. 1. ASIL decomposition schemes in ISO 26262 [1].

ASIL decomposition [1]. However, ASIL decomposition will affect the reliability and development cost simultaneously.

On the one hand, decomposing a high-ASIL task into several redundant low-ASIL tasks can enhance reliability due to redundancy, such that the reliability goal is easy to be satisfied.

On the other hand, decomposing a high-ASIL task to a low ASIL can reduce the development cost by 25%–100% because of the low requirement for low level in the implementation process [6], [7]. However, decomposition is implemented at the expense of adding redundant tasks, such that the total development cost of the redundant low-ASIL tasks is not necessarily less than that of the original high-ASIL task in this situation.

C. Our Contributions

A development life cycle of a safety-critical function usually involves the analysis, design, implementation, and testing phases. In this study, we aim to minimize the development cost of a distributed automotive function with a reliability goal during the design phase. The main contributions of this study are listed as follows.

1) We present a heuristic algorithm called reliability calculation of scheme (RCS) to calculate the reliability value and development cost of each ASIL decomposition scheme for each task with a low time complexity.
2) We present a heuristic algorithm called minimizing development cost with reliability goal (MDCRG) to select the scheme with the minimum development cost (MDC) while satisfying the reliability goal of the function.
3) We use a real-life benchmark and simulated functions based on real parameter values in experiments and validate the effectiveness of the proposed RCS and MDCRG algorithms.

The rest of this paper is organized as follows. Section II reviews related studies. Section III builds related models. Section IV explains automotive functional safety. Section V presents the RCS algorithm. Section VI presents the MDCRG algorithm. Section VII validates the performance of the presented algorithms. Section VIII concludes this study.

II. Related Work

The primary functional safety standards that are currently in use are the DO178B standard for avionic systems [8], the IEC 61508 standard for all kinds of industrial systems [9], and the ISO 26262 standard for automotive systems [1]. The criticality levels are represented as design assurance level (DAL), safety integrity level (SIL), and ASIL in avionic, industrial, and automotive systems, respectively. Many studies involve these functional safety standards, and the readers can be directed to [10] for a comparison of these standards. In this study, we mainly review related studies on development cost, criticality level decomposition, and the reliability goal.

Software development cost estimation is a widely researched topic. The reader is directed to [11] and [12] for reviews on this topic. One of the most influential software cost models is the constructive cost model [12]. Researchers have shown the consideration of development costs during the design phase of embedded systems [13]. The problem of deciding on the ASIL of a function is typically a manual process performed after hazard analysis and risk assessment. Researchers have started to propose automatic approaches to this problem [14]. A broader view is taken in [15], which proposes a method for the propagation, transformation, and refinement of functional safety requirements. Some studies on SIL decomposition [16], [17] and DAL decomposition [18] are solved to reduce development cost. For example, in [16] and [17], genetic and search-based metaheuristic algorithms for SIL decomposition are proposed; in [18], a tool called DALculus for the automatic allocation of DALs is presented, such that the smallest DAL may be allocated to a function. The ASIL decomposition for distributed functions was studied recently [6], [7].

As provided in ISO 26262, random hardware failures (also called transient failures in most studies) occur unpredictably during the life of a hardware element, but they follow a probability distribution [1]. A widely accepted reliability model was presented by Shatz and Wang [19], in which the transient failure of each hardware is characterized by a constant failure rate per time unit λ. The failure occurrence of a software task executed on hardware follows a constant-parameter Poisson's law because the hardware failure probability is extended to overall software [2]–[5], [20].

A function model for the description of distributed end-to-end computations in automobiles is represented by a directed acyclic graph (DAG) [6], [7], [21], [22]. Examples of active safety functions are brake-by-wire and adaptive cruise control [21], [23]. Scheduling tasks with quality of service (QoS) requirement for optimality on multiprocessors is known to be an NP-hard optimization problem [7], [24], [25]. In [26] and [27], Benoit *et al.* proved that evaluating the reliability of a DAG-based distributed function is an NP-complete problem. In the early development phase, reliability growth becomes critical in supporting decision-making for the overall development program, and [28] proposed a multiobjective, multistage reliability growth planning method. Reliability-aware design techniques and algorithms usually aim to minimize certain objectives while still satisfying the reliability goal. Higher reliability could result in a longer schedule length (or larger energy consumption) of a distributed function, and the problem of optimizing schedule length (or energy consumption) and reliability is considered to be a typical bicriteria optima or Pareto optima problem [29]–[32]. Energy-efficient scheduling with a reliability goal for a function with independent tasks has been studied extensively in [2], [33], and [34]. In [4], a shared recovery-based frequency allocation technique to minimize energy consumption with a reliability goal for a distributed function on a uniprocessor is proposed. All the aforementioned studies were only interested in schedule length or energy consumption minimization with a reliability goal. The recent investigation presented the MaxRe [20] and the RR (i.e., least resources to meet the reliability requirement) [5] algorithms to minimize resource consumption while satisfying the reliability goal of a distributed function on heterogeneous systems. In [35], the DAG_Heu algorithm is presented to minimize the resource cost for a distributed function with a timing constraint and a reliability goal on heterogeneous multiprocessors. However, MaxRe, RR, and DAG_Heu aim at minimizing resource consumption cost, which refers to the resource usage of processors when tasks are running, whereas development cost refers to the labor used in the development life cycle of systems. Therefore, the resource consumption cost and the development cost are two completely different concepts. Generally, development cost is reduced by ASIL decomposition in automotive systems [6], [7] and this is the research object of this paper.

In summary, we know that search algorithms have been developed to solve the problem of development cost reduction. This paper attempts to study this problem with another method. We would like to provide a new design reference for system designers and developers. We know that scheduling tasks with QoS requirement for optimality on multiprocessors is known to be an NP-hard optimization problem. Heuristic list scheduling is a popular method to schedule a DAG-based function in most works. To the best of our knowledge, no research work has employed heuristic list scheduling algorithms to study the problem of development cost reduction with reliability goal for a DAG-based function. In this study, we try to present a heuristic list scheduling to study this problem.

III. Automotive Functional Safety

In this section, we introduce some preliminaries of automotive functional safety that are related to ASIL decomposition and reliability goal.

A. Functional Safety Attributes

ISO 26262 defines two important functional attributes, namely severity and exposure. The standard also defines controllability, which represents the current state of the drivers. Severity, exposure, and controllability are mutually orthotropic. Table I shows the classifications of severity, exposure, and controllability in ISO 26262, where exposure is related to reliability.

The ASIL determination is based on these three attributes (severity, exposure, and controllability). Table II shows the ASIL

TABLE I
CLASSIFICATIONS OF SEVERITY, EXPOSURE, AND CONTROLLABILITY IN ISO 26262 [1]

Severity		Exposure		Controllability	
S0	No injuries	E0	Incredibly unlikely	C0	Controllable in general
S1	Light to moderate injuries	E1	Very low probability	C1	Simply controllable
S2	Severe to life-threatening injuries	E2	Low probability	C2	Normally controllable
S3	Life-threatening to fatal injuries	E3	Medium probability	C3	Difficult to control or uncontrollable
		E4	High probability		

TABLE II
ASIL DETERMINATION BASED ON SEVERITY, EXPOSURE, AND CONTROLLABILITY IN ISO 26262 [1]

Severity	Exposure	Controllability		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

determinations, where D is the highest level, and A is the lowest level. QM is the quality management, and it means that developing a function according to QM is sufficient, without regard to any safety-related design. Note that E0 and C0 do not constitute ASIL because E0 stands for incredibly unlikely (0%), and C0 stands for controllable in general (100%). In general, severity cannot be changed. The reason is that severity is associated with the consequences of a function when risk occurs and has been determined in hazard analysis and during the risk assessment phase. Controllability can be changed because it is decided by the current state of drivers; considering that this study focused on the design phase, we assume that the function is uncontrollable and it is denoted by C3 from a conservative perspective. In this study, if we want to reduce the ASIL of a function, then we need to reduce the exposure.

B. ASIL Decomposition

ISO 26262, Part 9, Section V provides the ASIL decomposition guide and scheme shown in Fig. 1 [1]. The decomposition affects the total development cost and reliability (i.e., successful probability) because a task has different worst case execution times (WCETs) and development costs on different ASILs. We consider a task n_i with ASIL C. According to Fig. 1(c), we can decompose n_i into two redundant tasks, namely n_i^1 with ASIL B and n_i^2 with ASIL A. n_i^1 can be further decomposed into two ASIL A tasks according to Fig. 1(b). As all the WCETs and development costs of tasks must be certified to the highest criticality level ASIL D, we assume that all tasks of the function

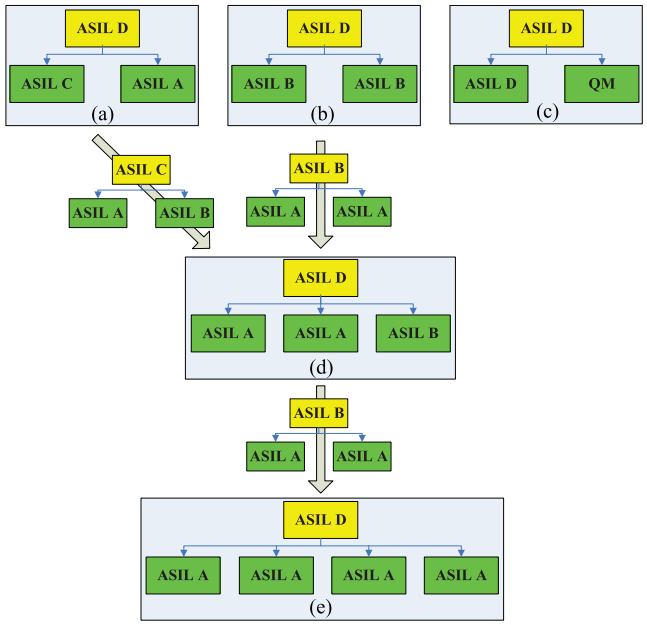


Fig. 2. Five ASIL decomposition schemes of ASIL D.

are first executed on ASIL D, and then each task is replicated to redundant tasks that can be executed in other ASILs through ASIL decomposition. We can obtain all possible schemes of ASIL D as follows.

