Fast Functional Safety Verification for Distributed Automotive Applications During Early Design Phase

Guoqi Xie, *Member, IEEE*, Gang Zeng, *Member, IEEE*, Yan Liu, Jia Zhou, Renfa Li, *Senior Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—Both response time and reliability are important functional safety properties that must be simultaneously satisfied learning from the automotive functional safety standard ISO 26262. Safety verification pertains to checking if an application meets a safe set of design specifications and complies with regulations. Introducing verification in the early design phase not only complies with the latest automotive functional safety standard but also avoids unnecessary design effort or reduces the design burden of the late design optimization phase. This study presents a fast functional safety verification (FFSV) method for a distributed automotive application during the early design phase. The first method FFSV1 finds the solution with the minimum response time under the reliability requirement, and the second method FFSV2 finds the solution with the maximum reliability under the response time requirement. We combine FFSV1 and FFSV2 to create union FFSV (UFFSV), which can obtain acceptance ratios higher than those of current methods. Experiments on real-life and synthetic distributed automotive applications show that UFFSV can obtain higher acceptance ratios than their existing counterparts.**

***Index Terms*—Automotive functional safety, ISO 26262, verification.**

Manuscript received April 18, 2017; revised July 28, 2017 and September 16, 2017; accepted October 5, 2017. Date of publication October 12, 2017; date of current version January 16, 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. *(Corresponding author: Yan Liu.)*

G. Xie, Y. Liu, J. Zhou, and R. Li are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China, and also with the Key Laboratory for Embedded and Network Computing of Hunan Province, Changsha 410082, China (e-mail: xgqman@hnu.edu.cn; liuyan@hun.edu.cn; knight_zhoujia@163.com; lirenfa@hnu.edu.cn).

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

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, 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/TIE.2017.2762621

I. Introduction

A. Motivation

AUTOMOTIVE system is a highly safety-critical industrial electronic system. Many active and passive safety applications have been developed to enhance safe driving, such as antilock braking system, brake-by-wire, and adaptive cruise control [1]. In particular, the road vehicles—functional safety standard ISO 26262 was officially released in 2011 for adapting safety of automotive applications [2]–[4]. Functional safety has become the preferential direction of automotive system development, and it refers to the absence of unreasonable risk caused by systematic failures and random hardware failures [2].

Safety usually refers to satisfying the response time requirement (i.e., real-time requirement, timing constraint, and deadline constraint) and reliability requirement (i.e., reliability goal, reliability assurance, and reliability constraint) of an application. Safety verification pertains to checking if an application meets a safe set of design specifications and complies with regulations. Automotive industry is cost sensitive to the mass market, and thus, the development cost, hardware cost, and resource cost design optimization for safety-critical distributed automotive applications have been studied [5]–[8]. However, the aforementioned works only focused on either satisfying the response time or reliability requirement rather than functional safety requirement. Response time and reliability requirements are nonfunctional requirements in requirements engineering discipline [9]; however, response time and reliability are important functional safety properties learning from the ISO 26262 standard; their requirements must be simultaneously satisfied for automotive functional safety [2]. Before cost design optimization, we should verify the feasibility of design optimization. If design optimization is infeasible, then designers can avoid unnecessary design effort. If it is feasible, then designers can reduce the design burden by using verification results as basis because verification is part of the design process. Introducing verification in the early design phase not only complies with the latest automotive functional safety standard but also avoids unnecessary design effort or reduces the design burden of the late design optimization phase.

A directed acyclic graph (DAG) can be used to represent a distributed automotive application with end-to-end computation,

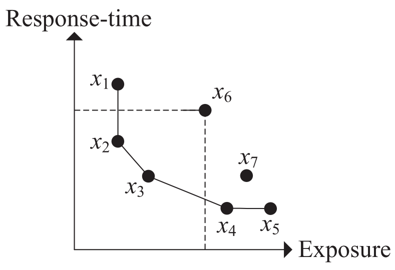


Fig. 1. Pareto curve for a bicriteria between response time and exposure [10], [11].

in which the nodes represent tasks and the edges represent the communication messages between tasks [1], [12]. The problem is that response time and reliability may not be satisfied simultaneously in practice because increasing reliability intuitively increases the response time of a DAG-based distributed application [10], [11]. ISO 26262 defines the exposure to represent the relative expected frequency of the operational conditions, in which hazardous events may occur and cause hazards and injuries [2]. That is, reliability is just the inverse expression of exposure. Response time minimization and exposure minimization (i.e., reliability maximization) are conflicting processes, such that verifying functional safety is a bicriteria optima problem. In Fig. 1, each point x^1–x^7 represents a solution of a bicriteria minimization problem [10], [11]. The points x^1, x^2, x^3, x^4, and x^5 are Pareto optima; the points x^1 and x^5 are weak optima, whereas the points x^2, x^3, and x^4 are strong optima. The set of all Pareto optima is called Pareto curve. In [10], Girault and Kalla presented a bicriteria scheduling heuristic (BSH) to generate an approximate Pareto curve of nondominated solutions, among which the designers can verify the functional safety by finding the points that satisfy the reliability and response time requirements simultaneously. However, the time complexity of BSH is as high as $O(|N| \times 2^{|U|})$, where $|N|$ and $|U|$ are the number of tasks and electronic control units (ECUs), respectively. Currently, a high-end automotive system comprises at least 70 heterogeneous ECUs, and the number of ECUs is expected to increase further in future automotive systems [1], [12]. Considering that the automotive industry is cost-sensitive, shortening the application's development cycle to reduce development cost is crucial. Therefore, a fast functional safety verification (FFSV) method with low time complexity should be proposed from a cost control perspective.

B. Overview of the Study

A life cycle of industrial software development usually involves the analysis, design, implementation, and testing phases [8], [13]–[15]. In this work, we focus on the early design phase, in which verification is used to determine the feasibility of design optimization in advance. The overview of this paper is shown in Fig. 2. The main contents are as follows.

1) In the analysis phase, we first assess the reliability requirement in Section III-C and the response time requirement in Section III-D. Even if the reliability requirement and the response time requirement are satisfied independently, the functional safety requirement containing reliability requirement and response time requirement may not be satisfied, because reliability maximization and response time minimization are conflicting.
2) In the early design phase, we propose two FFSV methods. One method called FFSV1 is to find the solution with the minimum response time under the reliability requirement, and the second method called FFSV2 is to find the solution with the maximum reliability under the response time requirement. We combine FFSV1 and FFSV2 to form a union FFSV (UFFSV). As long as either verification method returns true, the verification returns true.
3) In the late design phase, if at least one of the two methods can find a solution, then designers can present the cost optimization schemes based on the corresponding solutions.

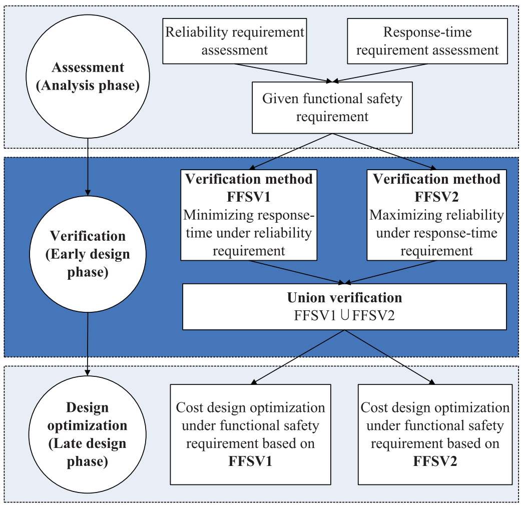

Assessment (Analysis phase)
Reliability requirement assessment
Response-time requirement assessment
Given functional safety requirement
Verification (Early design phase)
Verification method FFSV1 Minimizing response-time under reliability requirement
Verification method FFSV2 Maximizing reliability under response-time requirement
Union verification FFSV1 ∪ FFSV2
Design optimization (Late design phase)
Cost design optimization under functional safety requirement based on FFSV1
Cost design optimization under functional safety requirement based on FFSV2

Fig. 2. Overview of this study for fast functional safety verification.

C. Contribution of the Study

The main contributions of this study are to introduce functional safety verification in the early design phase for distributed automotive application development and propose two heuristic verification methods to achieve fast union verification. The details are summarized as follows.

1) The FFSV1 method solves the problem of minimizing response time under reliability requirement. The problem is divided into two subproblems, namely, satisfying reliability requirement and minimizing response time. The first subproblem is solved by transferring the reliability requirement of the application to each task. The second subproblem is solved by assigning each task to the ECU with the minimum earliest finish time (EFT) under satisfying the reliability requirement of the application. Finally, the verification result can be judged by comparing the obtained response time with the given response time requirement.
2) The FFSV2 method solves the problem of maximizing reliability under response time requirement. The problem is

also divided into two subproblems, namely, satisfying response time requirement and maximizing reliability. The first subproblem is solved by transferring the response time requirement of the application to each task. The second subproblem is solved by migrating partial tasks to the ECUs with the maximum reliability values under satisfying the response time requirement of the application. Finally, the verification result can be judged by comparing the obtained reliability with the given reliability requirement.