1) ASIL D itself has three decomposition schemes, ASIL C + ASIL A, ASIL B + ASIL B, and ASIL D, as shown in Fig. 1(d).
2) ASIL C can be decomposed into ASIL B + ASIL A according to Fig. 1(c), thus ASIL C + ASIL A can be further decomposed to ASIL B + 2 × ASIL A, as shown with the arrow from Fig. 2(a) to (d). Similarly, ASIL B + ASIL B can also be further decomposed to ASIL B + 2 × ASIL A, as shown with the arrow from Fig. 2(b) to (d). Therefore, these two decompositions can only be considered as the same result, as shown in Fig. 2(d).
3) ASIL B can be decomposed into ASIL A + ASIL A according to Fig. 1(b), such that ASIL B + 2 × ASIL A can be further decomposed to 4 × ASIL A, as shown with the arrow from Fig. 2(d) to (e).

Finally, five ASIL decomposition schemes of ASIL D are obtained as follows: ASIL C + ASIL A, ASIL B + ASIL B, ASIL D, ASIL B + 2 × ASIL A, and 4 × ASIL A, as shown with the arrows from Fig. 2(a) to (e).

Our objective is to choose the best scheme for each task.

TABLE III
CLASSES OF PROBABILITY OF EXPOSURE REGARDING DURATION/PROBABILITY OF EXPOSURE IN ISO 26262 [1]

	Exposure level	Probability of exposure	Reliability goal
E1	Very low probability	Not specified	At least exceeds 0.99
E2	Low probability	$<1\%$	0.99
E3	Medium probability	[1%, 10%]	>0.9
E4	High probability	$>10\%$	$<=0.9$

C. Exposure and Reliability Goal

ISO 26262 provides the duration/probability of exposure in [1, Table B.2, Annex B of Part 3], as shown in Table III. For example, the probability of exposure E4 is larger than 10% of average operating time. Note that ISO 26262 does not define the concept of reliability goal, but we can deduce the corresponding reliability goals for given exposure levels as mentioned earlier. For example, the probability of exposure E2 is less than 1% of average operating time. That is, the lowest probability of a occurrence hazardous event is close to 0.01; to ensure safety, the actual reliability must be larger than or equal to 1–0.01 = 0.99, which is considered as the reliability goal in this case. The reliability goals for other exposures can also be obtained according to the above rule. Finally, the reliability goals for exposures are shown in Table III.

IV. MODELS

A. System Architecture

Automotive E/E architecture has evolved into heterogeneous distributed integrated architecture because of the size, weight, and power consumption for cost and high-performance benefits [21], [23], [36]. A high-end automotive system comprises at least 70 heterogeneous electronic control units (ECUs) interconnected by a central gateway, and the number of ECUs is expected to increase further in future automotive systems [21], [37].

Similar to [21], we consider a distributed integrated platform as a controller area network (CAN) cluster (also called multidomain 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. Each ECU contains a central processing unit (CPU), random access memory and nonvolatile memory, and a network interface card [7]. An ECU can execute several tasks with different ASILs and a task can be executed on different ECUs. A task executed completely in one ECU sends messages to all its successor tasks, which may be located in the same ECU or different ECUs of different buses. For example, task n_1 is executed on ECU u_1 of CAN_1. It then sends a message $m_{1,2}$ to its successor task n_2 located in u_6 of CAN_3 (see Fig. 3). The central gateway is a highly important node that connects CAN clusters and allows messages to be passed from one bus to another. $U = \{u_1, u_2, \ldots, u_{|U|}\}$ represents a set of heterogeneous ECUs, where $|U|$ represents the size of set U. Note that for any set X, this study uses $|X|$ to denote its size.

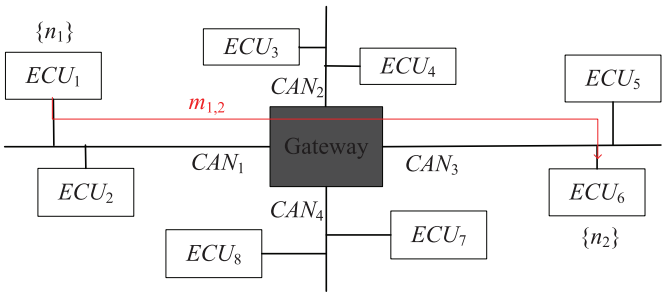


Fig. 3. Integrated automotive architecture [21].

B. Function Model

Integrated architecture leads to an increase in distributed functions with precedence-constrained tasks (e.g., brake-by-wires and adaptive cruise control), which can be distributed over multiple ECUs [21], [23]. A distributed function is represented by a DAG $G = (N, W, M, C, V)$ [5], [20], [21], [37], [38]. The tasks, messages, and development costs are described as follows.

1) N represents a set of tasks in G, and $n_i \in N$ represents the ith task of G. $pred(n_i)$ represents the set of the immediate predecessor tasks of n_i, whereas $succ(n_i)$ represents the set of the immediate successor tasks of n_i. The task with no predecessor task is denoted by n_{entry}, whereas the task with no successor task is denoted by 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. Each task $n_i \in N$ has different WCETs on different ECUs even in the same ASIL. A task with high ASIL has larger WCET than with low ASIL on the same ECU. This is a key aspect of mixed-criticality system because a higher level of assurance is required and this property significantly modifies/undermines many of the standard scheduling results [39].
2) W is a $4 \times |N| \times |U|$ cube, where $w_{i,k,h}$ denotes the WCET of n_i on the ECU u_k and the ASIL L_h. All the WCETs of the tasks are known through the analysis methods performed during the analysis phase [40]. All the WCET values will be taken as input to optimize the development cost during the design phase.
3) The communication between tasks mapped to different ECUs is performed through 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.
4) Accordingly, $c_{i,j} \in C$ represents a worst case response time (WCRT) of $m_{i,j}$ [37]. All the WCRTs of the messages are also known through the analysis methods performed during the analysis phase [41].
5) V is a $4 \times |N|$ matrix, where $v_{i,h}$ represents the development cost of n_i on the ASIL L_h. Each task has different development costs under different criticality levels. A task with high ASIL has larger development cost than that with low ASIL [6], [7]. The reason is that a task developed at a high ASIL will pay more effort to ensure functional safety

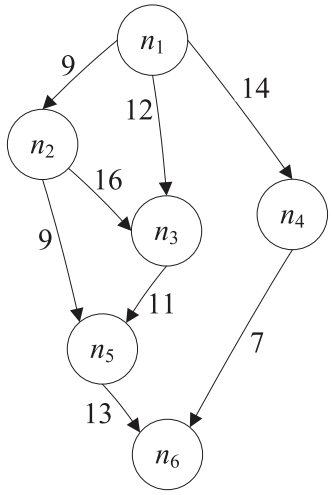


Fig. 4. Motivating example of a DAG-based distributed function.

than that at a low ASIL. Similar to [6], [7], we assume that the development effort for each task on each ASIL has been known because of the systematic nature of the development processes dictated by the standards. In this study, we consider nonpreemptive scheduling for ECUs which can be supported by eCos operating system.

C. Motivating Example

Fig. 4 shows a motivating example of a DAG-based distributed function with six tasks. Weight 9 of the edge between n_1 and n_2 in Fig. 4 represents the WCRT denoted by $c_{1,2} = 9$ if n_1 and n_2 are not allocated to the same ECU.

Table IV lists the WCETs of each task on four different ECUs $\{u_1, u_2, u_3, u_4\}$ and four different ASILs $\{L_\mathrm{A}, L_\mathrm{B}, L_\mathrm{C}, L_\mathrm{D}\}$. Weight 7 of n_1, u_2, and L_A in Table IV represents the WCET denoted by $w_{1,2,\mathrm{A}} = 7$. A task with high ASIL has larger WCET than that with low ASIL on the same ECU as explained earlier. A task with the same ASIL has different WCETs on different ECUs because of the heterogeneity of the ECUs.

Table V lists the development costs of each task on four different ASILs $\{L_\mathrm{A}, L_\mathrm{B}, L_\mathrm{C}, L_\mathrm{D}\}$. Weight 5 of n_1, L_A in Table V represents the development cost denoted by $v_{1,\mathrm{A}} = 5$. A task with high ASIL has larger development cost than that with low ASIL as explained earlier.

D. Problem Description

We assume that a distributed function G is given with a known reliability goal $R_{\mathrm{goal}}(G)$ that would be executed on a heterogeneous multiple ECUs set U and ASIL set $\{L_\mathrm{A}, L_\mathrm{B}, L_\mathrm{C}, L_\mathrm{D}\}$. Then, the problem to be addressed in this study is to allocate an ECU and ASIL for each task replica while reducing the development cost of G and satisfying its reliability goal $R_{\mathrm{goal}}(G)$. The formal description is finding the ECU and ASIL allocations of all the tasks to minimize the development cost of the function G:

$$DC(G) = \sum_{n_i \in N} DC\left(n_i\right) \tag{1}$$

subject to

$$R(G) = \prod_{n_i \in N} R\left(n_i, u_{pr(i)}\right) \geqslant R_{\mathrm{goal}}(G) \tag{2}$$

where $R(G)$ represents the actual reliability value and will be explained in detail in Section V-A. $DC(n_i)$ represents the development cost of n_i.

V. Reliability Calculation of Schemes

As the decomposition of ASIL D has five fixed schemes shown in Fig. 2, the work of this study is to select the scheme and corresponding ECUs that minimizes the development cost while satisfying the reliability goal of the function. Therefore, the first problem of this study is how to calculate possible reliability value of schemes.

A. Reliability and Task Replication

Generally, the occurrence of a transient failure for a task in a distributed function follows a Poisson's distribution [2]–[5], [19], [20], [26]. The reliability of an event in unit time t is denoted by

$$R\left(t\right) = e^{-\lambda t}$$

where λ is the *constant failure rate per time unit* for an ECU. We use λ_k to represent the constant failure rate per time unit of the ECU u_k. The reliability of n_i executed on u_k and L_h in its execution time is denoted by

$$R\left(n_i, u_k, L_h\right) = e^{-\lambda_k w_{i,k,h}} \tag{3}$$

and the exposure for n_i without using redundant replication is

$$E(n_i, u_k, L_h) = 1 - R\left(n_i, u_k, L_h\right) = 1 - e^{-\lambda_k w_{i,k,h}}. \tag{4}$$

This study assumes that the function is implemented as software tasks running on a distributed architecture. Because each task has a certain number of replicas (i.e., redundancy) after ASIL decomposition, and each scheme has a fixed number of replicas, we define $num(scheme_g)$ as the number of replicas of $scheme_g$. As long as one replica of n_i is successfully completed, n_i is reliable. Therefore, the reliability value of n_i with allocated scheme $scheme_g$ is calculated by

$$R(n_i, scheme_g) = 1 - \prod_{x=1}^{num(scheme_g)} \left(1 - R(n_i^x, u_{pr(n_i^x)}, L_{cl(pr(n_i^x))})\right) \tag{5}$$

where n_i^x represents the xth number of replicas of n_i. $u_{pr(n_i^x)}$ and $L_{cl(n_i^x)}$ represent the allocated ECU and ASIL of n_i^x, respectively. Then, the reliability of the distributed function is the product of the reliability values of all tasks and is calculated by

$$R(\mathrm{G}) = \prod_{n_i \in N} R(n_i) = \prod_{n_i \in N} R(n_i, scheme_{sc(n_i)}) \tag{6}$$

TABLE IV
WCETs (UNIT: MS) OF TASKS ON DIFFERENT ECUs AND ASILs

	n_1				n_2				n_3				n_4				n_5				n_6			
	u_1	u_2	u_3	u_4	u_1	u_2	u_3	u_4	u_1	u_2	u_3	u_4	u_1	u_2	u_3	u_4	u_1	u_2	u_3	u_4	u_1	u_2	u_3	u_4
L_{A}	4	7	5	8	10	9	4	7	5	8	13	6	14	11	10	6	14	12	16	8	1	6	3	8
L_{B}	6	9	7	10	12	11	6	9	7	10	14	8	16	13	12	8	16	14	18	10	3	8	5	10
L_{C}	8	11	9	12	14	13	8	11	9	12	16	10	18	15	14	10	18	16	20	12	5	10	7	12
L_{D}	10	13	11	14	16	15	10	13	11	14	18	12	20	17	16	12	20	18	22	14	7	12	9	14

TABLE V
DEVELOPMENT COSTS (UNIT: EUROS IN THOUSANDS) OF TASKS ON DIFFERENT ASILs

	n_1	n_2	n_3	n_4	n_5	n_6
L_{A}	5	7	5	4	5	8
L_{B}	8	12	8	7	9	13
L_{C}	12	17	11	11	14	18
L_{D}	16	22	14	15	18	22

TABLE VI
FAILURE RATES OF ECUs $\{u_1, u_2, u_3, u_4\}$

Parameter	u_1	u_2	u_3	u_4
λ_k	0.01	0.02	0.03	0.04

where $scheme_{sc(n_i)}$ represents the allocated scheme of n_i. Particularly, this study only involves the ECU failure and does not include the communication failure in the problem (i.e., the communication is assumed to be reliable in this study).

B. Reliability Calculation

In the previous analysis, to obtain the reliability value $R(G)$ of the function, we should first obtain the allocated scheme of n_i to calculate the reliability of each task (5). We first list an example of calculating the reliability value of each task on a given scheme. The failure rates of four ECUs are shown in Table VI.

ASIL D is decomposed into one ASIL C and one ASIL A in $scheme_1$ [see Fig. 1(a)]. Thus, n_1 with $scheme_1$ may be executed on the following ECUs and ASILs:

$$\begin{cases} R(n_1, u_1, L_{\mathrm{C}}) = 0.92311635 \\ R(n_1, u_2, L_{\mathrm{C}}) = 0.80251880 \\ R(n_1, u_3, L_{\mathrm{C}}) = 0.76337949 \\ R(n_1, u_4, L_{\mathrm{C}}) = 0.61878340 \\ R(n_1, u_1, L_{\mathrm{A}}) = 0.96078944 \\ R(n_1, u_2, L_{\mathrm{A}}) = 0.86935824 \\ R(n_1, u_3, L_{\mathrm{A}}) = 0.86070798 \\ R(n_1, u_4, L_{\mathrm{A}}) = 0.72614904. \end{cases} \tag{7}$$

Similarly a task has different WCETs on different ECUs or different ASILs, and the failure rates vary in different ECUs. Therefore, the reliability values are different in (7). As the number of replicas is two in $scheme_1$, we only need to select two reliability values among eight candidates of (7). To obtain a high reliability value for the scheme, we use the following steps to select reliability values for n_1 with $scheme_1$.

1) We select $R(n_1, u_1, L_{\mathrm{A}}) = 0.96078944$ because it has the maximum reliability value among eight candidates.
2) We then select the second maximum reliability value for L_{C}, and $R(n_1, u_1, L_{\mathrm{C}}) = 0.92311635$ can be selected. However, u_1 has been occupied by Step 1). We should select the third maximum reliability value $R(n_1, u_2, L_{\mathrm{C}}) = 0.80251880$, where u_2 has not been occupied, given the parallel execution of the distributed function.

Finally, the reliability of n_1 with $scheme_1$ is calculated by (5) as follows:

$$\begin{aligned} &R(n_1, scheme_1) \\ &= 1 - (1 - R(n_1, u_1, L_{\mathrm{A}}))\,(1 - R(n_1, u_2, L_{\mathrm{C}})) \\ &= 1 - (1 - 0.96078944)(1 - 0.80251880) = 0.99225665. \end{aligned}$$

C. RCS Algorithm

We presented a heuristic RCS algorithm, as shown in Algorithm 1, to calculate the reliability of each scheme based on the above-mentioned analysis.

The details of the RCS algorithm are explained as follows.

1) In Line 1, RCS obtains the decomposed ASIL map $map(scheme_g)$ $(< L_\alpha, num_\alpha >, < L_\beta, num_\beta >, \ldots, < L_\gamma, num_\gamma >)$ of $scheme_g$.
2) In Lines 2–6, RCS calculates $R(n_i, u_k, L_h)$ using (4), which is similar to the calculation of (7).
3) In Line 7, RCS sorts the reliability values in a list $descending\_reliability\_list$ by descending order of $R(n_i, u_k, L_h)$ values.
4) In Lines 8–22, RCS calculates $R(n_i, scheme_g)$ based on the heuristic idea explained in Section V-B to obtain a high reliability value for the scheme as much as possible.

The time complexity of the RCS algorithm is analyzed as follows.

1) The reliability of each task on each ECU and decomposed ASIL should be calculated in $O(\,|U|)$ time (Lines 2–6).
2) The $descending\_reliability\_list$ should be sorted in $O(\,|N| \times \log\,|N|)$ time (Line 7).
3) Calculating $R(n_i, scheme_g)$ should traverse all reliability values and verify the occupied ECUs, which can be performed in $O(|U| \times \log\,|U|)$ time (Lines 8–21).

Algorithm 1: RCS Algorithm.

Input: $U = \{u_1, u_2, \ldots, u_{|U|}\}$, $\{L_{\mathrm{A}}, L_{\mathrm{B}}, L_{\mathrm{C}}, \ldots, L_{\mathrm{D}}\}$, n_i and related values
Output: $R(n_i, scheme_g)$ and related values
1: Obtain the decomposed ASIL map $map(scheme_g)$ $(<L_\alpha, num_\alpha>, <L_\beta, num_\beta>, \ldots, <L_\gamma, num_\gamma>)$ of $scheme_g$, where num_α represents the number of L_α;
2: **for** (each ASIL $L_h \in map(scheme_g)$ **do**
3: **for** (each ECU $u_k \in U$) **do**
4: Calculate $R(n_i, u_k, L_h)$ using (4);
5: **end for**
6: **end for**
7: Sort the reliability values in a list $descending\_reliability\_list$ by descending order of $R(n_i, u_k, L_h)$ values.
8: **while** (true) **do**
9: $R(n_i, u_k, L_h) \leftarrow descending\_reliability\_list.out()$;
10: **if** (u_k has been occupied by any replica of n_i) **then**
11: **continue;**
12: **end if**
13: **if** (the number of times to select L_h reaches num_h) **then**
14: **continue;**
15: **end if**
16: **if** (the number of times to select any L_h reaches num_h) **then**
17: **break;**
18: **end if**
19: $u_{pr(n_i^x)} \leftarrow u_k$;
20: $L_{cl(n_i^x)} \leftarrow L_h$;
21: **end while**
22: Calculate $R(n_i, scheme_g)$ using (5);

Considering Line 7 occupies the main time, the time complexity of the RCS algorithm is $O(\,|N| \times \log |N|)$.