3) We combine FFSV1 and FFSV2 to form UFFSV. As long as either verification method returns true, the verification returns true, otherwise returns false.

II. Related Work

As stated in ISO 26262, random hardware failures (i.e., transient failures in most studies) occur unpredictably during the life cycle of a hardware element, but they follow a probability distribution [2]. A widely accepted reliability model was presented by Shatz and Wang [16], in which the transient failure of each hardware follows a constant-parameter Poisson law [8], [10].

Automotive industry is cost-sensitive to the mass market, as pointed out in Section I-A. The development cost, hardware cost, and resource consumption cost design optimization for safety-critical distributed automotive applications were studied in [5]–[8]. In [5] and [6], Gan *et al.* presented development cost minimization to satisfy the response time requirement by presenting genetic algorithm-based and tabu search-based metaheuristics, respectively. In [7], Gu *et al.* presented hardware cost minimization to satisfy the response time and security requirements. Despite the introduction of these methods, both reliability and response time requirements must be simultaneously satisfied learning from automotive functional safety standard. Response time minimization and reliability maximization (i.e., exposure minimization) are conflicting in scheduling a DAG-based application, and optimizing them is a bicriteria optimization problem as pointed out in Section I-A [10]. In [17], Zhao *et al.* presented a shared recovery-based frequency assignment technique to reduce energy consumption with a reliability requirement and a response time requirement for a DAG-based distributed application on a single processor. However, multiprocessors have been used in most embedded systems, such as automotive embedded systems. In [8], Xie *et al.* solved the problem of minimizing the resource consumption cost by satisfying the reliability requirement without using fault-tolerance on heterogeneous embedded systems. In [11], Xie *et al.* solved the problem of minimizing the redundancy by satisfying the reliability requirement using fault-tolerance on heterogeneous service-oriented systems. Considering the limited resources of embedded systems, fault-tolerance may be unsuitable [8].

III. Models

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

TABLE I
Notations in This Study

Notation	Definition
$w_{i,k}$	WCET of the task n_i on the ECU u_k
$c_{i,j}$	WCRT between the tasks n_i and n_j
$\text{rank}_u(n_i)$	Upward rank value of the task n_i
$u_{\text{pr}(i)}$	Assigned ECU of the task n_i
$\|X\|$	Size of the set X
λ_k	Failure rate of the ECU u_k
$n_{\text{seq}(y)}$	yth assigned task of the application
$R(n_i, u_k)$	Reliability of the task n_i on the ECU u_k
$R(n_i)$	Reliability of the task n_i
$R_{\text{req}}(n_i)$	Reliability requirement of the task n_i
$R_{\max}(n_i)$	Maximum reliability of the task n_i
$R(G)$	Reliability of the application G
$R_{\max}(G)$	Maximum Reliability of the application G
$R_{\text{req}}(G)$	Reliability requirement of the application G
$\text{RR}(G)$	Reliability ratio of the application G
$R_{\text{rrp}}(G)$	Ratio-based reliability pre-assignment of the application G
$R_{\text{rrp}}(n_i)$	Ratio-based reliability pre-assignment of the task n_i
$\text{LB}(G)$	Lower bound of the application G
$\text{RT}(G)$	Response time of the application G
$\text{EST}(n_i, u_k)$	Earliest start time of the task n_i on the ECU u_k
$\text{EFT}(n_i, u_k)$	Earliest finish time of the task n_i on the ECU u_k
$\text{AST}(n_i)$	Actual start time of the task n_i
$\text{AFT}(n_i)$	Actual finish time of the task n_i
$\text{RT}_{\text{req}}(n_i)$	Response time requirement of the task n_i
$\text{RT}_{\text{req}}(G)$	Response time requirement of the application G

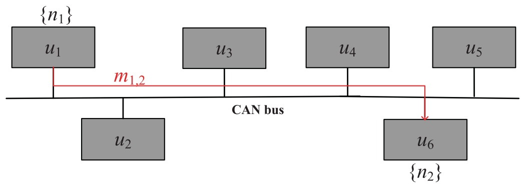


Fig. 3. Integration architecture of automotive electronic systems [1].

A. System Architecture and Models

Each application is implemented by a single ECU in an early automotive architecture where ECUs use point-to-point communication with a low degree of coupling. The next federated architecture enables the exchange of information between different applications through the network. New integrated architecture is the mainstream architecture of today's automobiles [18], [19]. We consider a distributed integrated architecture where several processors are mounted on the same controller area network (CAN) bus, as shown in Fig. 3 [6]. A task executed completely in one ECU sends messages to all its successor tasks, which may be located in the different ECUs. For example, task n_1 is executed on ECU u_1. It then sends a message $m_{1,2}$ to its successor task n_2 located in u_6. $U = \{u_1, u_2, ..., u_{|U|}\}$ 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.

A distributed application is represented by a DAG $G = (N, W, M, C)$ [1], [20], where the related parameters are described as follows.

1) N represents a set of tasks in G, and $n_i \in N$ represents the ith task of G. $\text{pred}(n_i)$ represents the set of the

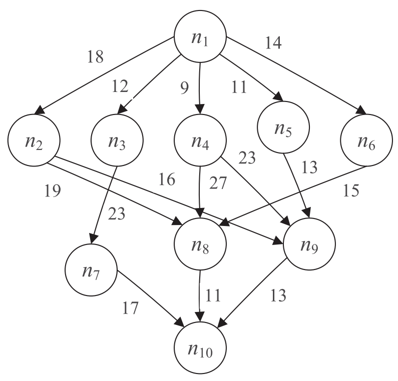


Fig. 4. Motivating example of a DAG-based distributed application with ten tasks [8], [20].

TABLE II
WCETs of Tasks on Different ECUs of the Motivating Distributed Application [8], [20]

Task	u_1	u_2	u_3
n_1	14	16	9
n_2	13	19	18
n_3	11	13	19
n_4	13	8	17
n_5	12	13	10
n_6	13	16	9
n_7	7	15	11
n_8	5	11	14
n_9	18	12	20
n_{10}	21	7	16

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}. Considering an automotive application (e.g., brake-by-wire) may be released by receiving collected data from multiple sensors and is completed by sending the performing action to multiple actuators, multiple n_{entry} or multiple n_{exit} tasks may exist. To adapt to the application model with only one entry task and one exit task, a dummy entry or exit task with zero-weight dependencies is added to the graph in this case.

2) W is a $|N| \times |U|$ matrix, where $w_{i,k}$ denotes the worse-case execution time (WCET) of n_i on the ECU u_k. Each task $n_i \in N$ has different WCET values on different ECUs due to the heterogeneity of ECUs [21]. All the WCETs of the tasks are determined through analysis methods performed (i.e., WCET analysis) during the analysis phase [1].

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. Accordingly, $c_{i,j} \in C$ represents the worst-case response time (WCRT) of $m_{i,j}$ [1]. All the WCRTs of the messages are also determined through analysis methods performed (i.e., WCRT analysis) during the analysis phase [1].

Scheduling in automotive systems can be either preemptive (e.g., OSEKTime) or nonpreemptive (e.g., eCos) [1]. Considering that many DAG-based distributed application scheduling algorithms generally use nonpreemptive scheduling [1], [8], [20], we consider nonpreemptive scheduling for ECUs in this study. Of course, the method of this paper can also be applied to preemptive scheduling.

Fig. 4 shows a motivating distributed application with tasks and messages [8], [20]. The example shows ten tasks executed on three ECUs $\{u_1, u_2, u_3\}$. The weight 18 of the edge between n_1 and n_2 represents WCRT, denoted by $c_{1,2}$ if n_1 and n_2 are not assigned to the same ECU. Table II presents the WCET matrix $|N| \times |U|$ of tasks on different ECUs. For example, the weight 14 of n_1 and u_1 in Table II represents WCET of n_1 on u_1, denoted by $w_{1,1} = 14$. The same task has different WCETs on different ECUs due to the heterogeneity of the ECUs. The motivating example will be used to explain the proposed verification methods in the paper. For simplicity, all units of all parameters are ignored in the example.