VI. Minimizing Development Cost With Reliability Goal

As the reliability value of each scheme for each task has been obtained by using the RCS algorithm in the previous section, we can solve the final problem of MDCRG on heterogeneous automotive systems. However, scheduling tasks with QoS requirement for optimality on multiprocessors is known to be an NP-hard optimization problem. List scheduling is a well-known heuristic method with low time complexity for a DAG-based distributed function [21], [37], [38], and it includes two phases. The first phase orders tasks based on the descending order of priorities (task prioritization), whereas the second phase allocates each task to the appropriate ECU (task allocation). Most works use search algorithms to find the decomposition schemes [6], [7], [16], [17]. The limitation of search algorithms is that they are time-consuming. Considering that the automotive industry is cost-sensitive, shortening the function's development life cycle to reduce development cost is crucial. Therefore, using heuristic list scheduling method with low time complexity to solve the problem is more suitable than search algorithms from a development progress control perspective. In this study, we also use the list scheduling to solve the subject problem by task prioritization and task allocation. Task prioritization takes the existing method explained in Section VI-A, whereas task allocation is decomposed into two subproblems: satisfying the reliability goal and minimizing the development cost, explained in Sections VI-B and VI-C, respectively.

TABLE VII
Upward Rank Values of the Tasks of the Distributed Function in Fig. 4 [38]

Task	n_1	n_2	n_3	n_4	n_5	n_6
$rank_u(n_i)$	117	96	67	38	42	10

A. Task Prioritization

Similar to [38], [21], and [42], this study uses the upward rank value ($rank_u$) of a task given by (8) as the common task priority standard. We use the highest level ASIL D to determine the task priority according to the ISO 26262 standard because a task has different WCETs on different ASILs. Therefore, all the tasks are ordered according to the decreasing order of $rank_u$:

$$rank_u(n_i) = \overline{w_{i,\mathrm{D}}} + \max_{n_j \in succ(n_i)} \{c_{i,j} + rank_u(n_j)\} \quad (8)$$

where $c_{i,j}$ represents the WCRT of the message $c_{i,j}$ as mentioned earlier. $\overline{w_{i,\mathrm{D}}}$ represents the average WCET of task n_i on ASIL D and is calculated by

$$\overline{w_{i,\mathrm{D}}} = \left(\sum_{k=1}^{|U|} w_{i,k,\mathrm{D}} \right) / |U|.$$

Table VII shows the upward rank values of all the tasks (see Fig. 4). If all the predecessors of n_i have been allocated to the ECUs, then n_i is prepared to be allocated. We assume that two tasks n_i and n_j satisfy $rank_u(n_i) > rank_u(n_j)$. If no precedence constraint exists between n_i and n_j, then n_i may not have higher priority than n_j. Finally, the task allocation order in G is $\{n_1, n_2, n_3, n_5, n_4, n_6\}$.

B. Satisfying Reliability Goal

The minimum and maximum reliability values can be obtained by traversing all the schemes because the reliability of each task on each scheme can be obtained through Algorithm 1. Both values are calculated by

$$R_{\min}(n_i) = \min_{g \in [1,5]} R(n_i, scheme_g) \quad (9)$$

and

$$R_{\max}(n_i) = \max_{g \in [1,5]} R(n_i, scheme_g) \quad (10)$$

respectively.

The minimum and maximum reliability values of G are calculated by

$$R_{\min}(G) = \prod_{n_i \in N} R_{\min}(n_i) \tag{11}$$

and

$$R_{\max}(G) = \prod_{n_i \in N} R_{\max}(n_i) \tag{12}$$

because the reliability of function G is the product of the reliability values of all the tasks (6).

As mentioned earlier, the reliability goal $R_{\text{goal}}(G)$ should be satisfied during ASIL decomposition. Note that $R_{\text{goal}}(G)$ should be larger than or equal to $R_{\min}(G)$; otherwise, $R_{\text{goal}}(G)$ is always satisfied. Meanwhile, $R_{\text{goal}}(G)$ should be less than or equal to $R_{\max}(G)$; otherwise, $R_{\text{goal}}(G)$ cannot always be satisfied. Hence, this study assumes that $R_{\text{goal}}(G)$ belongs to the scope $R_{\min}(G)$ and $R_{\max}(G)$, namely

$$R_{\min}(G) \leqslant R_{\text{goal}}(G) \leqslant R_{\max}(G). \tag{13}$$

In this study, we can obtain the $R_{\min}(G) = 0.43171052$ and $R_{\max}(G) = 0.99452100$ of the motivating distributed function. We assume that the reliability goal is $R_{\text{goal}}(G) = 0.9$.

The strategy of satisfying the reliability goal of the distributed function G is shown as follows. We assume that the task to be allocated is $n_{seq(j)}$, where $seq(j)$ represents the jth allocated sequence. Therefore, $\{n_{seq(1)}, n_{seq(2)}, \ldots, n_{seq(j-1)}\}$ represents the set of tasks that have been allocated, and $\{n_{seq(j+1)}, n_{seq(j+2)}, \ldots, n_{seq(|N|)}\}$ represents the set of tasks that have not been allocated. To ensure the reliability of the function at each task allocation, we presuppose that each task in $\{n_{seq(j+1)}, n_{seq(j+2)}, \ldots, n_{seq(|N|)}\}$ is allocated to the scheme with the maximum reliability value. Hence, when allocating $n_{seq(j)}$, the reliability of G is calculated by

$$R(G) = \prod_{x=1}^{j-1} R\left(n_{seq(x)}, u_{pr(seq(x))}\right) \times R\left(n_{seq(j)}\right) \times \prod_{y=j+1}^{|N|} R_{\max}\left(n_{seq(y)}\right).$$

Given that $R(G)$ should be larger than or equal to $R_{\text{goal}}(G)$, we have

$$R(G) = \prod_{x=1}^{j-1} R\left(n_{seq(x)}, u_{pr(seq(x))}\right) \times R\left(n_{seq(j)}\right) \times \prod_{y=j+1}^{|N|} R_{\max}\left(n_{seq(y)}\right) \geq R_{\text{goal}}(G)$$

namely

$$R(n_{seq(j)}) \geq \frac{R_{\text{goal}}(G)}{\prod_{x=1}^{j-1} R(n_{seq(x)}) \times \prod_{y=j+1}^{|N|} R_{\max}(n_{seq(y)})}. \tag{14}$$

Hence, we let the reliability goal of task $n_{seq(y)}$ be

$$R_{\text{goal}}(n_{seq(j)}) = \frac{R_{\text{goal}}(G)}{\prod_{x=1}^{j-1} R(n_{seq(x)}) \times \prod_{y=j+1}^{|N|} R_{\max}(n_{seq(y)})}. \tag{15}$$

Given that the minimum reliability of $n_{seq(j)}$ is $R_{\min}(n_{seq(j)})$ [calculated by (9)], $R_{\text{goal}}(n_{seq(j)})$ should be updated to

$$R_{\text{goal}}(n_{seq(j)}) = \max\{R_{\text{goal}}(n_{seq(j)}), R_{\min}(n_{seq(j)})\}. \tag{16}$$

Given that the maximum reliability of $n_{seq(j)}$ is $R_{\max}(n_{seq(j)})$ [calculated by (10)], $R_{\text{goal}}(n_{seq(j)})$ should be further updated to

$$R_{\text{goal}}(n_{seq(j)}) = \min\{R_{\text{goal}}(n_{seq(j)}), R_{\max}(n_{seq(j)})\}. \tag{17}$$

Therefore, the reliability goal of the function can be transferred to each task. In other words, we only let $n_{seq(j)}$ satisfy the following constraint:

$$R(n_{seq(j)}) \geqslant R_{\text{goal}}(n_{seq(j)}).$$

Hence, when allocating task $n_{seq(j)}$, we can directly consider the reliability goal $R_{\text{goal}}(n_{seq(j)})$ of $n_{seq(j)}$ and ignore the reliability goal of function G. As a result, a low time complexity heuristic algorithm can be achieved.

C. Minimizing Development Cost

We let $DC(n_i, scheme_g)$ represent the development cost of n_i with scheme $scheme_g$. We can obtain the development cost of each scheme shown in Fig. 2, given that each scheme has fixed ASILs, shown as follows:

$$\begin{cases} DC(n_i, scheme_1) = v_{i,\text{C}} + v_{i,\text{A}} \\ DC(n_i, scheme_2) = 2v_{i,\text{B}} \\ DC(n_i, scheme_3) = v_{i,\text{D}} \\ DC(n_i, scheme_4) = 2v_{i,\text{A}} + v_{i,\text{B}} \\ DC(n_i, scheme_5) = 4v_{i,\text{A}} \end{cases} \tag{18}$$

where $v_{i,h}$ represents the development cost of n_i on ASIL L_h as mentioned earlier.

The minimum and maximum development costs of each scheme can be obtained by traversing all the five schemes. Both costs are calculated by

$$DC_{\min}(n_i) = \min_{g \in [1,5]} DC(n_i, scheme_g) \tag{19}$$

and

$$DC_{\max}(n_i) = \max_{g \in [1,5]} DC(n_i, scheme_g) \tag{20}$$

respectively.

We let $scheme_{sc(n_i)}$ represent the allocated scheme of n_i. Thus, the total development cost of the distributed function is

$$DC(G) = \sum_{n_i \in N} DC(n_i) = \sum_{n_i \in N} DC(n_i, scheme_{sc(n_i)}). \tag{21}$$

The total development cost of function G is the sum of the development costs of all the tasks. Thus, the minimum and

maximum development costs of G are calculated by

$$DC_{\min}(G) = \sum_{n_i \in N} DC_{\min}(n_i) \tag{22}$$

and

$$DC_{\max}(G) = \sum_{n_i \in N} DC_{\max}(n_i). \tag{23}$$

The obtained minimum development value is $DC_{\min}(G) = 106$, and its corresponding reliability value is 0.65648591. However, 0.65648591 cannot satisfy the reliability value of 0.9, which is the reliability goal of the motivating distributed function. Therefore, we should particularly consider that the reliability goal must be satisfied in ASIL decomposition.

In this study, we can obtain the $R_{\min}(G) = 0.43171052$ and $R_{\max}(G) = 0.99452100$ of the motivating distributed function. We assume that the reliability goal is $R_{\text{goal}}(G) = 0.9$.

In the following paragraphs, the algorithm called MDCRG is presented to reduce the development cost of a distributed function while satisfying its reliability goal, as shown in Algorithm 2.

The core idea of MDCRG is that the reliability goal of the function is transferred to each task. Each task only selects the scheme with the MDC while satisfying its reliability goal. The main details are explained as follows.

1) MDCRG obtains the reliability goal of each task before it prepares the tasks to be allocated (Line 9).
2) MDCRG skips the schemes that do not satisfy the reliability goal (Lines 12–14); that is, it does not need to calculate the total reliability of the function and determine whether it satisfies the given reliability goal in each task allocation by traversing all the tasks.
3) MDCRG selects the scheme with the MDC for each task while satisfying the condition $R(n_i, scheme_g) < R_{\text{goal}}(n_i)$ (Lines 15–19).
4) If two schemes have the same development cost, then the scheme with the higher reliability value is selected (Lines 20–24).

The time complexity of the MDCRG algorithm is analyzed as follows. Scheduling all tasks must involve traversing all tasks, which can be done in $O(|N|)$ time. The RCS algorithm is called to calculate the reliability value of each scheme in $O(|N| \times \log |N|)$. Therefore, the time complexity of the MDCRG algorithm is $O(|N|^2 \times \log |N|)$. Thus, MDCRG implements effective development cost minimization without sacrificing time complexity.

Algorithm 2: The MDCRG Algorithm.

Input: $U = \{u_1, u_2, \ldots, u_{|U|}\}$, $\{L_{\text{A}}, L_{\text{B}}, L_{\text{C}}, \ldots, L_{\text{D}}\}$, G, $R_{\text{goal}}(G)$
Output: $R(G)$, $DC(G)$ and related values

1: Sort the tasks in a list $descending\_task\_list$ by descending order of $rank_u(n_i)$ values using (8);
2: **while** (there are tasks in $descending\_task\_list$) **do**
3: $n_i \leftarrow descending\_task\_list.out()$;
4: **for** ($g \leftarrow 1; g \leq 5; g$++) **do**
5: Calculate $R(n_i, scheme_g)$ using the RCS algorithm;
6: Calculate $DC(n_i, scheme_g)$ using (18)
7: **end for**
8: Calculate $R_{\min}(n_i)$ and $R_{\max}(n_i)$ using (9) and (10), respectively;
9: Calculate $R_{\text{goal}}(n_i)$ using (16); //1) calculate the reliability goal of n_i before it prepares to be allocated.
10: $sc(n_i) \leftarrow 0$, $DC(n_i) \leftarrow \infty$, $R(n_i) \leftarrow 0$;
11: **for** ($g \leftarrow 1; g \leq 5; g$++) **do**
12: **if** ($R(n_i, scheme_g) < R_{\text{goal}}(n_i)$) **then**
13: continue; //2) skip the schemes that do not satisfy the reliability goal of n_i.
14: **end if**
15: **if** ($DC(n_i, scheme_g) < DC(n_i)$) **then**
16: $sc(n_i) \leftarrow g$;
17: $R(n_i) \leftarrow R(n_i, scheme_{sc(n_i)})$;
18: $DC(n_i) \leftarrow DC(n_i, scheme_{sc(n_i)})$;
19: **end if//3) select the scheme with the minimum development cost** $DC(n_i, scheme_{sc(n_i)})$.
20: **if** ($DC(n_i, scheme_g) == DC(n_i)$&& $R(n_i, scheme_g) > R(n_i)$) **then**
21: $sc(n_i) \leftarrow g$;
22: $R(n_i) \leftarrow R(n_i, scheme_{sc(n_i)})$;
23: $DC(n_i) \leftarrow DC(n_i, scheme_{sc(n_i)})$;
24: **end if//4) if two schemes have the same development cost, then select the scheme with the higher reliability value** $R(n_i, scheme_{sc(n_i)})$.
25: **end for**
26: **end while**
27: Calculate the actual reliability $R(G)$ using (2);
28: Calculate the final development cost $DC(G)$ using (1).

D. Example of the MDCRG Algorithm

The process and results of the motivating example using the MDCRG algorithm are illustrated in this section. We assume that the constant failure rates for four ECUs are still shown in Table VI and that the reliability goal is still $R_{\text{seq}}(G) = 0.9$. Table VIII lists scheme allocation of the motivating distributed function using the MDCRG algorithm. Each row shows the selected scheme (denoted by bold text) and the corresponding reliability value and development cost. For example, MDCRG selects scheme $scheme_2$ because it has the MDC of 16 in satisfying the reliability goal of 0.90483742. We note that $scheme_3$ also has the MDC of 16, and its reliability value can also satisfy its reliability goal. However, if two schemes have the same development cost, then the scheme with the higher reliability value is selected. The advantage of such strategy is that the reliability goals of the remaining tasks can be reduced according to (15). Therefore, the actual reliability value and the final development cost of n_1 are $R(n_1) = 0.99040688$ and $DC(n_1) = 16$. All the remaining tasks use the same pattern with n_1. Finally, the actual reliability value is $R(G) = 0.918901856$, and the final development cost of function G is $DC(G) = 115$, which are calculated by (2) and (1), respectively.

TABLE VIII
SCHEME ALLOCATION OF THE MOTIVATING DISTRIBUTED FUNCTION USING THE MDCRG ALGORITHM

n_i	$R_{\text{goal}}(n_i)$	$R(n_i, scheme_1)\&$ $\&DC(n_i, scheme_1)$	$R(n_i, scheme_2)\&$ $DC(n_i, scheme_2)$	$R(n_i, scheme_3)\&$ $DC(n_i, scheme_3)$	$R(n_i, scheme_4)\&$ $DC(n_i, scheme_4)$	$R(n_i, scheme_5)\&$ $DC(n_i, scheme_5)$
n_1	0.90483742	0.9922566517	**0.9904068816**	0.9048374216	0.9990297118	0.9998046020
n_2	0.91314970	0.9796949624	**0.9813724324**	0.8521437922	0.9978749226	0.9995670928
n_3	0.93161239	**0.9895937216**	0.9877450816	0.8958341414	0.9980252518	0.9995031120
n_5	0.93866938	0.9642236319	**0.9638911118**	0.8187307518	0.9908100719	0.9970899120
n_4	0.97244400	0.9661400315	0.9661485514	0.8187307515	**0.9929348415**	0.9985732416
n_6	0.97933739	**0.9981963426**	0.995883326	0.9323938222	0.9998733829	0.9999734832
$R(G) = 0.918901856, DC(G) = 115$						

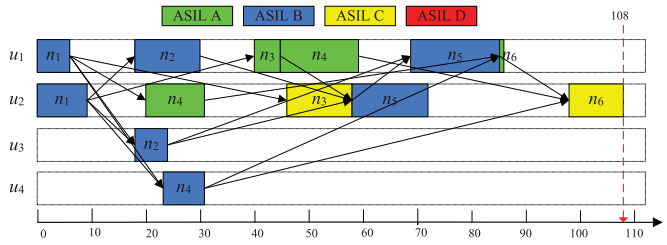


Fig. 5. Task mapping of the motivating distributed function.

We observe that the final reliability is $R(G) = 0.918901856 > R_{\text{goal}}(G) = 0.9$. Thus, the reliability goal of the function has been satisfied. The MDC of the function is $DC_{\min}(G) =$ €106 k [calculated by (22)] in Section VI-C, which is €9 k less than that using MDCRG. However, the corresponding reliability value for $DC_{\min}(G) =$ €106 k is merely $R(G) = 0.65648591$.

Fig. 5 also shows the task mapping of the motivating distributed function G using MDCRG. The tasks with different colors mean that they are executed on corresponding ASILs, where ASIL A, ASIL B, ASIL C, and ASIL D are denoted by green, blue, yellow, and red, respectively. For example, n_1 selects the $scheme_2$ containing two ASIL B executed on two ECU u_1 and u_2. The reason is that $scheme_2$ has the MDC of 16 while satisfying n_1's reliability goal of 0.90483742, shown in Table VIII. Finally, the end-to-end response time is 108 ms. Note that the arrows in Fig. 5 represent generated communications between tasks.