B. Reliability Model

There are two major temporal types of failures, namely, the transient failure (i.e., random hardware failures) and the permanent failure [8], [10]. This study only considers the transient failure of ECUs because the automotive functional safety standard ISO 26262 only combines the random hardware failures and reliability together [2]. ISO 26262 specifies that random hardware failures occur unpredictably during the life cycle of a hardware but follows a probability distribution [2]. In general, transient fault for a task in a DAG-based distributed application follows the Poisson distribution [8], [10]. The reliability of an event in unit time t is denoted by $R(t) = e^{-\lambda t}$, where λ is the constant failure per time unit (i.e., failure rate) for an ECU [16]. We let λ_k represent the constant failure rate per time unit of the ECU u_k, and the reliability of n_i executed on u_k and its execution time is denoted by

$$R(n_i, u_k) = e^{-\lambda_k w_{i,k}}. \tag{1}$$

The reliability of the DAG-based distributed application is calculated by [8], [10]

$$R(G) = \prod_{n_i \in N} R(n_i, u_{\text{pr}(n_i)}) \tag{2}$$

where $u_{\text{pr}(n_i)}$ represents the assigned ECU of n_i. As CAN bus has high fault-tolerance capacity, this study only considers ECU failure and does not factor the communication failure into the problem (i.e., communication is assumed reliable).

C. Reliability Requirement Assessment

As the WCET of each task on each ECU has been determined by the WCET analysis method during the analysis phase, the maximum reliability value of task n_i can be obtained by

TABLE III
UPWARD RANK VALUES FOR TASKS OF THE MOTIVATING DISTRIBUTED APPLICATION

Task	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
$\text{rank}_u(n_i)$	108	77	80	80	69	63.3	42.7	35.7	44.3	14.7

traversing all the ECUs, and it is calculated by

$$R_{\max}(n_i) = \max_{u_k \in U} R(n_i, u_k). \tag{3}$$

As the reliability of application G is the product of reliability values of all the tasks, [see (2)], the maximum reliability value of application G is calculated by

$$R_{\max}(G) = \prod_{n_i \in N} R_{\max}(n_i). \tag{4}$$

The reliability requirement $R_{\text{req}}(G)$ must be less than or equal to $R_{\max}(G)$. Hence, $R_{\text{req}}(G)$ must have the following constraint:

$$R_{\text{req}}(G) \leqslant R_{\max}(G) \tag{5}$$

otherwise, the reliability requirement assessment cannot be passed.

For example, we assume that the failure rates of ECUs u_1, u_2, and u_3 are $\lambda_1 = 0.0005$, $\lambda_2 = 0.0002$, and $\lambda_3 = 0.0009$, respectively. We can calculate that the maximum reliability value of the application is $R_{\max}(G) = 0.974335$. We let reliability requirement $R_{\text{req}}(G) = 0.96$, which can pass the assessment, because

$$0.96 \leqslant 0.974335$$

according to (5) of the reliability requirement assessment.

D. Response Time Requirement Assessment

Considering the problem of mapping tasks to multiple processors (ECUs) is NP-hard [22], obtaining a minimum end-to-end response time of a distributed automotive application is an NP-hard optimization problem [20]. The heterogeneous EFT (HEFT) algorithm is a widely accepted heuristic list-scheduling algorithm for minimizing response time [20], and it is applied for the response time requirement assessment of distributed automotive applications [1]. The two-phase HEFT algorithm has the following important steps.

1) Prioritizing task. The HEFT algorithm uses the upward rank value (rank_u) of a task [see (6)] as the task priority standard. In this case, the tasks are arranged according to the descending order of rank_u, which is obtained by

$$\text{rank}_u(n_i) = \overline{w_i} + \max_{n_j \in \text{succ}(n_i)} \{c_{i,j} + \text{rank}_u(n_j)\} \tag{6}$$

where $\overline{w_i}$ represents the average WCET of task n_i. Table III shows the upward rank values of all the tasks in Fig. 4. Note that only if all the predecessors of n_i have been assigned will n_i prepare to be assigned. We assume that two tasks n_i and n_j satisfy $\text{rank}_u(n_i) > \text{rank}_u(n_j)$; if no precedence constraint exists between n_i and n_j, n_i does not necessarily take precedence for n_j to be assigned. Therefore, the task assignment order in G is $\{n_1, n_3, n_4, n_2, n_5, n_6, n_9, n_7, n_8, \text{and } n_{10}\}$.

2) Response time minimization. The attributes $\text{EST}(n_i, u_k)$ and $\text{EFT}(n_i, u_k)$ represent the earliest start time (EST) and EFT, respectively, of task n_i on ECU u_k. $\text{EFT}(n_i, u_k)$ is considered the task allocation criterion because it can meet the local optimal of each task. The aforementioned attributes are calculated as follows:

$$\begin{cases} \text{EST}(n_{\text{entry}}, u_k) = 0; \\ \text{EST}(n_i, u_k) = \max \left\{ \begin{array}{l} \text{avail}[k], \\ \max\limits_{n_h \in \text{pred}(n_i)} \{\text{AFT}(n_h) + c'_{h,i}\} \end{array} \right\} \end{cases} \tag{7}$$

and

$$\text{EFT}(n_i, u_k) = \text{EST}(n_i, u_k) + w_{i,k}. \tag{8}$$

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

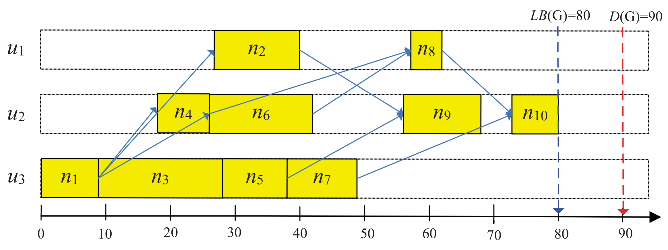


Fig. 5. Task scheduling for lower bound calculation.

The response time requirement assessment processes are as follows.

1) The response time obtained by the HEFT algorithm represents the application's lower bound. The lower bound means the minimum response time of a distributed application by the HEFT algorithm and is calculated as follows:

$$\text{LB}(G) = \text{AFT}(n_{\text{exit}}) \tag{9}$$

where n_{exit} represents the exit task as mentioned earlier. Fig. 5 shows the scheduling of lower bound calculation of the motivating example, where $\text{LB}(G) = 80$. Note that the arrows in Fig. 5 represent the generated communications between precedence constrained tasks.

2) A known response time requirement $\text{RT}_{\text{req}}(G)$ (i.e., deadline) is then provided for the application on the basis of the actual physical time requirement after hazard analysis

and risk assessment (see Fig. 5). $\mathrm{RT}_{\mathrm{req}}(G)$ must be larger than or equal to $\mathrm{LB}(G)$. Hence, $\mathrm{RT}_{\mathrm{req}}(G)$ must derive the following constraint:

$$\mathrm{LB}(G) \leqslant \mathrm{RT}_{\mathrm{req}}(G) \tag{10}$$

otherwise, the response time requirement assessment cannot be passed. For this motivating example, we let the response time requirement as $\mathrm{RT}_{\mathrm{req}}(G) = 90$, which can pass the assessment, because

$$80 \leqslant 90$$

according to (10) of the reliability requirement assessment.

E. Problem Statement

In this study, the problem to be solved is to verify whether a solution exists for the following conditions:

$$\mathrm{RT}(G) \leqslant \mathrm{RT}_{\mathrm{req}}(G) \tag{11}$$

$$R(G) \geqslant R_{\mathrm{req}}(G) \tag{12}$$

can be simultaneously satisfied. If the result is true, then the functional safety verification is passed and the late cost design optimization is feasible; otherwise, functional safety verification is failed and the late cost design optimization is infeasible.

IV. Minimizing Response Time Under Reliability Requirement

This section presents the first verification method by minimizing the response time under the reliability requirement.

A. Satisfying Reliability Requirement

In [8], Xie *et al.* solved the problem of minimizing the resource consumption cost under the reliability requirement by proposing the MRCRG algorithm, where the strategy of satisfying the reliability requirement is as follows. Assume that the task to be assigned is $n_{s(y)}$, where $s(y)$ represents the yth assigned task (sequence number), $\{n_{s(1)}, n_{s(2)}, \ldots, n_{s(y-1)}\}$ represents the task set where the tasks have been assigned, and $\{n_{s(y+1)}, n_{s(y+2)}, \ldots, n_{s(|N|)}\}$ represents the task set where the tasks have not been assigned. The tasks are arranged according to the descending order of rank_u values as done in the HEFT algorithm. To ensure that the reliability of the application is satisfied at each task assignment, we presuppose that each unassigned task in $\{n_{s(y+1)}, n_{s(y+2)}, \ldots, n_{s(|N|)}\}$ is assigned to the ECU with the maximum reliability value. Therefore, when assigning $n_{s(y)}$, the reliability of G has the following constraint:

$$R_{s(y)}(G) = \prod_{x=1}^{y-1} R\left(n_{s(x)}\right) \times R\left(n_{s(y)}\right) \times \prod_{z=y+1}^{|N|} R_{\max}\left(n_{s(z)}\right) \geqslant R_{\mathrm{req}}(G)$$

then, $R\left(n_{s(y)}\right)$ must derive the following constraint:

$$R(n_{s(y)}) \geqslant \frac{R_{\mathrm{req}}(G)}{\prod_{x=1}^{y-1} R(n_{s(x)}) \times \prod_{z=y+1}^{|N|} R_{\max}(n_{s(z)})}.$$

In the following, MRCRG lets the reliability requirement of the task $n_{s(y)}$ be

$$R_{\mathrm{req}}(n_{s(y)}) = \frac{R_{\mathrm{req}}(G)}{\prod_{x=1}^{y-1} R(n_{s(x)}) \times \prod_{z=y+1}^{|N|} R_{\max}(n_{s(z)})}. \tag{13}$$

The reliability requirement of the application is transferred to each task, and the actual reliability must have the following constraint:

$$R(n_{s(y)}) \geqslant R_{\mathrm{req}}(n_{s(y)}).$$

However, a major limitation for the above-mentioned transfer is that the reliability pre-assignment with the maximum reliability value for unassigned tasks is severely pessimistic toward an unfair reliability usage among tasks and thus results in the limited minimization of the response time. In view of the pessimistic reliability pre-assignment for unassigned tasks in MRCRG, this section presents an optimistic reliability pre-assignment method.