VII. EXPERIMENTS

A. Experimental Metrics

Considering that this study aims to minimize the development cost of a distributed automotive function with a reliability goal, performance metrics selected for comparison should be the actual reliability value and final development cost of the function. Meanwhile, computation time (i.e., the time to find a solution) should be included from a development life cycle perspective. The final development cost $DC(G)$ and the actual reliability value $R(G)$ are calculated by (1) and (2), respectively. The computation time is measured from the start time to the end time of an algorithm to find a solution.

Algorithms compared with the proposed MDCRG algorithms are the MDC and genetic algorithm with reliability goal (GARG) algorithms. MDC is obtained by (22) and is optimal because it does not consider a reliability goal or timing constraint. GARG is a genetic algorithm to search an enough optimal solution. Genetic algorithm has been developed to solve the problem of development cost reduction for DAG-based distributed functions. Therefore, we believe that MDC and GARG are suitable as compared algorithms in this study.

Considering that this study focuses on the design phase, the function parameters used in this phase are known based on real deployment. In other words, these values have been obtained in the analysis phase. We use the parameter values of real automotive systems as experimental data. The parameter values of the function used in this study are as follows. The failure rate of each task falls in the range of 10^{-5}–10^{-4} in the time unit of 1 μs. The WCETs of the tasks and the WCRTs of the messages fall under the range of 100–1600 μs. The development cost of each task falls in the range of €0–€30 k. The aforementioned values are generated with uniform distribution.

The distributed functions will be tested on a simulated system based on the above real function parameter values to reflect a real deployment. A main advantage of simulation is that it can greatly reduce life cycle cost during the design phase and effectively provide certain optimization guide to the implementation phase. The simulated system is configured with 16 heterogeneous ECUs by creating 16 ECU objects based on the known parameter values using Java on a standard desktop computer with 2.6-GHz Intel CPU and 4-GB memory.

Note that the values of experimental results are obtained by executing one run for one function. Many tests with the same parameter values and scales are preformed and show the same regular pattern and relatively stable results. In other words, experiments are repeatable and do not affect the consistency of the results. Considering that GARG is a genetic algorithm and is a randomized method, the experiment needs to be run multiple times and use statistical tests for the analysis of results. We did several experiments using the GARG algorithm, we recorded that GARG is also relatively stable to the results produced by MDC and MDCRG. Therefore, GARG-generated values are also obtained by executing one run for one function.

B. Real-Life Benchmark

We use the real-life benchmark of an automotive case study shown in Fig. 6 adopted from [6]. This function consists of six function blocks: engine controller with seven tasks (n_1–n_7), automatic gear box with four tasks (n_8–n_{11}), antilocking brake system with six tasks (n_{12}–n_{17}), wheel angle sensor with

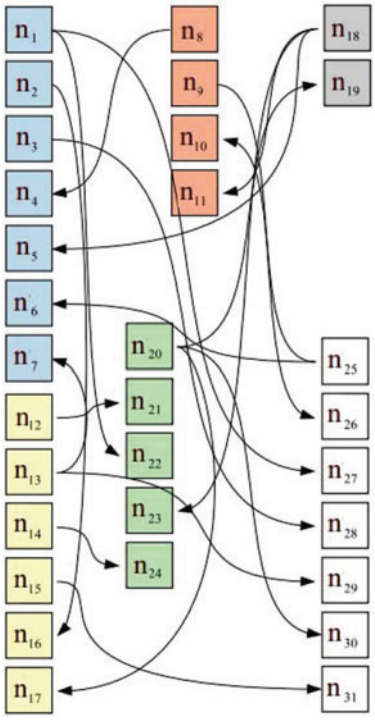


Fig. 6. Benchmark of real-time automotive function from [6].

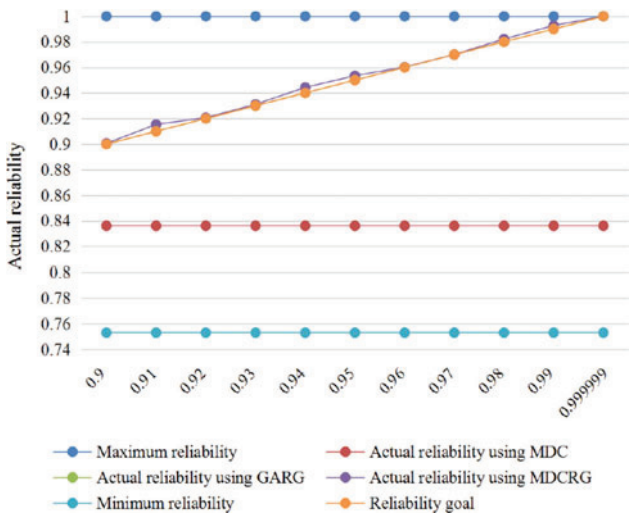


Fig. 7. Actual reliability values of the real-life function for varying reliability goals.

two tasks (n_{18}–n_{19}), suspension controller with five tasks (n_{20}–n_{24}), and body work with seven tasks (n_{25}–n_{31}).

Experiment 1: This experiment is conducted to compare the actual reliability values and the final development costs of a real-life function for varying reliability goals. The reliability goal is changed from 0.9 to 0.99 with a 0.01 increment because values of reliability goals fall in the range of exposure E3 and E2 (see Table III). Meanwhile, the maximum reliability goal for the function is set as 0.999999, which belongs to the reliability goal in E1.

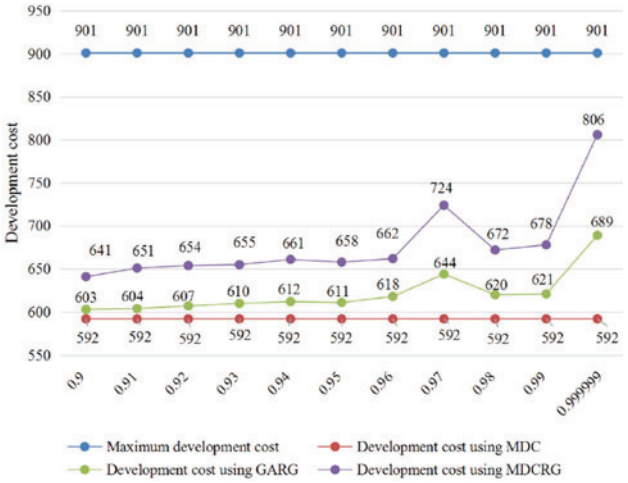


Fig. 8. Development costs (unit: Euros in thousands) of the real-life function for varying reliability goals.

Fig. 7 shows the actual reliability values of the real-life function for varying reliability goals. The following observations are found.

1) The minimum and maximum reliability values calculated by (11) and (12) are 0.753075 and 0.999999999, respectively.
2) In all the cases, the actual reliability values using the MDCRG and GARG algorithm can always satisfy and are very close to the corresponding reliability goals. The maximum difference between actual reliability values and reliability goals are merely 0.006 and 0.0001 for MDCRG and GARG, respectively.
3) In all the cases, the actual reliability values using the MDC algorithm are approximately 0.836166, which is larger than the minimum reliability value of 0.753075, but they cannot satisfy individual reliability goals.

The above-mentioned results validate that MDC is not designed to satisfy the reliability goals of functions in practice. By contrast, MDCRG and GARG can always satisfy the reliability goals, and the actual reliability values are close to the reliability goals without excessive waste.

Fig. 8 shows the development costs of the real-life function for varying reliability goals, and the following observations are made.

1) The minimum and maximum development costs calculated by (22) and (23) are €592 k and €901 k, respectively. Thus, the development costs using MDC is €592 k.
2) The development costs obtained by MDCRG are not increased linearly with increased reliability goal. The minimum and maximum development costs are €641 k ($R_{\text{goal}} = 0.9$) and €724 k ($R_{\text{goal}} = 0.97$), respectively. The reason could be explained as follows: There are five fixed ASIL decomposition schemes and these schemes could add more redundant tasks for some reliability goals. However, we can select the optimal reliability goal that has MDC in a given interval. For example, some automakers may expect a reliability goal of 0.97, but the actual selected reliability goal will be 0.98 because it generates the

lowest development cost as long as the reliability goal is not less than 0.97.

3) The development costs using MDCRG are higher than the MDCs using MDC. The reason is that MDCRG needs to satisfy its reliability goal and skip some schemes that cannot satisfy the reliability goal of the functions in the task allocation (see Lines 12–19 of Algorithm 2). However, the development costs using MDCRG are still close to the MDCs and are distant from the maximum development costs.
4) The development costs using GARG are always between those using MDC and MDCRG. The reason is that GARG can find the exact MDCs while satisfying given reliability goals, whereas MDCRG is a heuristic list scheduling algorithm to obtain approximate MDCs. The results show that GARG can save as much as 14.5% of development cost than MDCRG.
5) Although development costs are increasing with reliability goals overall, they do not increase linearly. For example, we can easily observe that when the reliability goal is 0.97, the second maximum development cost with €724 k and €644 k increases for MDCRG and GARG, respectively. Such results indicate that larger reliability goals do not lead to lower development costs using MDCRG.
6) Besides some interesting relations between development cost and reliability goal are derived from the experiment results, it would be to assess the efficiency of the MDCRG and GARG algorithms, namely how much time they take to find the results. Our results show that MDCRG merely needs less than 1 s to find the results in all cases. However, GARG needs at least 19 h to find the development cost in each case. As pointed out in Section II, shortening the function's development life cycle to reduce development cost is crucial during the design phase because automotive industry is cost-sensitive. Therefore, using heuristic list scheduling method is more suitable than genetic algorithm from a development progress control perspective.

C. Simulated Functions With Real Parameter Values

To further validate the effectiveness of the proposed MDCRG algorithm, we use additional simulated functions with the same real parameter values of the real-life function to observe the results.

Experiment 2: The increasing complexity of automotive systems will likely lead to future automotive functions to include at least 50 tasks and may reach 100 tasks. This experiment shows the development costs of functions for varying reliability goals, which are changed from 0.9 to 0.98 with 0.01 increments, because exposure E3 is primarily used in actual system design. The reliability goals can always be satisfied in all cases in Experiment 1. Hence, the reliability values of the functions are no longer provided. Considering that GARG is very time-consuming and more tasks will take more running time, we no longer provide the running time values using GARG.

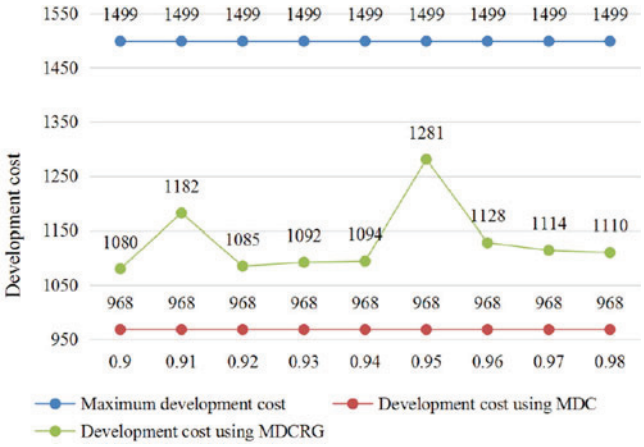


Fig. 9. Development costs (unit: Euros in thousands) of simulated function with 500 tasks for varying reliability goals.

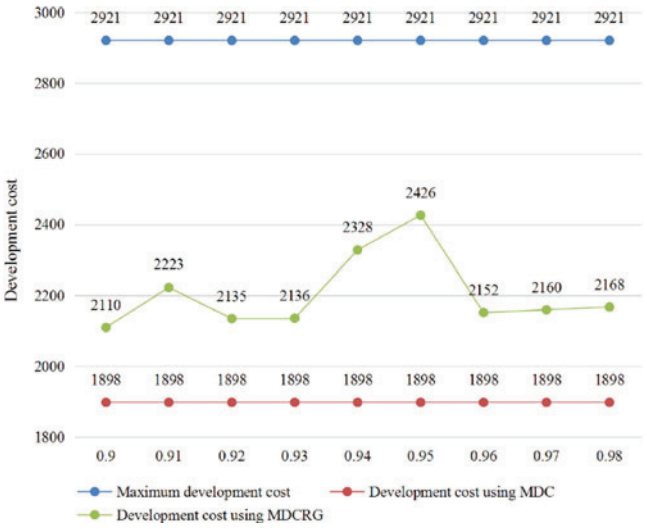


Fig. 10. Development costs (unit: Euros in thousands) of simulated function with 100 tasks for varying reliability goals.

Figs. 9 and 10 show the development costs of functions with 50 tasks and 100 tasks, respectively, for varying reliability goals. The following observations are made.

1) The minimum and maximum development costs for the function with 50 tasks are €968 k and €1499 k, respectively (see Fig. 9), whereas those for the function with 100 tasks are €1898 k and €2921 k (see Fig. 10). Thus, more tasks need more development costs, which are approximately linearly increased.
2) The development costs for the function with 50 tasks using MDCRG fall under the range of €1080 k and €1281 k (see Fig. 9), whereas those for the function with 100 tasks fall under the range of €1898 k and €2921 k (see Fig. 10). Such results further indicate that the development costs are approximately linearly increased with the increment of tasks.
3) Similar to Experiment 1, the development costs for functions with 50 tasks and 100 tasks using MDCRG still

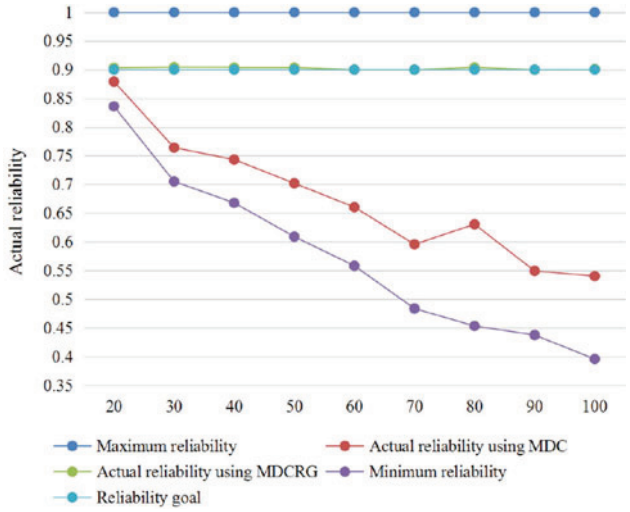


Fig. 11. Actual reliability values of the real-life function for varying numbers of tasks.

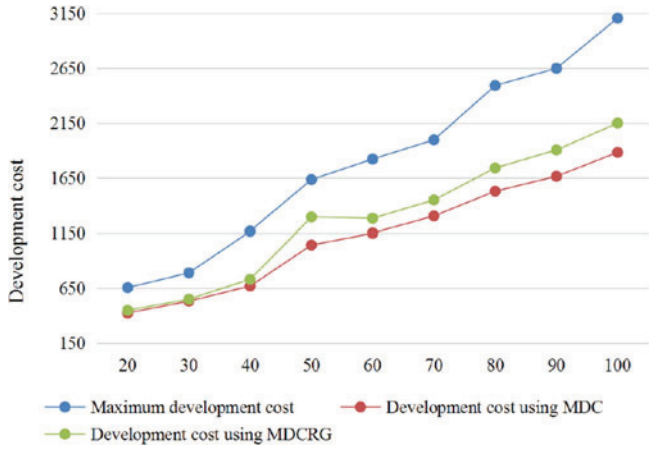


Fig. 12. Development costs (unit: Euros in thousands) of the real-life function for varying numbers of tasks.

do not increase linearly with the increment of reliability goals. When the reliability goal is 0.95, both functions need maximum reliability costs. The minimum reliability cost occurs when the reliability goal is 0.9.

Experiment 3: We are interested in obtaining the results for varying number of tasks of functions with a reliability goal fixed at 0.9, given that MDCRG usually obtains minimum developments cost when the reliability goal is 0.9, which is the least reliability goal of E3. In this experiment, the number of tasks is changed from 20 to 100 with ten increments.

Figs. 11 and 12 show the actual reliability values and development costs of functions for varying number of tasks.

1) Fig. 11 shows that the minimum reliability values and the reliability values using MDC are reduced with the increment of the numbers of tasks. The reason is that the reliability of a function is the product of all its tasks.
2) We find that the reliability values using MDC cannot satisfy the reliability goal of 0.9 in all the cases, whereas those using MDCRG can still always satisfy and are very close to the reliability goal in all the cases. The maximum difference between the actual reliability values and the reliability goals is merely 0.004.
3) All the algorithms would generate increased development costs with the increment of the number of tasks, as shown in Fig. 12, because more tasks could lead to higher development costs. Fortunately, the development costs using MDCRG are still close to the MDCs and are distant from the maximum development costs in all cases, even when the number of tasks is increased.

In summary, combined with the results of real-life and simulated functions, the proposed MDCRG algorithm is effective in minimizing development costs while still satisfying corresponding reliability goals. We believe that MDCRG can effectively explore a part of the design space of development cost during the design phase in the development life cycle of a safety-critical function. Programmers can implement the function according to the ASIL decomposition scheme of each task generated by MDCRG in the later implementation phase.

VIII. Conclusion

This study develops an effective development cost minimization solution for a distributed automotive function with a reliability goal on heterogeneous automotive systems by using ASIL decomposition during the design phase. This study presented two heuristic algorithms, RCS and MDCRG, to solve the problem. RCS calculates the reliability value of each ASIL decomposition scheme, and MDCRG transfers the reliability goal of the function to each task and selects the ASIL decomposition scheme with the MDC, while satisfying the reliability goal of the task. MDCRG is validated with real-life and simulated distributed automotive functions in various situations. Considering that the timing constraint is also an important functional safety requirement in real-time embedded systems and has been studied by the state-of-the-art studies in minimizing development cost, our future studies will simultaneously consider the timing constraint and reliability goal to minimize the development cost for distributed automotive functions on heterogeneous architecture. It will be feasible to consider timing requirement and reliability goal one by one in the design phase, or to consider them simultaneously to conduct biobjective optimization.

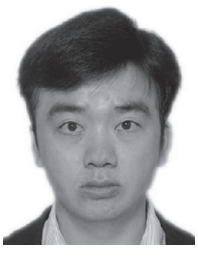
Acknowledgment

The authors would like to express their gratitude to the associate editor and three anonymous reviewers for their constructive comments, which have helped to improve the quality of this paper.