We first define the reliability ratio of the application.

Definition 1 (Reliability Ratio): The reliability ratio of the application is the ratio of the reliability requirement of the application to the maximum reliability of the application:

$$\mathrm{RR}(G) = \frac{\sqrt[|N|]{R_{\mathrm{req}}(G)}}{\sqrt[|N|]{R_{\max}(G)}}. \tag{14}$$

We obtain $\mathrm{RR}(G) \leqslant 1$ because $R_{\mathrm{req}}(G) \leqslant R_{\max}(G)$ according to (5) in the previous reliability requirement assessment.

Unlike in MRCRG, in which each unassigned task in $\{n_{s(y+1)}, n_{s(y+2)}, \ldots, n_{s(|N|)}\}$ is pre-assigned to the ECU with the maximum reliability, the ratio-based reliability pre-assignment calculated by (15) is pre-assigned for these unassigned tasks in this study:

$$R_{\mathrm{rrp}}(n_{s(z)}) = R_{\max}(n_{s(z)}) \times \mathrm{RR}(G). \tag{15}$$

Obviously, we have

$$R_{\mathrm{rrp}}(n_{s(z)}) \leqslant R_{\max}(n_{s(z)}) \tag{16}$$

because $\mathrm{RR}(G) \leqslant 1$ [see (14)]. Correspondingly, the reliability requirement of task $n_{s(y)}$ is then changed to

$$R_{\mathrm{req}}(n_{s(y)}) = \frac{R_{\mathrm{req}}(G)}{\prod_{x=1}^{y-1} R(n_{s(x)}) \times \prod_{z=y+1}^{|N|} R_{\mathrm{rrp}}(n_{s(z)})}. \tag{17}$$

The sole difference between (13) and (17) is that $R_{\max}(n_{s(y)})$ in the former is replaced with $E_{\mathrm{rrp}}(n_{s(y)})$ in the latter. In this manner, the reliability requirement of the application can still be transferred to each task. We prove the correctness in Theorem 1.

Theorem 1: When the ratio-based reliability pre-assignment $R_{\mathrm{rrp}}(n_i)$ [see (15)] is used, each task n_i in the distributed automotive application G can always find an available ECU to satisfy the reliability requirement $R_{\mathrm{req}}(G)$ of the application.

Proof: We prove the theorem step by step as follows.

First, we let the product of the ratio-based reliability pre-assignments of all tasks be

$$R_{\text{rrp}}(G) = \prod_{n_i \in N} R_{\text{rrp}}(n_i). \tag{18}$$

If we can prove that the product of the ratio-based reliability pre-assignments of all tasks is larger than or equal to the given reliability requirement of the application, then the pre-assignment method is feasible and the theorem is proved.

Second, we substitute (15) into (18) and obtain

$$\begin{aligned} R_{\text{rrp}}(G) &= \prod_{n_i \in N} R_{\text{rrp}}(n_i) = \prod_{n_i \in N} (R_{\max}(n_i) \times \text{RR}(G)) \\ &= \prod_{n_i \in N} R_{\max}(n_i) \times \prod_{n_i \in N} \text{RR}(G). \end{aligned} \tag{19}$$

Third, we substitute (14) into (19) and yield

$$\begin{aligned} R_{\text{rrp}}(G) &= \prod_{n_i \in N} R_{\max}(n_i) \times \prod_{n_i \in N} \frac{\sqrt[|N|]{R_{\text{req}}(G)}}{\sqrt[|N|]{R_{\max}(G)}} \\ &= R_{\max}(G) \times \frac{R_{\text{req}}(G)}{R_{\max}(G)} = R_{\text{req}}(G). \end{aligned} \tag{20}$$

Finally, given that $R_{\text{rrp}}(G)$ is equal to $R_{\text{req}}(G)$ under the ratio-based reliability pre-assignment for tasks in (20), we can find assigned ECUs to satisfy $R_{\text{req}}(G)$. Thus, Theorem 1 is proved.

The reliability requirement of each task is valid according to Theorem 1. That is, just let the actual reliability value of $n_{s(y)}$ satisfy the following constraint:

$$R(n_{s(y)}) \geqslant R_{\text{req}}(n_{s(y)}).$$

Hence, when assigning task $n_{s(y)}$, the reliability requirement $R_{\text{req}}(n_{s(y)})$ of $n_{s(y)}$ can be directly considered and the reliability requirement of the application G is not a concern. In this manner, a low time complexity heuristic algorithm can be achieved.

B. Minimizing Response Time

Similar to the HEFT algorithm [20], we also let n_i be allocated to the ECU with the minimum EFT by using the insertion-based scheduling strategy, which is a local optimal heuristic strategy. Local optimality means that when allocating each task, minimum EFT is optimal for the current task; however, minimum EFT is not global optimal for the application. In other words, local optimality is related to the current task, whereas global optimality is related to the application. Considering that the reliability requirement of each task has been satisfied in the previous section, we just minimize the AFT of each task by traversing all available ECUs under its reliability requirement. That is, the assigned ECU $u_{\text{pr}(i)}$ for n_i is determined by

$$\text{EFT}(n_i, u_{\text{pr}(i)}) = \min_{u_k \in U, R(n_i, u_k) \geqslant R_{\text{req}}(n_i)} \{\text{EFT}(n_i, u_k)\}. \tag{21}$$

In this manner, the same low complexity heuristic algorithm as the HEFT algorithm is achieved.

Algorithm 1: The FFSV1 Algorithm.

```
Input: $U = \{u_1, u_2, \ldots, u_{|U|}\}$, $G$, $R_{\text{req}}(G)$, $\text{RT}_{\text{req}}(G)$
Output: $R(G)$, $\text{RT}(G)$, verification result
 1: Sort the tasks in a list $task\_list$ by descending order of
    $\text{rank}_u$ values;
 2: Calculate $\text{RR}(G)$ using (14);
 3: while (there are tasks in $task\_list$) do
 4:   $n_i \leftarrow n_{s(y)} \leftarrow task\_list.out()$;
 5:   Calculate $R_{\text{req}}(n_i)$ using (17);
 6:   for (each ECU $u_k \in U$) do
 7:     Calculate $R(n_i, u_k)$ using (1);
 8:     if ($R(n_i, u_k) < R_{\text{req}}(n_i)$) then
 9:       continue;
10:     end if
11:     Calculate $\text{EFT}(n_i, u_k)$ using (8);
12:   end for
13:   Select the ECU $u_{\text{pr}(i)}$ with the minimum EFT;
14:   $\text{AFT}(n_i) \leftarrow \text{EFT}(n_i, u_{\text{pr}(i)})$;
15: end while
16: Calculate the reliability $R(G)$ using (2);
17: $\text{RT}(G) \leftarrow \text{AFT}(n_{\text{exit}})$;
18: if ($\text{RT}(G) < \text{RT}_{\text{req}}(G)$) then
19:   return true;
20: else
21:   return false;
22: end if
```

Then, we assign n_i to $u_{\text{pr}(i)}$ based on (21), and the actual AFT and AST of n_i are calculated as follows:

$$\text{AFT}(n_i) = \text{EFT}(n_i, u_{\text{pr}(i)}) \tag{22}$$

$$\text{AST}(n_i) = \text{AFT}(n_i) - w_{i,\text{pr}(i)}. \tag{23}$$

The first functional safety verification (FFSV1) algorithm is proposed, as shown in Algorithm 1.

The idea of FFSV1 is that the reliability requirement of the application is transferred to each task using a ratio-based reliability pre-assignment, and each task just selects the ECU with the minimum EFT under the reliability requirement. The main details are explained as follows.

1) Prioritizing task. In line 1, FFSV1 sorts all tasks in a list task_list by descending order of rank_u values. In line 2, FFSV1 calculates the reliability ratio using (14). In the following, all tasks will be traversed.
2) Satisfying the reliability requirement. In line 5, FFSV1 calculates the reliability requirement of the current task n_i using (17). Specifically, in lines 8–10, FFSV1 skips the ECUs that do not satisfy the reliability requirement of n_i.
3) Minimizing the response time. In lines 6–13, FFSV1 selects the ECU with the minimum EFT for the current task n_i that satisfies the condition of $R(n_i, u_k) \geqslant R_{\text{req}}(n_i)$.
4) Verifying functional safety. In lines 16 and 17, FFSV1 calculates the final reliability $R(G)$ and the response time

TABLE IV
TASK ASSIGNMENT OF THE MOTIVATING DISTRIBUTED APPLICATION GENERATED BY FFSV1

n_i	$R_{\text{req}}(n_i)$	$\text{EFT}(n_i, u_1)$	$\text{EFT}(n_i, u_2)$	$\text{EFT}(n_i, u_3)$	$R(n_i)$
n_1	0.995329	–	**16**	–	0.996805
n_3	0.994451	39	**29**	–	0.997403
n_4	0.993972	–	**37**	–	0.998401
n_2	0.990318	**47**	56	–	0.993521
n_5	0.992716	59	**50**	–	0.997403
n_6	0.991933	60	66	**39**	0.991933
n_9	0.994838	–	**75**	–	0.997603
n_7	0.992769	**59**	90	–	0.996506
n_8	0.992588	**69**	86	–	0.997503
n_{10}	0.992209	–	**87**	–	0.998601
$\text{RT}(G) = 87$, $R(G) = 0.966185 > R_{\text{req}}(G) = 0.96$					

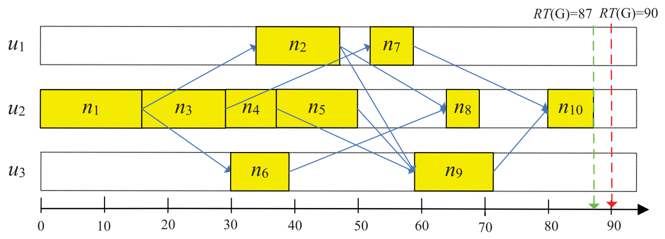


Fig. 6. Task scheduling of the motivating distributed application generated by FFSV1.

$\text{RT}(G)$ of the application, respectively. In lines 18–22, FFSV1 judges the verification result by comparing the obtained response time with the given response time requirement.

The time complexity of the FFSV1 algorithm is analyzed as follows: scheduling all the tasks of the application must traverse all tasks, which can be done within $O(|N|)$ time; calculating the EFT value of the current task must traverse its immediate precedence tasks, all ECUs, which can be done in $O(|N| \times |U|)$ time. Therefore, the time complexity of the FFSV1 algorithm is $O(|N|^2 \times |U|)$, which is equal to that of the HEFT algorithm.

C. Example of the FFSV1 Method

Table IV shows the task assignments of the motivating example using FFSV1, where each row represents a task assignment. The values in the bold text mean that the ECU is selected with the minimum EFT. The values denoted with "–" mean that assigning a task to the ECU cannot satisfy the reliability requirement of the task. We can see that the actual reliability value of each task is larger than its reliability requirement. The final reliability is $R(G) = 0.9662$, which can satisfy the reliability requirement of 0.96. Fig. 6 also shows the task scheduling of the motivating distributed application generated by FFSV1, where the response time is 87, which satisfies the response time requirement of 90. The arrows in Fig. 6 represent generated communications between tasks. Therefore, the functional safety requirement of the motivating distributed application can be satisfied and verification result is true by using FFSV1.

V. MAXIMIZING RELIABILITY UNDER RESPONSE TIME REQUIREMENT

This section presents the second verification method by maximizing reliability under response time requirement.

A. Satisfying Response Time Requirement

To implement fast verification, this section aims to transfer the response time requirement of the application to each task. The objective of satisfying the response time requirement of distributed application G is to assure that all the tasks are finished in $\text{RT}_{\text{req}}(G)$ without violating the precedence constraints among tasks. Obviously, the previous response time requirement assessment using the HEFT algorithm in Fig. 5 satisfies the response time requirement, but the reliability value is excessively low. Therefore, we can further enhance the reliability after using the HEFT algorithm.

We can migrate the tasks to other ECUs to achieve reliability enhancement without violating the response time requirement of the application and precedence constraints among tasks. Some slacks may exist per ECU, and tasks previously assigned to other ECUs can be migrated to these slacks as long as such migration can obtain a higher reliability than the previous assignment for the current task. In this section, contrary to the HEFT algorithm and FFSV1, the tasks are arranged according to the ascending order of rank_u values to implement the RT_{req} extension of tasks.

To implement task migration without violating the precedence constraints among tasks, we first make the following revisions of EST and $\text{RT}_{\text{req}}(n_i)$ calculations as follows.

1) The EST defined in (7) considers the earliest available time $\text{avail}[k]$ that ECU u_k is ready for task execution. However, as we consider task migration in this section, the task n_i should not be restricted by $\text{avail}[k]$, but can be migrated; thus, we update EST as follows:

$$\begin{cases} \text{EST}(n_{\text{entry}}, u_k) = 0 \\ \text{EST}(n_i, u_k) = \max\limits_{n_x \in \text{pred}(n_i)} \{\text{AST}(n_x) + c'_{x,i}\} \end{cases}. \tag{24}$$

2) Only if the task is migrated to another ECU, will larger reliability value be generated; thus, each task must have individual $\text{RT}_{\text{req}}(n_i, u_k)$ on different ECUs, as follows:

$$\begin{cases} \text{RT}_{\text{req}}(n_{\text{exit}}, u_k) = \text{RT}_{\text{req}}(G) \\ \text{RT}_{\text{req}}(n_i, u_k) = \min\limits_{n_j \in \text{succ}(n_i)} \{\text{AST}(n_j) - c'_{i,j}\} \end{cases}. \tag{25}$$

For example, the ESTs and RT_{req} of n_{10} on all slacks can be obtained as

$$\begin{cases} \text{EST}(n_{10}, u_1) = 81 \\ \text{EST}(n_{10}, u_2) = 73 \\ \text{EST}(n_{10}, u_3) = 81 \end{cases} \quad \begin{cases} \text{RT}_{\text{req}}(n_{10}, u_1) = 90 \\ \text{RT}_{\text{req}}(n_{10}, u_2) = 90 \\ \text{RT}_{\text{req}}(n_{10}, u_3) = 90 \end{cases}.$$

Even though the EST and RT_{req} are extended to each ECU for each task, task migration must be further constrained because each ECU is not always available at all times after the HEFT algorithm is used. Some slacks remain in the ECUs due to response time requirement assessment, and task migration or task assignment is conducted by task insertion. The slack set for

ECU u_k is defined as follows:

$$S_{i,k} = \{S_{i,k,1}, S_{i,k,2}, S_{i,k,|S_k|}\}$$

where $S_{i,k,1}$ represents the first slack on u_k for n_i. Each slack has a start time (ST) and end time (ET), and we define the yth slack $S_{i,k,y}$ as follows:

$$S_{i,k,y} = [t_{\text{s}}(S_{i,k,y}), t_{\text{e}}(S_{i,k,y})]$$

where $t_{\text{s}}(S_{i,k,y})$ and $t_{\text{e}}(S_{i,k,y})$ represent ST and ET, respectively. As the application has a given response time requirement, the ET of the last slack must be $t_{\text{e}}(S_{i,k,|S_k|}) = \text{RT}_{\text{req}}(G)$. For example, when assigning the task n_{10} in Fig. 5, the slacks on u_1, u_2, and u_3 are

$$\begin{cases} S_{10,1} = \{[0, 27], [40, 57], [62, 90]\} \\ S_{10,2} = \{[0, 18], [42, 56], [68, 90]\} \\ S_{10,3} = \{[49, 90]\} \end{cases}.$$

Notably, n_{10} must be removed from u_2, such that optimal slacks can be selected for it.

To avoid violating the precedence constraints among tasks, task n_i should be assigned to the slacks that satisfy the new EST and RT_{req} constraints:

$$\text{EST}(n_i, u_k) = \max\{\text{EST}(n_i, u_k), t_{\text{s}}(S_{i,k,t})\} \quad (26)$$

$$\text{RT}_{\text{req}}(n_i, u_k) = \min\{\text{RT}_{\text{req}}(n_i, u_k), t_{\text{e}}(S_{i,k,t})\}. \quad (27)$$

For example, the new EST and RT_{req} values of n_{10} on all ECUs are updated to

$$\begin{cases} \text{EST}(n_{10}, u_1) = 81 \\ \text{EST}(n_{10}, u_2) = 73 \\ \text{EST}(n_{10}, u_3) = 81 \end{cases} \qquad \begin{cases} \text{RT}_{\text{req}}(n_{10}, u_1) = 90 \\ \text{RT}_{\text{req}}(n_{10}, u_2) = 90 \\ \text{RT}_{\text{req}}(n_{10}, u_3) = 90 \end{cases}.$$

B. Maximizing Reliability

This section iteratively assigns each task to the ECU with the maximum reliability under the response time requirement, such that FFSV can be achieved. Considering that task assignment is actually task insertion, we must determine whether task insertion is feasible on each ECU before task assignment. The reason is that small slacks may not accommodate the insertions of tasks with long WCETs. We make the following constraints before the task assignment.

1) The maximum execution time (MET) for n_i on u_k must be derived as follows:

$$\text{MET}(n_i, u_k) = \text{RT}_{\text{req}}(n_i, u_k) - \text{EST}(n_i, u_k). \quad (28)$$

For example, when assigning task n_{10} based on Fig. 5, the METs for n_{10} should be

$$\begin{cases} \text{MET}(n_{10}, u_1) = 9 < 21 = w_{10,1} \\ \text{MET}(n_{10}, u_2) = 17 > 7 = w_{10,2} \\ \text{MET}(n_{10}, u_3) = 9 < 16 = w_{10,3} \end{cases}. \quad (29)$$

2) *The constraint given as follows must be satisfied, else n_i* cannot be inserted into the slack:

$$\text{MET}(n_i, u_k) \geqslant w_{i,k}. \quad (30)$$

For example, n_{10} cannot be inserted into u_1 because $\text{MET}(n_{10}, u_1) = 9$ is less than $w_{10,1} = 21$, which is the WCET of n_{10} on u_1, shown in (30).

3) The strategy of maximizing reliability involves the following steps: considering that the response time requirement of each task has been satisfied in the previous section, we only maximize the reliability of each task by traversing all available ECUs under the task that can be inserted into the slacks of ECUs. That is, the assigned ECU $u_{\text{pr}(i)}$ is determined by

$$R(n_i, u_{\text{pr}(i)}) = \max_{u_k \in U, \text{MET}(n_i, u_k) \geqslant w_{i,k}} \{R(n_i, u_k)\}. \quad (31)$$

$R(n_i, u_{\text{pr}(i)})$ must satisfy the following constraint:

$$R(n_i, u_{\text{pr}(i)}) \geqslant R_{\text{heft}}(n_i)$$

where $R_{\text{heft}}(n_i)$ represents the reliability value of n_i obtained by the HEFT algorithm. The reason is that when the reliability values of n_i on other ECUs are less than $R_{\text{heft}}(n_i)$, then n_i is still assigned to the original ECU obtained by the HEFT algorithm.

4) Then, we assign n_i to $u_{\text{pr}(i)}$ based on (31), and the actual AFT and AST of n_i are updated as follows:

$$\text{AFT}(n_i) = \text{RT}_{\text{req}}(n_i, u_{\text{pr}(i)}) \quad (32)$$

$$\text{AST}(n_i) = \text{RT}_{\text{req}}(n_i, u_{\text{pr}(i)}) - w_{i,\text{pr}(i)}. \quad (33)$$

The FFSV2 algorithm is proposed, as shown in Algorithm 2.

In FFSV2, the response time requirement of the application is transferred to each task by task migration, and each task just selects the ECU with the maximum reliability under the task that can be inserted into the slack of the ECU without violating the precedence constraints of tasks and response time requirement. The main details are explained as follows.

1) Prioritizing task. In line 1, FFSV2 invokes the HEFT algorithm to obtain the initial assignments of task in the application G. In line 2, FFSV2 sorts all tasks in a list task_list in ascending order of rank_u values. In the following, all tasks will be traversed.
2) Satisfying response time requirement. In line 6, FFSV2 calculates the response time requirements of the current task n_i on all ECUs. In lines 8–10, FFSV2 skips the ECU that n_i cannot be inserted into.
3) Maximizing reliability. In lines 5–13, FFSV2 selects the ECU with the maximum reliability value for the current task n_i that satisfies the condition of $\text{MET}(n_i, u_k) \geqslant w_{i,k}$.
4) Verifying functional safety. In lines 16 and 17, FFSV2 calculates the final reliability $R(G)$ and the response time $\text{RT}(G)$ of the application, respectively. In lines 18–22, FFSV2 judges the verification result by comparing the obtained reliability value with the given reliability requirement.

The time complexity of the FFSV2 algorithm is also $O(|N|^2 \times |U|)$, which remains equal to that of the HEFT algorithm. FFSV2 also implements low-time complexity FFSV.

Algorithm 2: The FFSV2 Algorithm.

```
Input: $U = \{u_1, u_2, \ldots, u_{|U|}\}$, $G$, $R_{\text{req}}(G)$, $\text{RT}_{\text{req}}(G)$
Output: $R(G)$, $\text{RT}(G)$, verification result
 1: Invoke the HEFT algorithm to obtain the initial
    assignments of task in the application $G$;
 2: Sort the tasks in a list $task\_list$ by ascending order of
    $\text{rank}_u$ values;
 3: while (there are tasks in $task\_list$) do
 4:   $n_i \leftarrow task\_list.out()$;
 5:   for (each ECU $u_k \in U$) do
 6:     Calculate $\text{RT}_{\text{req}}(n_i, u_k)$ using (25);
 7:     Calculate $\text{MET}(n_i, u_k)$ using (30);
 8:     if ($\text{MET}(n_i, u_k) < w_{i,k}$) then
 9:       continue;
10:     end if
11:     Calculate $R(n_i, u_k)$ using (1);
12:   end for
13:   Select the ECU $u_{\text{pr}(i)}$ with the maximum reliability
      value;
14:   $\text{AFT}(n_i) \leftarrow \text{RT}_{\text{req}}(n_i, u_{\text{pr}(i)})$;
15: end while
16: Calculate the reliability $R(G)$ using (2);
17: $\text{RT}(G) \leftarrow \text{AFT}(n_{\text{exit}})$;
18: if ($R(G) < R_{\text{req}}(G)$) then
19:   return true;
20: else
21:   return false;
22: end if
```

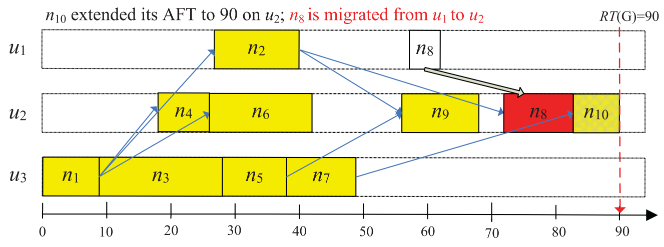


Fig. 7. FFSV2-generated scheduling of n_{10} and n_8.

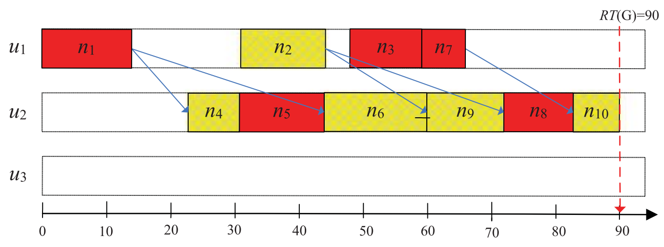


Fig. 8. Task scheduling of the motivating distributed application generated by FFSV2.

C. Example of the FFSV2 Method

Figs. 7 and 8 show the Gantt charts of scheduling the motivating example using FFSV2. The task n_{10} extends its AST and AFT on u_2 as denoted with shadows in Fig. 7. When reassigning n_8, it is migrated from u_2 to u_1 where maximum reliability value of 0.997802 can be obtained without violating the precedence constraints among tasks.

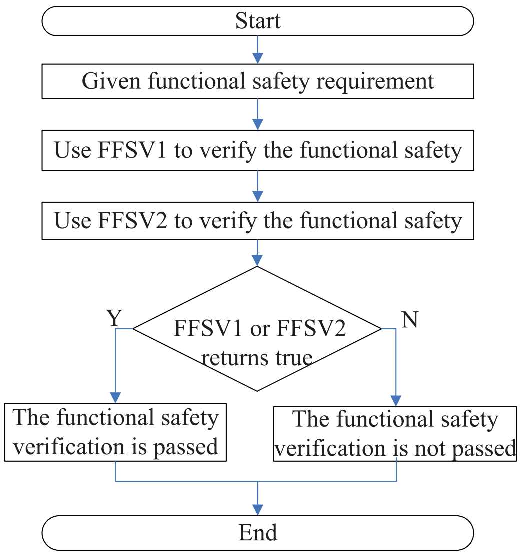

Start
Given functional safety requirement
Use FFSV1 to verify the functional safety
Use FFSV2 to verify the functional safety
FFSV1 or FFSV2 returns true
Y
N
The functional safety verification is passed
The functional safety verification is not passed
End

Fig. 9. Flow chart of the union verification.

Finally, the final response time of the application is 90, which can satisfy the response time requirement of 90, as shown in Fig. 8. The reliability of application is calculated as 0.964737, which satisfies the reliability requirement of 0.96. Therefore, the functional safety requirement of the motivating distributed application can also be satisfied and verification result is true by using FFSV2.

D. Union Verification

Finally, we combine FFSV1 and FFSV2 to form a UFFSV. The flow chart of the union verification is presented in Fig. 9. As long as either verification method returns true, the verification returns true. Regardless, both FFSV1 and FFSV2 should be invoked for the following reasons.

1) FFSV1 and FFSV2 have different emphases, but they are complementary; FFSV1 discusses the minimization of response time under reliability requirement, while FFSV2 explores the maximization of reliability under response time requirement. Combining these two methods achieves the maximum acceptance rate.
2) Considering that both FFSV1 and FFSV2 are fast verification methods with low time complexity, UFFSV exhibits low time complexity and does not affect running efficiency.
3) Using their individual verification results as basis, FFSV1 and FFSV2 can guide individual cost design optimization solutions in the late design phase.

Besides union verification, the following methods can also be considered in the future.

1) We have partially considered important aspect in requirement and design phases shown in Fig. 2. However, escape risk design, safety adequacy assessment for safety aspect can also be considered [23], [24]. Escape risk assessment and design increase reliability and ensure confidence in

functioning. They can also increase the opportunity to satisfy functional safety requirements. Escape risk assessment and design are common in both minimizing response time and enhancing reliability. Therefore, considering escape risk assessment and design methods to handle emergency situations when stochastic or systematic failures arise is an interesting work in both requirement and design phases.

2) Multicriteria decision method can also be used to solve the bicriteria between response time and reliability [25]. However, traditional methods, such as weighted combination method, efficacy coefficient method, ideal point method, multiplication division method, and minimax method, are imprecise in solving the nonlinear optimization problem. Intelligent methods, such as particle swarm optimization and genetic algorithms, are able to find the precise solution, but they are extremely time-consuming and not suitable for cost-sensitive automotive applications from a cost control perspective.

VI. Experimental Evaluation

A. Real-Life Benchmark

We use the real-life benchmark of an automotive case study adopted from [8]. This application consists of six functional 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 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}). The ECU number of the system is 16, and the failure rate of each task is within the range of 10^{-6}–$16 \times 10^{-6}/\mu$s. The WCETs of the tasks and the WCRTs of the messages are within the range of 100–400 μs. Considering that this paper presents two types of verification methods, we first compare these two algorithms with their existing counterparts.

Experiment 1: This experiment is conducted to compare the response time values of actual application for varying reliability requirements. To the best of our knowledge, MRCRG is a state-of-the-art method to minimize the resource consumption cost under the reliability requirement for a DAG-based distributed application on heterogeneous architectures [8]; we name such approach as minimizing response time with pessimism (MRTP) and compare it with FFSV1 in this study. Considering that the maximum reliability value is $R_{\max}(G) = 0.986271$, the reliability requirement is changed from 0.9 to 0.98 with 0.01 increment; 0.9–0.98 fall within the range of exposure E3 in ISO 26262. Considering that the actual reliability values are always larger than or equal to the corresponding reliability requirements using MRTP and FFSV1, we do not list the reliability values in this experiment. The response time values for varying reliability requirements are shown in Fig. 10 , where the x-axis and y-axis represent the reliability requirement and actual response time, respectively.

1) Although the response time values do not increase proportionally with the increase of reliability requirements, increasing the reliability will increase the response times on the whole, especially on higher reliability requirements. The curve indicates that response time minimization and reliability maximization are conflicting in general.

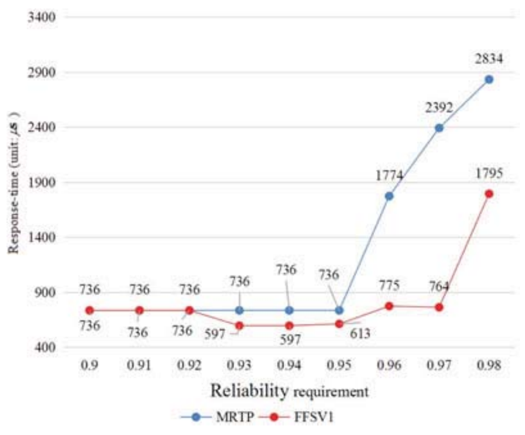


Fig. 10. Response time values of the real-life application for varying reliability requirements.

2) According to the analysis of the relationship between response time and reliability, an FFSV1-generated curve in Fig. 10 should be a Pareto optima curve. However, it is not. We can see that the response time values obtained with reliability requirements from 0.9 to 0.92 are higher than those with 0.93. As shortened development cycle is extremely critical for the development cost control of the automotive industry, the FFSV1 algorithm tradeoffs accuracy for saving time to ensure development progress, thus affecting the accuracy of the results. Therefore, FFSV1 cannot guarantee that the Pareto optima curve in Fig. 1 can be obtained.

3) FFSV1 generates shorter (or equal) response response time values than (or to) MRTP on all the cases. When reliability requirements are more than 0.95, FFSV1 is more prominent than MRTP in minimizing response time values. FFSV1 can reduce as much as 68.06% of response time compared with MRTP when the reliability requirement is 0.97 because FFSV1 uses a relative optimistic reliability pre-assignment toward unfair reliability usage among tasks; thus, FFSV1 can achieve shorter response time values than MRTP. Therefore, FFSV1 is confirmed as a good choice for the first type of functional safety verification (i.e., minimizing response time under the reliability requirement).

Experiment 2: This experiment is conducted to compare the reliability values of the actual application for varying response time requirements. To the best of our knowledge, the FFSV2 method has no similar research. We know that the HEFT algorithm is a well-studied assessment algorithm for a distributed application, and it can always satisfy the response time requirement. Therefore, we let the HEFT algorithm be the comparative algorithm for FFSV2. The lower bound of the application is $\text{LB}(G) = 736$ μs. Therefore, the response time requirement is changed from 736 to 1536 μs with 100 μs increments, as shown the x-axis in Fig. 11 . Considering that actual response time

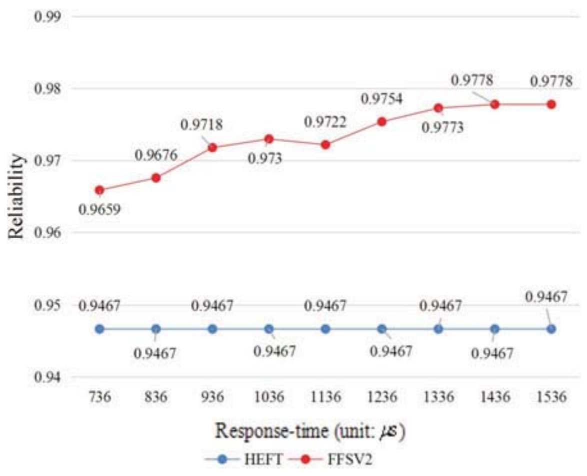


Fig. 11. Reliability values of the real-life application for varying response time requirements.

values are always less than or equal to the corresponding response time requirements using the HEFT algorithm and FFSV2, we do not list the response time values in this experiment. The reliability values for varying response time requirements are shown in Fig. 11, where the *x*-axis and *y*-axis represent the response time requirement and actual reliability, respectively.

1) As we expect, the HEFT-generated reliability values are fixed and do not change with the increment of response time requirements. The reason is that it is merely to obtain a lower bound and does not involve reliability enhancement.
2) FFSV2 always generates higher reliability values than the HEFT algorithm on all the cases. Even if the response time requirement is equal to the lower bound, FFSV2 still generates a higher reliability value of 0.019211 than the HEFT algorithm. The reliability values increase with the increase of response time requirement using FFSV2. When the response time requirement is 1536 μs, the reliability difference reaches 0.03126 between HEFT algorithm and FFSV2. The increased curve of FFSV2 indicates that larger response time requirement can form larger slacks, resulting in larger reliability values. The increased reliability values by FFSV2 confirm that FFSV2 is a good choice for the second type of functional safety verification (i.e., maximizing reliability under the response time requirement).

B. Synthetic Applications

Given the increasing complexity of automotive systems, future automotive applications are likely to include at least 50 tasks and at most 100 tasks. To further validate the effectiveness, we use additional simulated applications with the same real parameter values of the real-life application to observe the results. Randomly generated distributed applications can be generated by the task graph generator.[1] The ECU number and failure rates

[1] [Online]. Available: https://sourceforge.net/projects/taskgraphgen/

TABLE V
ACCEPTANCE RATIOS OF SYNTHETIC APPLICATIONS FOR VARYING NUMBERS OF TASKS

	$\lvert N\rvert = 50$	$\lvert N\rvert = 60$	$\lvert N\rvert = 70$	$\lvert N\rvert = 80$	$\lvert N\rvert = 90$	$\lvert N\rvert = 100$
MRTP [8]	75%	53%	57%	20%	31%	28%
HEFT [20]	75%	50%	57%	14%	17%	0%
BSH [10]	86%	72%	63%	54%	44%	40%
FFSV1	89%	63%	63%	49%	42%	42%
FFSV2	88%	73%	71%	53%	50%	47 %
UFFSV	**92%**	**80%**	**76%**	**57%**	**56%**	**50%**

of ECUs, WCETs of the tasks, and WCRTs of the messages are the same as the real-life applications.

Experiment 3: This experiment shows the acceptance ratios of synthetic applications for various tasks using six verification methods. We set the application parameters as follows: the average WCET is 200 μs, the communication-to-computation ratio is 1, the shape parameter is 1, and the heterogeneity factor is 0.5. The heterogeneity factor values are in the 0–1 scope in the task graph generator, where 0.1 and 1 are the lowest and highest heterogeneity factors, respectively. Task numbers are changed from 50 to 100 with 10 increments. The reliability requirement is changed from 0.9 with 0.01 increments until reaching the maximum reliability values. The response time requirement is changed from lower bound values with a 100 μs with the same number iteration of reliability requirement. Table V shows the acceptance ratios of synthetic applications for various tasks using the six verification methods.

1) The MRTP and HEFT algorithms have lower acceptance ratios than FFSV1 and FFSV2, respectively, under different task numbers. The advantages of FFSV1 and FFSV2 are apparent when comparing with the MRTP and HEFT algorithms, respectively. A special case is that the acceptance ratio using the HEFT algorithm is 0 when the task number is 100 because the HEFT algorithm merely obtains the maximum reliability of 0.893546, which is less than 0.9.
2) FFSV2 has slightly higher acceptance ratios than FFSV1 in most cases, but the results are not absolute. We cannot determine whether FFSV1 or FFSV2 is better because they are dual problems, and they adopt completely different ideas.
3) Although BSH can obtain relative higher acceptance ratios than MRTP and HEFT in most cases, the values are less than those by UFFSV; the reason is that BSH is also an approximate heuristic method and it cannot find a good enough solution. In addition, BSH consumes about 100 s for each case, where UFFSV can be finished in about 1 ms. The reason is that BSH has high time complexity of $O(|N| \times 2^{|U|})$. Considering that BSH cannot complete computation within an acceptable time when the ECU number reaches 70, it is not suitable for new-generation automotive systems.
4) The union verification method UFFSV shows higher acceptance ratios than FFSV1 and FFSV2 alone. The union verification is definitely better than individual independent verifications. UFFSV can improve as much as 37%

(when $N = 80$) and 50% (when $N = 100$) acceptance ratios than the MRTP and HEFT algorithms, respectively. In summary, the union verification method UFFSV shows good results for both real-life and synthetic distributed applications compared with other counterparts.

VII. Conclusion

We proposed a novel FFSV method called UFFSV for distributed automotive applications on heterogeneous architecture in the early design phase of industrial electronic software development. UFFSV is a fast heuristic method and it can shorten the application's development lifecycle. UFFSV can improve as much as 37% and 50% acceptance ratios than their existing counterparts, respectively, by experiments. If the verification does not pass, then analysts need to re-analyze the factors that affect the verification results. If the verification passes, then designers can present the cost optimization schemes based on the corresponding verification solutions. Therefore, in our future work, we will improve the system analysis to achieve the predetermined verification requirement if the verification does not pass and implement cost design optimization based on the results of UFFSV if the verification passes.

Acknowledgment

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

References

[1] G. Xie, G. Zeng, Z. Li, R. Li, and K. Li, "Adaptive dynamic scheduling on multi-functional mixed-criticality automotive cyber-physical systems," *IEEE Trans. Veh. Technol.*, vol. 66, no. 8, pp. 6676–6692, Aug. 2017.

[2] *Road Vehicles-Functional Safety*, ISO Std. 26262, 2011.

[3] S. S. Williamson, A. K. Rathore, and F. Musavi, "Industrial electronics for electric transportation: Current state-of-the-art and future challenges," *IEEE Trans. Ind. Electron.*, vol. 62, no. 5, pp. 3021–3032, May 2015.

[4] A. Monot, N. Navet, B. Bavoux, and F. Simonot-Lion, "Multisource software on multicore automotive ECUs—Combining runnable sequencing with task scheduling," *IEEE Trans. Ind. Electron.*, vol. 59, no. 10, pp. 3934–3942, Oct. 2012.

[5] J. Gan, P. Pop, and J. Madsen, "Tradeoff analysis for dependable real-time embedded systems during the early design phases," Ph.D. dissertation, Dept. Informat. Math. Model., Tech. Univ. Denmark, Kongens Lyngby, Denmark, 2014.

[6] 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.

[7] Z. Gu, G. Han, H. Zeng, and Q. Zhao, "Security-aware mapping and scheduling with hardware co-processors for flexray-based distributed embedded systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 3044–3057, Oct. 2016.

[8] G. Xie, Y. Chen, Y. Liu, Y. Wei, R. Li, and K. Li, "Resource consumption cost minimization of reliable parallel applications on heterogeneous embedded systems," *IEEE Trans. Ind. Informat.*, vol. 13, no. 4, pp. 1629–1640, Aug. 2017.

[9] M. Glinz, "On non-functional requirements," in *Proc. 15th IEEE Int. Requirements Eng. Conf.*, 2007, pp. 21–26.

[10] A. Girault and H. Kalla, "A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate," *IEEE Trans. Dependable Secure Comput.*, vol. 6, no. 4, pp. 241–254, Oct./Dec. 2009.

[11] G. Xie *et al.*, "Minimizing redundancy to satisfy reliability requirement for a parallel application on heterogeneous service-oriented systems," *IEEE Trans. Serv. Comput.*, vol. PP, no. 99, pp. 1–11, 2017, doi: 10.1109/TSC.2017.2665552.

[12] 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.

[13] Ö. Ciftcioglu, "Verification and validation in the life-cycle of real-time software development," in *Real Time Computing*. New York, NY, USA: Springer, 1994, pp. 678–679.

[14] N. Doddappa, D. Oreper, and J. D. Vincent, "Method and system for feature automation," U.S. Patent 20 100 235 807 A1, Sep. 16 2010.

[15] V. Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *IEEE Trans. Ind. Informat.*, vol. 9, no. 3, pp. 1234–1249, Aug. 2013.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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, Mar. 2002.

[21] Z. Gu, C. Wang, M. Zhang, and Z. Wu, "WCET-aware partial control-flow checking for resource-constrained real-time embedded systems," *IEEE Trans. Ind. Electron.*, vol. 61, no. 10, pp. 5652–5661, Oct. 2014.

[22] J. D. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, Jun. 1975.

[23] A. Brandsater, "Risk assessment in the offshore industry," *Saf. Sci.*, vol. 40, nos. 1–4, pp. 231–269, Jul. 2002.

[24] J. D. Sime, "An occupant response shelter escape time (ORSET) model," *Saf. Sci.*, vol. 38, no. 2, pp. 109–125, Jul. 2001.

[25] I. Linkov, A. Varghese, S. Jamil, T. P. Seager, G. Kiker, and T. Bridges, *Multi-Criteria Decision Analysis: A Framework for Structuring Remedial Decisions at Contaminated Sites*. Dordrecht, The Netherlands: Springer, 2004.

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 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 research interests include embedded and cyber-physical systems, parallel and distributed systems, and software engineering and methodology.

Dr. Xie is a member of the ACM and China Computer Federation. He was the recipient of the Best Paper Award from ISPA 2016. He

Gang Zeng (M'05) received the Ph.D. degree in information science from Chiba University, Chiba, Japan, in 2006.

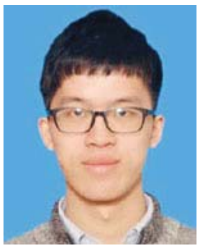
He is an Associate Professor with the Graduate School of Engineering, Nagoya University, Nagoya, Japan. From 2006 to 2010, he was a Researcher, and then an Assistant Professor in the Center for Embedded Computing Systems, the Graduate School of Information Science, Nagoya University. His research interests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

Dr. Zeng is a member of the Information Processing Society of Japan.

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

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

Dr. Liu is a member of the China Computer Federation.



Jia Zhou is currently working toward the Ph.D. degree at Hunan University, Changsha, China.

His research interests include distributed embedded systems and safety-critical cyber-physical systems.

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

He is a Professor of computer science and electronic engineering with Hunan University, Changsha, China. He is the Director of 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, Changsha, China. His major research interests include computer architectures, embedded computing systems, cyber-physical systems, and Internet of Things.

Dr. Li is a member of the council of the China Computer Federation 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 State University of New York Distinguished Professor of computer science. He has published more than 500 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

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