References

[1] *Road Vehicles—Functional Safety—Part 1: Vocabulary*, ISO Standard 26262–1, 2011.
[2] D. Zhu and H. Aydin, "Reliability-aware energy management for periodic real-time tasks," *IEEE Trans. Comput.*, vol. 58, no. 10, pp. 1382–1397, Oct. 2009.

[3] B. Zhao, H. Aydin, and D. Zhu, "On maximizing reliability of real-time embedded applications under hard energy constraint," *IEEE Trans. Ind. Informat.*, vol. 6, no. 3, pp. 316–328, Aug. 2010.
[4] B. Zhao, H. Aydin, and D. Zhu, "Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 2, pp. 99–109, Mar. 2013.
[5] L. Zhao, Y. Ren, and K. Sakurai, "Reliable workflow scheduling with less resource redundancy," *Parallel Comput.*, vol. 39, no. 10, pp. 567–585, Jul. 2013.
[6] J. Gan, P. Pop, and J. Madsen, "Tradeoff analysis for dependable real-time embedded systems during the early design phases," Ph.D. dissertation, Dept. Inf. Math. Model., Tech. Univ. Denmark, Kongens Lyngby, Denmark, 2014.
[7] D. Tămaş-Selicean and P. Pop, "Design optimization of mixed-criticality real-time embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 14, no. 3, May 2015, Art. no. 50.
[8] E. Denney and G. Pai, "Automating the assembly of aviation safety cases," *IEEE Trans. Rel.*, vol. 63, no. 4, pp. 830–849, Dec. 2014.
[9] IEC, "IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems," *Int. Electrotech. Commission*, Geneva, Switzerland, 2010.
[10] J. Machrouh *et al.*, "Cross domain comparison of system assurance," in *Proc. Embedded Real Time Softw. Syst.*, Toulouse, France, 2012, pp. 1–3.
[11] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 33–53, Jan. 2007.
[12] B. Boehm, C. Abts, and S. Chulani, "Software development cost estimation approaches–A survey," *Ann. Softw. Eng.*, vol. 10, nos. 1–4, pp. 177–205, Nov. 2000.
[13] J. A. Debardelaben, V. K. Madisetti, and A. J. Gadient, "Incorporating cost modeling in embedded-system design," *IEEE Des. Test Comput.*, vol. 14, no. 3, pp. 24–35, Jul. 1997.
[14] Y. Papadopoulos *et al.*, "Automatic allocation of safety integrity levels," in *Proc. 1st Workshop Crit. Autom. Appl. Robustness Safety*, 2010, pp. 7–10.
[15] D. Sojer, C. Buckl, and A. Knoll, "Propagation, transformation and refinement of safety requirements," in *Proc. 3rd Workshop Non-Funct. Syst. Properties Domain Specific Model. Lang.*, 2010, pp. 1–15.
[16] D. Parker, M. Walker, L. S. Azevedo, Y. Papadopoulos, and R. E. Araújo, "Automatic decomposition and allocation of safety integrity levels using a penalty-based genetic algorithm," in *Proc. Int. Conf. Ind. Eng. Other Appl. Appl. Intell. Syst.*, 2013, pp. 449–459.
[17] L. S. Azevedo, D. Parker, M. Walker, Y. Papadopoulos, and R. E. Araujo, "Automatic decomposition of safety integrity levels: Optimization by tabu search," in *Proc. 2nd Workshop Crit. Autom. Appl. Robustness Safety 32nd Int. Conf. Comput. Safety Rel. Security*, 2013, pp. 1–6.
[18] P. Bieber, R. Delmas, and C. Seguin, "DALculus—Theory and tool for development assurance level allocation," in *Proc. Int. Conf. Comput. Safety Rel. Security*. 2011, pp. 43–56.
[19] S. M. Shatz and J. P. Wang, "Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems," *IEEE Trans. Rel.*, vol. 38, no. 1, pp. 16–27, Apr. 1989.
[20] L. Zhao, Y. Ren, Y. Xiang, and K. Sakurai, "Fault-tolerant scheduling with dynamic number of replicas in heterogeneous systems," in *Proc. 12th IEEE Int. Conf. High Performance Comput. Commun.*, 2010, pp. 434–441.
[21] 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.
[22] 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, Nov. 2009.
[23] M. Di Natale and A. L. 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, Apr. 2010.
[24] J. D. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, Jun. 1975.
[25] S. K. Baruah, "Task partitioning upon heterogeneous multiprocessor platforms," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2004, pp. 536–543.
[26] A. Benoit, L.-C. Canon, E. Jeannot, and Y. Robert, "Reliability of task graph schedules with transient and fail-stop failures: Complexity and algorithms," *J. Scheduling*, vol. 15, no. 5, pp. 615–627, Oct. 2012.
[27] A. Benoit, F. Dufossé, A. Girault, and Y. Robert, "Reliability and performance optimization of pipelined real-time systems," *J. Parallel Distrib. Comput.*, vol. 73, no. 6, pp. 851–865, 2013.
[28] Z. Li, M. Mobin, and T. Keyser, "Multi-objective and multi-stage reliability growth planning in early product-development stage," *IEEE Trans. Rel.*, vol. 65, no. 2, pp. 769–781, Jun. 2016.
[29] A. Dogan and F. Ozguner, "Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 308–323, Mar. 2002.
[30] A. Doğan and F. Özgüner, "Biobjective scheduling algorithms for execution time–reliability trade-off in heterogeneous computing systems," *Comput. J.*, vol. 48, no. 3, pp. 300–314, Mar. 2005.
[31] A. Girault and H. Kalla, "A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate," *IEEE Trans. Depend. Secure Comp.*, vol. 6, no. 4, pp. 241–254, Oct.–Dec. 2009.
[32] J. J. Dongarra, E. Jeannot, E. Saule, and Z. Shi, "Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems," in *Proc. 19th ACM Int. Symp. Parallel Algorithms Archit.*, 2007, pp. 280–288.
[33] M. Lin, Y. Pan, L. T. Yang, M. Guo, and N. Zheng, "Scheduling co-design for reliability and energy in cyber-physical systems," *IEEE Trans. Emerg. Topics Comput.*, vol. 1, no. 2, pp. 353–365, Dec. 2013.
[34] Z. Li, L. Wang, S. Li, S. Ren, and G. Quan, "Reliability guaranteed energy-aware frame-based task set execution strategy for hard real-time systems," *J. Syst. Softw.*, vol. 86, no. 12, pp. 3060–3070, Dec. 2013.
[35] J. Yi, Q. Zhuge, J. Hu, S. Gu, M. Qin, and H. M. Sha, "Reliability-guaranteed task assignment and scheduling for heterogeneous multiprocessors considering timing constraint," *J. Signal Process. Syst.*, vol. 81, no. 3, pp. 1–17, Dec. 2015.
[36] R. Obermaisser, C. El Salloum, B. Huber, and H. Kopetz, "From a federated to an integrated automotive architecture," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 7, pp. 956–965, Jul. 2009.
[37] 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.
[38] 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.
[39] A. D. Burns and R. Davis, "Mixed-criticality systems: A review (eighth edition)," 2016. [Online]. Available: http://www-users.cs.york.ac.uk/burns/review.pdf
[40] G. Bernat, A. Colin, and S. M. Petters, "WCET analysis of probabilistic hard real-time systems," in *Proc. 23rd IEEE Real-Time Syst. Symp.*, 2002, pp. 279–288.
[41] G. Xie *et al.*, "WCRT analysis of can messages in gateway-integrated in-vehicle networks," *IEEE Trans. Veh. Technol.*, vol. 66, no. 11, pp. 9623–9637, Nov. 2017.
[42] G. Xie, G. Zeng, L. Liu, R. Li, and K. Li, "Mixed real-time scheduling of multiple dags-based applications on heterogeneous multi-core processors," *Microprocess. Microsyst.*, vol. 47, pp. 93–103, Nov. 2016.



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

He is currently an Associate Professor of computer science and engineering with Hunan University. He was a Postdoctoral Researcher with Nagoya University, Nagoya, Japan, from 2014 to 2015, and with Hunan University, from 2015 to 2017. His major interests include embedded and cyber-physical systems, parallel and distributed systems, and software engineering and methodology.

Dr. Xie is a member of ACM and CCF. He was the recipient of best paper award at ISPA 2016.

Yuekun Chen is currently working toward the Ph.D. degree in computer science and engineering at Hunan University, Changsha, China.

Her research interests include embedded computing, fault tolerance computing, and software engineering.

Yan Liu received the Ph.D. degree in computer science and engineering from Hunan University, Changsha, China, in 2010.

He is currently an Assistant Professor with the College of Computer Science and Electronic Engineering, Hunan University. His major interests include computer architectures and embedded computing systems.

Dr. Liu is a member of CCF.

Renfa Li (M'05–SM'10) received the Ph.D. degree in electronic engineering from Huazhong University of Science and Technology, Wuhan, China, in 2002.

He is currently a Professor of computer science and electronic engineering with Hunan University, Changsha, China. He is the Director with the Key Laboratory for Embedded and Network Computing of Hunan Province, Changsha, China. He is also an expert committee member of the National Supercomputing Center in Changsha, China. His major 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 currently a SUNY Distinguished Professor of computer science with the State University of New York at New Paltz, New Paltz, NY, USA. He has authored or coauthored more than 520 journal articles, book chapters, and refereed conference papers. His current 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.

Prof. Li was the recipient of several best paper awards. He is currently serving or has served on the editorial boards of IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON CLOUD COMPUTING, IEEE TRANSACTIONS ON SERVICES COMPUTING, and IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING.