Resource-Cost-Aware Fault-Tolerant Design Methodology for End-to-End Functional Safety Computation on Automotive Cyber-Physical Systems

GUOQI XIE, Hunan University
GANG ZENG, Nagoya University
JIYAO AN and RENFA LI, Hunan University
KEQIN LI, State University of New York

Automotive functional safety standard ISO 26262 aims to avoid unreasonable risks due to systematic failures and random hardware failures caused by malfunctioning behavior. Automotive functions involve distributed end-to-end computation in automotive cyber-physical systems (ACPSs). The automotive industry is highly cost-sensitive to the mass market. This study presents a resource-cost-aware fault-tolerant design methodology for end-to-end functional safety computation on ACPSs. The proposed design methodology involves early functional safety requirement verification and late resource cost design optimization. We first propose the functional safety requirement verification (FSRV) method to verify the functional safety requirement consisting of reliability and response time requirements of the distributed automotive function during the early design phase. We then propose the resource-cost-aware fault-tolerant optimization (RCFO) method to reduce the resource cost while satisfying the functional safety requirement of the function during the late design phase. Finally, we perform experiments with real-life automotive and synthetic automotive functions. Findings reveal that the proposed RCFO and VFSR methods demonstrate satisfactory resource cost reduction compared with other methods while satisfying the functional safety requirement.

CCS Concepts: • Computer systems organization → Embedded and cyber-physical systems; Dependable and fault-tolerant systems and networks;

Additional Key Words and Phrases: Automotive cyber-physical systems (ACPSs), functional safety, reliability, response time, resource cost

This work was supported in part by the National Key R&D Program of China under Grant 2017YFB0202901, Grant 2017YFB0202905, the National Natural Science Foundation of China under Grant 61702172, Grant 61672217, the Natural Science Foundation of Hunan Province under Grant 2018JJ3076, the CCF-Venustech Open Research Fund under Grant CCF-VenustechRP2017012, the Open Research Project of the State Key Laboratory of Synthetical Automation for Process Industries (SAPI), Northeastern University, China under Grant PAL-N201803, and the Fundamental Re-search Funds for the Central Universities.

Authors’ addresses: G. Xie, J. An, and R. Li (corresponding author), College of Computer Science and Electronic Engineering, Hunan University, Key Laboratory for Embedded and Network Computing of Hunan Province, Changsha, 410082, China; email: [xgqman, jt_anbob, lirenfa]@hnu.edu.cn; G. Zeng, Graduate School of Engineering, Nagoya University, Nagoya, 4648603, Japan; email: sogo@ertl.jp; K. Li, Department of Computer Science, State University of New York, New Paltz, NY 12561; email: lik@newpaltz.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 Association for Computing Machinery.
2378-962X/2018/08-ART4 $15.00
https://doi.org/10.1145/3162052

ACM Transactions on Cyber-Physical Systems, Vol. 3, No. 1, Article 4. Publication date: August 2018.
1 INTRODUCTION

1.1 Background

Automotive embedded systems are complex heterogeneous distributed embedded systems, which contain up to 70 heterogeneous electronic control units (ECUs) distributed on network buses; these systems support the end-to-end computation for distributed automotive functions [26]. The implementation of distributed automotive functions depends on the interaction, feedback, and coordination of ECUs, 360° sensors, and actuators through networks [9]. A function model to describe the distributed end-to-end computations in automobiles is called a task graph [26]. An end-to-end distributed automotive function is released by receiving collected data from the sensor and is completed by sending the performing action to the actuator in automotive embedded systems. Therefore, the task graph is restricted to be directed and acyclic and is called a directed acyclic graph (DAG) [26, 30, 31], where the nodes represent tasks, and the edges represent the communication messages between tasks. Examples of active safety functions are brake-by-wire and adaptive cruise control [6, 26]. The inherent heterogeneity, interaction, and diverse nature of automotive embedded systems require the joint and tight interaction between the cyber (networked computational) and physical worlds [9, 26]. Thus, automotive embedded systems are typical cyber-physical systems (CPSs) and are called automotive cyber-physical systems (ACPSs) [4, 9, 21, 26].

Since the invention of the automobile, people have aimed for safe driving. The earliest safety belts, later airbags, and other passive safety measures have saved millions of lives. The later-developed antilock braking system (ABS) and other active safety functions significantly enhanced the safety of automobiles. Safety must always be ensured in the use of automobiles. To cope with the growing safety risks of cars, the road vehicles functional safety standard, ISO 26262, was officially released in 2011 to adapt to automotive functional safety [12]. Functional safety refers to the absence of unreasonable risk due to hazards caused by malfunctioning behavior of electrical and electronic (E/E) systems [12]. Hazards include systemic failures and random hardware failures in ACPSs. Missing deadlines and abnormal execution are considered as typical systematic and random hardware failures, respectively. In other words, both response time and reliability are important functional safety properties. A functional safety requirement is the combination of 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). The functional safety requirement must be simultaneously satisfied learning from ISO 26262 [12]. If response time and reliability requirements cannot be satisfied simultaneously, then the function is incorrect and it would cause fatal injuries to people [26].

1.2 Motivation

CPSs frequently require massive redundancy to satisfy a function’s requirement for high reliability [28]. Fault tolerance is an effective way to satisfy the functional safety requirement, and primary-backup replication is an important software fault-tolerant technique [24]. However, replication-based fault-tolerance technology inevitably generates redundancy for high reliability and therefore, causes negative influences on the satisfaction of the response time requirement and resource cost optimization required for an automotive function.
Fault tolerance can cause an excessively long response time of the end-to-end distributed function because increasing the reliability also increases the response time of the function [8, 24] (i.e., response time minimization and reliability maximization are conflicting processes). Missing the response time requirement is considered a typical systematic failure that will cause fatal injury and accidents [26].

Fault tolerance can increase the resource cost required for an automotive function because of redundancy. Taking into account the limited resources of embedded systems, resource consumption should be reduced as much as possible while satisfying the reliability requirement [23]. In addition, the automotive industry is cost-sensitive to the mass market; thus, increasing the resource cost is not an acceptable option [27].

Therefore, reducing the resource cost is desirable in ACPS, but it faces the limit of satisfying the functional safety requirement. To solve this problem, we aim to study the resource-cost-aware fault-tolerant design methodology for end-to-end functional safety computation on ACPSs.

### 1.3 Our Contributions

A development life cycle of safety-critical systems usually involves analysis, design, implementation, and testing phases [23]. The current study focuses on the design phase, which is divided into the early functional safety verification and late resource cost optimization phases. The main contributions of this study are as follows:

1. Considering that the functional safety requirement must be satisfied before the resource cost optimization, we propose the verifying functional safety requirement (VFSR) method to verify whether the functional safety requirement of an end-to-end distributed automotive function can be satisfied in the given ACPS platform during the early design phase. As long as the functional safety requirement is satisfied, then the late resource cost optimization can be processed. VFSR is a heuristic method, which divides the problem into three subproblems, namely, prioritizing tasks, satisfying reliability requirement, and minimizing response time. If the actual response time obtained by VFSR is less than or equal to the response time requirement, then the verification is passed and the functional safety requirement of the function can be satisfied.

2. If the functional safety requirement is passed by VFSR during the early design phase, we then propose the resource-cost-aware fault-tolerant optimization (RCFO) method to reduce the resource cost while satisfying the functional safety requirement of the end-to-end distributed automotive function during the late design phase. RCFO is also a heuristic method that divides the problem into four subproblems, namely, prioritizing tasks, satisfying the reliability requirement, satisfying the response time requirement, and optimizing resource cost.

3. Experiments with both real-life automotive and synthetic automotive functions are performed. Findings show that the proposed design methods demonstrate satisfactory resource cost reduction compared with other counterparts while satisfying the functional safety requirement.

The remainder of this article is organized as follows. Section 2 reviews the related literature. Section 3 presents the models and problem statement. Section 4 presents the VFSR method. Section 5 presents the RCFO method. Section 6 evaluates the proposed methods with experiments. Section 7 concludes this study.
2 RELATED WORK

Numerous studies have investigated the cost problem of an end-to-end distributed function in various computing environments. Heuristic methods are widely used methods in practice because the problem of mapping tasks to multiple processors for an optimal schedule is NP-hard [20]. Resource-cost-aware scheduling methods have been proposed to minimize the resource cost while satisfying the response time requirement [1, 5, 29], or to minimize the response time while satisfying the resource cost requirement [22]. However, the aforementioned works either focus on homogeneous systems or do not consider the communication between tasks. ACPSs are typical heterogeneous distributed embedded systems, where the communication between tasks mapped to different ECUs is performed through message passing over the bus.

Resource cost minimization has been widely studied in embedded systems. In [16], the authors solved the cost minimization while satisfying the hard or soft response time requirement for an end-to-end distributed function on heterogeneous embedded systems. In [13, 14], the authors solved the cost minimization while satisfying the response time requirement for an end-to-end distributed function by efficient task assignment also on heterogeneous embedded systems. In [15], the authors developed the resource cost model and analyzed and verified the resource cost scenarios for embedded systems using priced timed automata. Aside from resource, cost also involves hardware cost and development cost in the embedded system design. In [11], the authors proposed hardware cost minimization for an end-to-end distributed function to satisfy the response time and security requirements by presenting integer linear programming and heuristics, respectively. In [7, 18], the authors proposed development cost minimization for end-to-end distributed functions to satisfy the response time requirement by presenting genetic-method-based and tabu-search-based metaheuristics, respectively.

With regard to functional safety, resource cost minimization while satisfying the reliability requirement has been studied recently. In [23], we studied the problem of resource cost minimization for an end-to-end distributed function while satisfying the reliability requirement on heterogeneous embedded systems without using fault tolerance. The main limitation of a non-fault-tolerant method is that the high reliability requirement for a large end-to-end distributed function is difficult to satisfy. In [33] and [32], the authors presented the maximum reliability (MaxRe) and least resources to meet the reliability requirement (RR) methods to reduce the resource cost for an end-to-end distributed function by minimizing the task redundancy on heterogeneous distributed systems using fault tolerance. In [24], we also studied the same problem as MaxRe and RR and proposed the heuristic replication for redundancy minimization (HRRM) method, which exhibited significant improvement in resource cost reduction. However, all the aforementioned studies merely consider the reliability requirement and do not take the response time requirement into account. As pointed out in Section 1.1, the automotive functional safety requirement must simultaneously include the reliability and response time requirements.

To the best of our knowledge, the deadline-required RR (DRR) method presented in [32] is the state-of-the-art method to minimize resource cost while satisfying the response time and reliability requirements (i.e., the functional safety requirement) for an end-to-end distributed function by replication-based fault tolerance on heterogeneous distributed systems. The DRR method can also be applied to ACPSs because they have similar models and the same objectives. We summarize that DRR provides a basic solution for this problem by solving four critical subproblems: prioritizing tasks, satisfying the reliability requirement, satisfying the response time requirement, and optimizing the resource cost. However, DRR is prone to scheduling rejection (i.e., the functional safety requirement is not satisfied) in most cases; even if DRR accepts the scheduling, it is not resource cost aware; i.e., the resource cost is not reduced significantly (refer to Section 4.2 for more
Table 1. Important Notations in This Study

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_{i,k} )</td>
<td>WCET of the task ( n_i ) on the ECU ( u_k )</td>
</tr>
<tr>
<td>( c_{i,j} )</td>
<td>WCRT of the message ( m_{[i,j]} ) from the tasks ( n_i ) to ( n_j )</td>
</tr>
<tr>
<td>( \text{rank}_{u_k}(n_i) )</td>
<td>Upward rank value of the task ( n_i )</td>
</tr>
<tr>
<td>(</td>
<td>X</td>
</tr>
<tr>
<td>( \lambda_k )</td>
<td>Failure rate per time unit of the ECU ( u_k )</td>
</tr>
<tr>
<td>( r_k )</td>
<td>Resource cost failure rate per time unit of the ECU ( u_k )</td>
</tr>
<tr>
<td>( num_i )</td>
<td>Number of replicas of the task ( n_i )</td>
</tr>
<tr>
<td>( n^\beta_i )</td>
<td>( \beta )th replica of the task ( n_i )</td>
</tr>
<tr>
<td>( u_{pr(n^\beta_i)} )</td>
<td>Assigned ECU of the replica ( n^\beta_i )</td>
</tr>
<tr>
<td>( n_{seq(y)} )</td>
<td>( y )th assigned task of the function</td>
</tr>
<tr>
<td>( R(n_i, u_k) )</td>
<td>Reliability of the task ( n_i ) on the ECU ( u_k )</td>
</tr>
<tr>
<td>( R(n_i) )</td>
<td>Reliability of the task ( n_i )</td>
</tr>
<tr>
<td>( R(G) )</td>
<td>Reliability of the function ( G )</td>
</tr>
<tr>
<td>( RT(G) )</td>
<td>Response time of the function ( G )</td>
</tr>
<tr>
<td>( EST(n_i, u_k) )</td>
<td>Earliest start time of the task ( n_i ) on the ECU ( u_k )</td>
</tr>
<tr>
<td>( EFT(n_i, u_k) )</td>
<td>Earliest finish time of the task ( n_i ) on the ECU ( u_k )</td>
</tr>
<tr>
<td>( LFT(n_i, u_k) )</td>
<td>Latest finish time of the task ( n_i ) on the ECU ( u_k )</td>
</tr>
<tr>
<td>( RC(n_i, u_k) )</td>
<td>Resource cost of the task ( n_i ) on the ECU ( u_k )</td>
</tr>
<tr>
<td>( RC(n_i) )</td>
<td>Resource cost of the task ( n_i )</td>
</tr>
<tr>
<td>( RC(G) )</td>
<td>Resource cost of the function ( G )</td>
</tr>
<tr>
<td>( R_{req}(G) )</td>
<td>Reliability requirement of the function ( G )</td>
</tr>
<tr>
<td>( R_{req}(n_i) )</td>
<td>Reliability requirement of the task ( n_i )</td>
</tr>
<tr>
<td>( R_{up_req}(n_i) )</td>
<td>Upper bound on reliability requirement of the task ( n_i )</td>
</tr>
<tr>
<td>( RT_{req}(G) )</td>
<td>Response time requirement of the function ( G )</td>
</tr>
<tr>
<td>( RT_{req}(n_i) )</td>
<td>Response time requirement of the task ( n_i )</td>
</tr>
</tbody>
</table>

details). Therefore, DRR is not suitable for automotive functions, where safety is critical and cost is sensitive.

3 MODELS AND PROBLEM STATEMENT

Table 1 lists important notations and their definitions that are used in this study.

3.1 System Architecture

This study considers an integrated automotive electrical and electronic (E/E) architecture as a controller area network (CAN) cluster, where more than four or five CAN buses are integrated by a central gateway and several ECUs are mounted on each CAN bus [26]. Considering that physical processes are compositions of many parallel processes, we use the same configuration as [26], in which some ECUs connect to several sensors and other ECUs connect to several actuators, as shown in Figure 1. In this situation, partial ECUs can release the function by receiving the collected data from the sensors, and other partial ECUs can complete the function by sending the performing action to the actuators. When a task is executed completely in an ECU, it will send messages to all its successor tasks, which may be located in different ECUs. For example, task \( n_1 \) is executed on ECU \( u_1 \) (\( ECU_1 \)), and then it sends a message \( m_{1,2} \) to its successor task \( n_2 \) located in \( u_6 \) (\( ECU_6 \)) (see Figure 1). We let \( U = \{ u_1, u_2, \ldots, u_{|U|} \} \) represent a set of heterogeneous ECUs in the platform, where \( |U| \) represents the size of set \( U \). For any set \( X \), this study uses \(|X|\) to denote its size.
3.2 Function Model

An end-to-end distributed function is represented by a DAG $G = (N, W, M, C)$ \cite{19, 23, 24, 26, 32, 33}, where the related parameters are described below.

1. $N$ represents a set of nodes in $G$, and each node $n_i \in N$ represents a task. $\text{pred}(n_i)$ represents the set of the immediate predecessor tasks of $n_i$, $\text{succ}(n_i)$ represents the set of the immediate successor tasks of $n_i$. The task that has no predecessor task is denoted as $n_{\text{entry}}$, and the task that has no successor task is denoted as $n_{\text{exit}}$. Considering that an automotive function (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 $n_{\text{entry}}$ or $n_{\text{exit}}$ tasks may exist. To adapt to the function 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 an $|N| \times |U|$ matrix, where $w_{i,k}$ denotes the worst-case execution time (WCET) of $n_i$ runs on $u_k$. Each task $n_i \in N$ has different WCET values on different ECUs owing to the heterogeneity of ECUs. The WCET of a task is the maximum execution time among all possible real execution time values when the task is executed on a specific ECU. All the WCETs of the tasks are known and determined through analysis methods performed (i.e., WCET analysis \cite{3}) during the analysis phase.

3. The communication between tasks mapped to different ECUs is performed through message passing over the bus. $M$ 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 $m_{i,j}$ if $n_i$ and $n_j$ are not assigned to the same ECU. The WCRT of a message is the maximum response time among all possible real response time values when the message is transmitted on a specific hardware platform. If $n_i$ and $n_j$ are assigned to the same ECU, then the communication time is 0. The above communication scheme also complies with the automotive open system architecture (AUTOSAR) standard. All the WCRTs of the messages are also known and determined through analysis methods performed (i.e., WCRT analysis \cite{25}) during the analysis phase.

Fig. 1. ACPS architecture \cite{26}.
Scheduling in ACPS can be either preemptive (e.g., OSEKTime) or nonpreemptive (e.g., eCos) [26]. Considering that many DAG-based distributed function scheduling algorithms generally use nonpreemptive scheduling [23, 26, 27], we consider nonpreemptive scheduling for ECUs in this study. Of course, the method of this article can also be applied to preemptive scheduling.

Figure 2 shows a motivating example of an end-to-end distributed function. Table 2 is a matrix of WCET in Figure 2. The example shows 10 tasks executed on three ECUs \{u_1, u_2, u_3\}. The weight 14 of n_1 and u_1 in Table 2 represents the WCET denoted by \( w_{1,1} = 14 \). We see that the same task has different WCETs on different ECUs because of the heterogeneity of the ECUs [23, 24]. The weight 18 of edge (Figure 2) between n_1 and n_2 represents the WCRT denoted as \( c_{1,2} \) if n_1 and n_2 are not assigned to the same ECU.

### 3.3 Reliability Model

The two major types of failures are transient failure and permanent failure [8, 23]. The present study considers the transient failure of ECUs because ISO 26262 only combines the random hardware failures and reliability [12]. ISO 26262 specifies that random hardware failures occur unpredictably during the life cycle of a hardware but follows a probability distribution [12]. Generally, the transient failure of a task in an end-to-end distributed function follows the Poisson distribution [8, 23]. The reliability of an event in unit time \( t \) is denoted by \( R(t) = e^{-\lambda t} \), where \( \lambda \) is the failure rate per time unit of an ECU [17]. We use \( \lambda_k \) to represent the constant failure rate per time unit of the ECU \( u_k \). The reliability of \( n_i \) executed on \( u_k \) in its execution time is denoted by

\[
R(n_i, u_k) = e^{-\lambda_k w_{i,k}},
\]

and the failure probability for \( n_i \) without using the replication is

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

Similar to [24, 32, 33], the present study also uses the active replication scheme to implement fault tolerance. For the active replication scheme, each task is simultaneously replicated on several ECUs, and the task will succeed if at least one of the ECUs does not fail. For the passive scheme, whenever an ECU fails, the task will be rescheduled to proceed on a backup ECU. Two replicas (including primary and backups) of the same task cannot be assigned to the same ECU in the

---

**Table 2. WCETs of Tasks on Different ECUs of the Motivating End-to-End Distributed Function**

<table>
<thead>
<tr>
<th>Task</th>
<th>( u_1 )</th>
<th>( u_2 )</th>
<th>( u_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_1</td>
<td>14</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>n_2</td>
<td>13</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>n_3</td>
<td>11</td>
<td>13</td>
<td>19</td>
</tr>
<tr>
<td>n_4</td>
<td>13</td>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td>n_5</td>
<td>12</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>n_6</td>
<td>13</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>n_7</td>
<td>7</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>n_8</td>
<td>5</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>n_9</td>
<td>18</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>n_{10}</td>
<td>21</td>
<td>7</td>
<td>16</td>
</tr>
</tbody>
</table>

---

**Fig. 2. Motivating example of an end-to-end distributed function with 10 tasks [19, 23, 24].**
active replication scheme. Considering that each task has a certain number of replicas with the active replication scheme, we define \( \text{num}_i \) \((\text{num}_i \leq |U|)\) as the number of replicas of \( n_i \). Thus, the replica set of \( n_i \) is \( \{n^1_i, n^2_i, \ldots, n^{\text{num}_i}_i\} \), where \( n^1_i \) is the primary and the others are the backups.

As long as one replica of \( n_i \) is successfully completed, then we can recognize that no failure occurs for \( n_i \), and the reliability of \( n_i \) is updated to

\[
R(n_i) = 1 - \prod_{\beta=1}^{\text{num}_i} \left( 1 - R(n^\beta_i, u_{pr(n^\beta_i)}) \right),
\]

(3)

where \( u_{pr(n^\beta_i)} \) represents the assigned ECU of the \( \beta \)th replica \( n^\beta_i \). Thus, the reliability of the end-to-end distributed function is

\[
R(G) = \prod_{n_i \in N} R(n_i).
\]

(4)

Similar to [23, 27], this study only considers ECU failures and assumes reliable communication because CAN buses provide high fault-tolerance.

### 3.4 Resource Cost Model

The priced timed automata defines the computation resource rate per time unit for task execution, which means that the computation resource grows constantly with the computation resource rate in an ECU to access data stored in memory [2, 15, 23]. The computation resource cost of the \( n_i \) on \( u_k \) is the product of the WCET of \( n_i \) and the corresponding computation resource rate of \( u_k \). It is calculated by

\[
RC(n_i, u_k) = w_{i,k} \times r_k,
\]

(5)

where \( r_k \) represents the resource cost rate of the ECU \( u_k \). The priced timed automata also defines the communication resource cost; we only focus on the computation resource cost in this study. Therefore, the resource cost of the function is the product of the WCETs of all the replicas of tasks and the corresponding resource cost rates of ECUs; that is,

\[
RC(G) = \sum_{n_i \in N} RC(n_i) = \sum_{n_i \in N} \left( \sum_{\beta=1}^{\text{num}_i} RC(n^\beta_i, u_k) \right) = \sum_{n_i \in N} \left( \sum_{\beta=1}^{\text{num}_i} w_{i,pr(n^\beta_i)} \times r_{pr(n^\beta_i)} \right).
\]

(6)

### 3.5 Problem Statement

The problem to be addressed is to assign replicas to ECUs for each task to minimize the resource cost:

\[
RC(G) = \sum_{n_i \in N} RU(n_i),
\]

subject to the response time requirement:

\[
RT(G) = \max_{\beta \in [1, \text{num}_\text{exit}]} \{AFT(n^\beta_i, \text{exit})\} \leq RT_{\text{req}}(G),
\]

and the reliability requirement:

\[
R(G) = \prod_{n_i \in N} R(n_i) \geq R_{\text{req}}(G),
\]

for \( \forall i : 1 \leq i \leq |N| \).

Given that the problem of mapping tasks to multiple processors (ECUs) is NP-hard [20], the above-stated problem is an NP-hard optimization problem.
4 VERIFYING FUNCTIONAL SAFETY REQUIREMENT

Before optimizing the resource cost during the late design phase, we must verify whether the functional safety requirement can be satisfied in the given ACPS platform during the early design phase. If the functional safety requirement cannot be satisfied, then the following resource cost optimization is infeasible and an unnecessary late design effort can be avoided. Otherwise, it is feasible, and the optimization can proceed based on the verification result during the late design phase. We first verify the functional safety requirement in this section and then optimize the resource cost in the next section.

4.1 Prioritizing Tasks

Verifying the functional safety requirement is also an NP-hard optimization problem. We use heuristic list scheduling to solve this problem in this study. List scheduling is the most well-known heuristic method for an end-to-end distributed function. This method must solve the prioritizing task problem before the task assignment. Prioritizing task means determining the priorities of tasks. Ordering tasks in descending order of the upward rank value \( \text{rank}_u(n_i) \) is considered the de facto prioritizing task criterion \([19, 24, 26]\) and is used by DRR \([32]\). It is calculated by

\[
\text{rank}_u(n_i) = \overline{w_i} + \max_{n_j \in \text{succ}(n_i)} \{ c_{i,j} + \text{rank}_u(n_j) \}, \tag{7}
\]

where \( \overline{w_i} \) represents the average WCET of task \( n_i \) and is calculated by

\[
\overline{w_i} = \left( \sum_{k=1}^{\left| U \right|} w_{i,k} \right) / \left| U \right|.
\]

Table 3 shows the upward rank values of all the tasks of the motivating function in Figure 2. \( n_i \) can be allocated to ECU only if all the predecessors of \( n_i \) have been 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 \( n_j \), then \( n_i \) does not necessarily take precedence for \( n_j \) to be assigned. Finally, the task priority order in the motivating function \( G \) is \( \{ n_1, n_3, n_4, n_2, n_5, n_6, n_9, n_7, n_8, n_{10} \} \).

After the priorities of tasks are determined, we must solve the task assignment problem of minimizing response time while satisfying the reliability requirement to verify the functional safety requirement. This problem is decomposed into two subproblems: satisfying the reliability requirement and minimizing response time. The details are explained in the following sections.

4.2 Satisfying Reliability Requirement

In \([32]\), DRR transfers the function’s reliability requirement to each task, and each task needs to satisfy its reliability requirement, which is calculated as follows:

\[
\begin{align*}
R_{\text{req}}(n_{\text{entry}}) &= \left\lceil \frac{1}{\left| N \right|} R_{\text{req}}(G) \right\rceil, \\
R_{\text{req}}(n_{\text{seq}(y)}) &= \left\lceil \frac{1}{\left| N \right|} \prod_{x=1}^{y-1} R_{\text{req}}(n_{\text{seq}(x)}) \right\rceil, \\
R_{\text{req}}(n_{\text{seq}(y)}) &= \left\lceil \frac{1}{\left| N \right|} \prod_{x=1}^{y-1} R_{\text{req}}(n_{\text{seq}(x)}) \right\rceil.
\end{align*}
\tag{8}
\]

Table 3. Upward Rank Values for Tasks of the Motivating Function

<table>
<thead>
<tr>
<th>Task</th>
<th>( n_1 )</th>
<th>( n_2 )</th>
<th>( n_3 )</th>
<th>( n_4 )</th>
<th>( n_5 )</th>
<th>( n_6 )</th>
<th>( n_7 )</th>
<th>( n_8 )</th>
<th>( n_{10} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{rank}_u(n_i) )</td>
<td>108</td>
<td>77</td>
<td>80</td>
<td>80</td>
<td>69</td>
<td>63.3</td>
<td>42.7</td>
<td>35.7</td>
<td>44.3</td>
</tr>
</tbody>
</table>

ACM Transactions on Cyber-Physical Systems, Vol. 3, No. 1, Article 4. Publication date: August 2018.
where $n_{\text{seq}(y)}$ represents the $y$th assigned task. In other words, for the entry task, its reliability requirement is $\sqrt[n]{R_{\text{req}}(G)}$; for the rest of the tasks (i.e., nonentry tasks), their reliability requirements are calculated continuously based on the actual reliability achieved by previous assignments. However, as pointed out in [24], the reliability requirements calculated by Equation (8) of high-priority tasks are significantly lower than those of the low-priority tasks. That is, the actual reliability requirements exhibit unfairness among tasks, such that the DRR method generates unnecessary redundancy while satisfying the function’s reliability requirement. In this study, the same upper bound on reliability requirement $R_{\text{up, req}}(n_i)$ proposed in [24] is used as the task’s reliability requirement; that is,

$$R_{\text{up, req}}(n_i) = \sqrt[n]{R_{\text{req}}(G)}. \quad (9)$$

The main advantage is that $\sqrt[n]{R_{\text{req}}(G)}$ is considered as the maximum reliability requirement. Any actual reliability requirements exceeding $\sqrt[n]{R_{\text{req}}(G)}$ are inefficient.

We then have the following heuristic strategy as [24]: assume that the task to be assigned is $n_{\text{seq}(y)}$ ($n_{\text{seq}(y)}$ represents the $y$th assigned task as mentioned earlier); then $\{n_{\text{seq}(1)}, n_{\text{seq}(2)}, \ldots, n_{\text{seq}(y-1)}\}$ represents the task set with assigned tasks, and $\{n_{\text{seq}(y+1)}, n_{\text{seq}(y+2)}, \ldots, n_{\text{seq}(N)}\}$ represents the task set with unassigned tasks. To ensure that the reliability of the function is satisfied at each task assignment, we presuppose that each task in $\{n_{\text{seq}(y+1)}, n_{\text{seq}(y+2)}, \ldots, n_{\text{seq}(N)}\}$ is assigned to the ECU with reliability value on the upper bound (Equation (9)). Thus, when assigning $n_{\text{seq}(y)}$, the function’s reliability requirement is

$$R_{\text{req}}(G) = \prod_{x=1}^{y-1} R(n_{\text{seq}(x)}) \times R_{\text{req}}(n_{\text{seq}(y)}) \times \prod_{z=y+1}^{N} R_{\text{up, req}}(n_{\text{seq}(z)}).$$

Then, the reliability requirement for the task $n_{\text{seq}(y)}$ should be

$$R_{\text{req}}(n_{\text{seq}(y)}) = \frac{R_{\text{req}}(G)}{\prod_{x=1}^{y-1} R(n_{\text{seq}(x)}) \times \prod_{z=y+1}^{N} R_{\text{up, req}}(n_{\text{seq}(z)})}.$$

### 4.3 Minimizing Response Time

In [32], DRR transfers the function’s response time requirement to each task, and each task merely needs to satisfy the individual response time requirement, which is calculated as follows:

$$\begin{align*}
RT_{\text{req}}(n_{\text{exit}}) &= RT_{\text{req}}(G) \\
RT_{\text{req}}(n_i) &= \min_{n_j \in \text{succ}(n_i)} \left\{ RT_{\text{req}}(n_j) - \bar{w}_{ij} - c_{ij} \right\}.
\end{align*} \quad (11)$$

That is, for the exit task, its reliability requirement is equal to that of the function; for the nonexit tasks, their response time requirements are computed according to a traversal of the function in reverse topological order. However, DRR is prone to scheduling rejection in most cases because $RT_{\text{req}}(n_i)$ is too small for high-priority tasks. In fact, for an end-to-end distributed function, all the tasks share a response time requirement; as long as the exit task can be completed within the response time requirement, the function can satisfy its response time requirement. In other words, the response time requirements of tasks need not be considered, and we just focus on whether the final response time of the function satisfies its response time requirement. Therefore, to satisfy the response time requirement as much as possible, we need to finish the function execution in the shortest time while satisfying its reliability requirement.

On the above analysis, we aim to obtain a short response time of the end-to-end distributed function as soon as possible while satisfying its reliability requirement and directly compare the actual response time with the response time requirement to verify the functional safety requirement.
We first let the attributes $EST(n_i^\beta, u_k)$ and $EFT(n_i^\beta, u_k)$ represent the earliest start time (EST) and the earliest finish time (EFT), respectively, of the replica $n_i^\beta$ on the ECU $u_k$. $EFT(n_i^\beta, u_k)$ is considered as the de facto task assignment criterion because it satisfies the local optimum of each precedence-constrained task by using the greedy policy [19, 32, 33]. Given that the strict schedule is used, the aforementioned attributes are calculated as follows:

\[
\begin{aligned}
EST(n_{\text{entry}}^\beta, u_k) &= 0 \\
EST(n_i^\beta, u_k) &= \max \left\{ \text{avail}[k], \max_{n_h \in \text{pred}(n_i), \alpha \in [1, \text{num}_h]} \left\{ \text{AFT}(n_i^\alpha) + c_{h,i}' \right\} \right. \\
\end{aligned}
\]

and

\[
EFT(n_i^\beta, u_k) = EST(n_i^\beta, u_k) + w_{i,k}.
\]

$avail[k]$ is the earliest available time when ECU $u_k$ is ready for task execution. $AFT(n_h^\alpha)$ is the actual finish time of the replica $n_h^\alpha$ and is calculated by

\[
AFT(n_h^\alpha) = EFT(n_h^\alpha, u_{pr(n_h^\alpha)}).
\]

c_{h,i}'$ represents the WCRT between $n_h^\alpha$ and $n_i^\beta$. If $n_h^\alpha$ and $n_i^\beta$ are allocated to the same ECU, then $c_{h,i}' = 0$; otherwise, $c_{h,i}' = c_{h,i}$. $n_i^\beta$ is allocated to the ECU with the minimum EFT by using the insertion-based scheduling policy that $n_i^\beta$ can be inserted into the slack with the minimum EFT.

The actual response time of the function is the AFT of the replica of the exit task $n_{\text{exit}}$; this replica has the maximum AFT among all the replicas of $n_{\text{exit}}$. That is, we have

\[
RT(G) = \max_{\beta \in [1, \text{num}_\text{exit}]} \left\{ AFT(n_{\text{exit}}^\beta) \right\}.
\]

Finally, $RT(G)$ and $RT_{\text{req}}(G)$ are compared to verify the functional safety requirement.

### 4.4 The VFSR Method

Based on the aforementioned analysis, we present the heuristic method VFSR described in Algorithm 1 to verify the functional safety requirement of the function.

The main idea of VFSR is that the reliability requirement of the function is transferred to each task. VFSR simply iteratively assigns the current task’s replicas to available ECUs with the minimum EFTs until its reliability requirement is satisfied. “available ECUs” means that no other replicas of the current task have been assigned to these ECUs. The main steps are explained as follows:

1. In Line 1, VFSR orders tasks in the descending order of $rank_{\alpha}(n_i, u_k)$ using Equation (7). In Lines 2 to 10, VFSR heuristically assigns the tasks of the function.
2. In Lines 3 to 9, VFSR iteratively assigns the current task’s replicas to available ECUs with the minimum EFTs until its reliability requirement is satisfied. In particular, the reliability requirement of each task is obtained in Line 4.
3. In Lines 10 to 13, VFSR calculates the resource cost $RC(G)$, response time $RT(G)$, and reliability value $R(G)$ of the function.
4. In Lines 14 to 18, VFSR compares the actual response time $RT(G)$ with the response time requirement $RT_{\text{req}}(G)$ to verify the functional safety requirement of the function.

Compared with the DRR method [32], the main advantages of the proposed VFSR method are as follows:
ALGORITHM 1: The VFSR method

Input: $G = (N, W, M, C, U, R_{req}(G), RT_{req}(G))$

Output: $NR(G), RC(G), RT(G), R(G)$ and related values

1: Order tasks in the descending order of $\text{rank}_u(n_i, u_k)$ using Equation (7);

2: for ($y \leftarrow 1; y \leq |N|; y++$) do

3: $n_i \leftarrow n_{seq}(y)$;

4: Calculate $R_{req}(n_i)$ using Equation (10);

5: $R(n_i) \leftarrow 0$; // initial value is 0

6: while ($R(n_i) < R_{req}(n_i)$) do

7: Select available replica $n_i^β$ and ECU $u_{pr}(n_i^β)$ with the minimum EFT;

8: Calculate $R(n_i)$ using Equation (3);

9: end while

10: end for

11: Calculate $RC(G)$ using Equation (6);

12: Calculate $RT(G)$ using Equation (15);

13: Calculate $R(G)$ using Equation (4);

14: if ($RT(G) < RT_{req}(G)$) then

15: return true;

16: else

17: return false;

18: end if

(1) VFSR recalculates the reliability requirement of each task based not only on its previous assignments ($\{n_{seq}(1), n_{seq}(2), \ldots, n_{seq}(y-1)\}$) but also on succeeding preassignments ($\{n_{seq}(y+1), n_{seq}(y+2), \ldots, n_{seq}(|N|)\}$), whereas DRR is merely based on previous assignments.

(2) VFSR iteratively assigns the current task’s replicas to available ECU with the minimum EFT to reduce its response time values until its reliability requirement is satisfied, whereas DRR iteratively assigns the current task’s replicas to available ECUs with the maximum reliability value to reduce the number of replicas. However, a minimum number of replicas does not mean the shortest response time in heterogeneous distributed systems.

(3) VFSR rejects the schedule only when the actual response time of the function exceeds its response time requirement, whereas DRR rejects the schedule as long as any actual response time of a task exceeds its response time requirement.

The time complexity of the VFSR method is analyzed as follows: All tasks should be traversed once, which can be conducted in $O(|N|)$ time. The number of replicas should be lower than or equal to the number of ECUs, which can be completed in $O(|U|)$ time. Calculating the AFT of each replica should be conducted in $O(|N| \times |U|)$ time. Thus, the time complexity of the VFSR method is $O(|N|^2 \times |U|^2)$, which is similar to that of the DRR method. Thus, VFSR implements efficient fault tolerance without increasing the time complexity.

We assume that the failure rates for three ECUs are $\lambda_1 = 0.001, \lambda_2 = 0.002$, and $\lambda_3 = 0.003$. Meanwhile, resource cost rates for three ECUs are $r_1 = 3, r_2 = 1$, and $r_3 = 2$. We likewise assume that the reliability requirement of the motivating function in Figure 2 is $R_{req}(G) = 0.9$. Table 4 shows the task assignment for each task of the motivating function using the VFSR method. Each row shows the assigned ECUs (in boxes) and the actual reliability value of the function. VFSR iteratively assigns the current task’s replicas to available ECUs with minimum EFT. For example, the reliability requirement of $n_1$ is $R_{req}(n_1) = 0.989519$; VFSR selects the ECUs $u_3$ and $u_1$ with the minimum and second minimum EFTs, respectively, to satisfy its reliability requirement. Finally, the response time
Table 4. Task Assignment of the Motivating Function Using the VFSR Method

<table>
<thead>
<tr>
<th>(n_i)</th>
<th>(R_{req}(n_i))</th>
<th>(EFT(n_i,u_1))</th>
<th>(EFT(n_i,u_2))</th>
<th>(EFT(n_i,u_3))</th>
<th>(R(n_i))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n_1)</td>
<td>0.989519</td>
<td>14</td>
<td>16</td>
<td>9</td>
<td>0.999630</td>
</tr>
<tr>
<td>(n_3)</td>
<td>0.979511</td>
<td>32</td>
<td>39</td>
<td>45</td>
<td>0.989060</td>
</tr>
<tr>
<td>(n_4)</td>
<td>0.979966</td>
<td>45</td>
<td>31</td>
<td>40</td>
<td>0.984127</td>
</tr>
<tr>
<td>(n_2)</td>
<td>0.985335</td>
<td>45</td>
<td>51</td>
<td>50</td>
<td>0.987084</td>
</tr>
<tr>
<td>(n_5)</td>
<td>0.987766</td>
<td>57</td>
<td>44</td>
<td>35</td>
<td>0.999241</td>
</tr>
<tr>
<td>(n_6)</td>
<td>0.977758</td>
<td>58</td>
<td>60</td>
<td>44</td>
<td>0.999656</td>
</tr>
<tr>
<td>(n_9)</td>
<td>0.967843</td>
<td>76</td>
<td>73</td>
<td>81</td>
<td>0.976288</td>
</tr>
<tr>
<td>(n_7)</td>
<td>0.980962</td>
<td>65</td>
<td>68</td>
<td>66</td>
<td>0.993024</td>
</tr>
<tr>
<td>(n_8)</td>
<td>0.977500</td>
<td>70</td>
<td>84</td>
<td>87</td>
<td>0.995012</td>
</tr>
<tr>
<td>(n_{10})</td>
<td>0.972103</td>
<td>107</td>
<td>89</td>
<td>102</td>
<td>0.986098</td>
</tr>
</tbody>
</table>

\[ RC(G) = 285, \quad RT(G) = 89, \quad R(G) = 0.912586 \]

Fig. 3. Task mapping of the motivating function using VFSR.

of the function is \(RT(G) = 89\), which is less than the given response time requirement of 100. Equation (4) also yields the actual reliability value of the function as 0.912586, which exceeds the given reliability requirement value of 0.9, and it exceeds the given reliability requirement value of 0.9. Therefore, the functional safety requirement of the motivating function is satisfied. Meanwhile, the resource cost is \(RC(G) = 285\) calculated by Equation (6).

Figure 3 also shows the corresponding task mapping of the motivating function \(G\) according to Table 4, where arrows represent the communications between tasks.

5 RESOURCE-COST-AWARE FAULT TOLERANCE

Once the VFSR method returns true (i.e., the functional safety requirement is satisfied) during the early design phase, we can optimize the resource cost based on the results of the VFSR method during the late design phase. Considering that some reliability overspending occurs between the actual reliability value and the reliability requirement, and a large time slack is present between the actual response time and the response time of the function after using VFSR, we can reduce the resource cost by eliminating or reducing the reliability overspending and time slack. As shown in Figure 3 using the VFSR method, the reliability overspending is \(0.912585 - 0.9 = 0.012585\), and the time surplus is \(100 - 89 = 11\) for the motivating function.
5.1 Satisfying Reliability Requirement

Similar to the VFSR method, list scheduling is used to solve the problem stated in Section 3.5. In contrast to VFSR, where the tasks are arranged in descending order of $\text{rank}_u$ values (i.e., from entry to exit tasks by downward optimization), this section sorts the tasks in the ascending order of $\text{rank}_u$ values (i.e., from exit to entry tasks by upward optimization). The reason is that slacks exist between the RT$(G)$ and the RT$_{\text{req}}$(G) by VFSR, and the exit task with the minimum $\text{rank}_u$ can change its AFT to RT$_{\text{req}}$(G), followed by the remaining tasks. For example, the slacks in Figure 3 are between RT$(G)$ = 89 and RT$_{\text{req}}$(G) = 100. $n_{10}$ will be optimized first, followed by $n_8$, $n_7$, $n_6$, $n_5$, $n_2$, $n_4$, $n_3$, and $n_1$. In the following discussion, we first explain how the reliability requirement of each task can be calculated.

Considering that the initial actual reliability of each task has been obtained by VFSR, when reassigning the current task $n_{\text{seq}}(y)$, we let the VFSR-generated assignments of predecessor tasks be fixed. The reliability requirement of the current reassigned task $n_{\text{seq}}(y)$ is calculated as follows:

$$R_{\text{req}}(G) = \prod_{x=1}^{y-1} R_{\text{vfsr}}(n_{\text{seq}}(x)) \times R(n_{\text{seq}}(y)) \times \prod_{z=y+1}^{N} R_{\text{rcfo}}(n_{\text{seq}}(z)), \quad (16)$$

where $R_{\text{vfsr}}(n_{\text{seq}}(x))$ and $R_{\text{rcfo}}(n_{\text{seq}}(z))$ represent the actual reliability values generated by the VFSR and RCFO methods, respectively. In other words, for reliability assurance, the reliability values of other tasks are unchanged when calculating the reliability requirement of the current task, such that the reliability requirement of the current task is

$$R(n_{\text{seq}}(y)) = \frac{R_{\text{req}}(G)}{\prod_{x=1}^{y-1} R_{\text{vfsr}}(n_{\text{seq}}(x)) \times \prod_{z=y+1}^{N} R_{\text{rcfo}}(n_{\text{seq}}(z))}. \quad (17)$$

5.2 Satisfying Response Time Requirement

After the reliability requirements of tasks are obtained, we then determine their response time requirements. We assume that $n_{10}$ has been reassigned in $u_2$ (denoted with green color) using RCFO shown in Figure 4, where we have AST$(n_{10}) = 93$ and AFT$(n_{10}) = 100$. We then consider the second task to be reassigned $n_8$. In the following discussion, $n_8$ is used as the example to explain the process.

As we aim to reassign all the tasks from the exit to the entry tasks in the ascending order of $\text{rank}_u$, we first removed the current task $n_8$ from Figure 4, shown in Figure 5. We then reassign $n_8$’s replicas into Figure 5 with less resource cost without violating its reliability and response time requirements. $n_8$’s reliability requirement can be calculated using Equation (17), and its response time requirement is calculated as follows:
Fig. 5. Remove $n_8$ from Figure 4 of the motivating function.

(1) Considering that $n_8$’s successor task $n_{10}$ has been reassigned by RCFO and cannot be changed, the latest finish time (LFT) of the current task $n_i$ is restricted by its successor tasks because of the precedence constraints among them. LFT is calculated as follows:

$$LFT(n_{\text{exit}}, u_k) = RT_{\text{req}}(G)$$

$$LFT(n_i, u_k) = \min_{n_j \in \text{succ}(n_i), y \in [1,num_j]} \left\{ \text{AST}(n_j^y) - c_i' \right\}.$$

(2) Considering that $n_8$’s predecessor tasks $n_2$, $n_4$, and $n_6$ have been assigned by VFSR and cannot be changed, the earliest start time (EST) of the current task $n_i$ is also restricted by its predecessor tasks owing to the precedence constraints among them. EST is calculated as follows:

$$EST(n_{\text{entry}}, u_k) = 0$$

$$EST(n_i, u_k) = \max_{n_h \in \text{pred}(n_i), a \in [1,num_h]} \left\{ \text{AFT}(n_j^a) + c_i' \right\}.$$

For example, the ESTs and LFTs of $n_8$ on all the ECUs in Figure 5 can be obtained as

$$\begin{align*}
EST(n_8, u_1) &= 59 \\
EST(n_8, u_2) &= 73 \\
EST(n_8, u_3) &= 73
\end{align*}$$

$$\begin{align*}
LFT(n_8, u_1) &= 82 \\
LFT(n_8, u_2) &= 93 \\
LFT(n_8, u_3) &= 82.
\end{align*}$$

(3) Even when the EST and LFT have been obtained on each ECU for the current task, task reassignment must be further constrained because each ECU is not always available because other tasks have already taken parts of the ECUs and only some slacks remain in the ECUs, as shown in Figure 5. Therefore, task reassignment is actually task insertion. The slack set on ECU $u_k$ for $n_i$ 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). The $q$th slack $S_{i,k,q}$ is defined as follows:

$$S_{i,k,q} = [t_s(S_{i,k,q}), t_e(S_{i,k,q})],$$

where $t_s(S_{i,k,q})$ and $t_e(S_{i,k,q})$ represent corresponding ST and ET, respectively. For example, when assigning the task $n_8$ in Figure 5, the slacks on $u_1$, $u_2$, and $u_3$ for $n_8$ are

$$\begin{align*}
S_{8,1} &= \{[14, 21], [65, 100]\} \\
S_{8,2} &= \{[0, 23], [44, 61], [73, 93]\} \\
S_{8,3} &= \{[9, 25], [44, 100]\}.
\end{align*}$$
To avoid violating the precedence constraints among tasks, the current task \( n_i \) should be assigned to the slacks that satisfy the new EST and LFT constraints as follows:

\[
EST(n_i, u_k) = \max \{EST(n_i, u_k), t_s(S_i, k, t)\},
\]

and

\[
LFT(n_i, u_k) = \min \{LFT(n_i, u_k), t_e(S_i, k, t)\}.
\]

For example, the new EST and LFT values of \( n_8 \) on all the ECUs shown in Figure 5 are as follows:

\[
\begin{align*}
EST(n_8, u_1) &= 59 \\
EST(n_8, u_2) &= 73 \\
EST(n_8, u_3) &= 73
\end{align*}
\]

\[
\begin{align*}
LFT(n_8, u_1) &= 82 \\
LFT(n_8, u_2) &= 93 \\
LFT(n_8, u_3) &= 82
\end{align*}
\]

Considering that the ESTs and LFTs for \( n_i \) have been obtained, we can decide which ECU \( n_i \) can be inserted by judging whether Equation (21) is found:

\[
LFT(n_i, u_k) - AST(n_i, u_k) \geq w_{i,k}.
\]

For example, \( n_8 \) can be inserted into \( u_1 \) and \( u_2 \) because

\[
\begin{align*}
LFT(n_8, u_1) - AST(n_8, u_1) &= 82 - 59 = 23 \geq w_{8,1} = 5 \\
LFT(n_8, u_2) - AST(n_8, u_2) &= 93 - 73 = 20 \geq w_{8,2} = 11 \\
LFT(n_8, u_3) - AST(n_8, u_3) &= 82 - 73 = 9 < w_{8,3} = 14
\end{align*}
\]

### 5.3 Optimizing Resource Cost

After the reliability and response time requirements of the current task have been obtained in the previous section, the current task can be inserted into the ECUs. Given that our objective is to reduce the resource cost, the strategy is as follows: we simply iteratively assign the current task \( n_i \)'s replica to available and insertable ECU with the minimum resource costs until \( n_i \)'s reliability requirement is satisfied. That is, the assigned ECU \( u_{\text{min}} \) and corresponding resource cost \( RC(n_i, u_{\text{min}}) \) for \( n_i \) are determined by the following:

\[
RC(n_i, u_{\text{min}}) = \min_{u_k \in U, u_k \text{ is available, } u_k \text{ is insertable}} \{RC(n_i, u_k)\}.
\]

“\( u_k \) is available” means that no other replicas of \( n_i \) have been assigned to \( u_k \) as explained earlier. “\( u_k \) is insertable” means that \( n_i \) can be inserted into the \( u_k \) without exceeding the slack limitation and violating the precedence constraint with its predecessors and successors. We continue to use \( n_8 \) to explain the resource cost optimization process.

(1) \( n_8 \) is first inserted into \( u_2 \) of Figure 5 because it has the minimum resource cost of \( 1 \times 11 = 11 \), as shown in Figure 6.
Fig. 7. $n_8$ is then inserted into $u_1$ of Figure 6 of the motivating function.

Fig. 8. Remove $n_1^1$ from Figure 7 of the motivating function.

(2) The reliability of the replica $n_8^1$ assigned to $u_2$ is 0.978240 (calculated by Equation (1)), which cannot satisfy $n_8$’s reliability requirement of 0.981290 (calculated by Equation (17)). Therefore, $n_8^2$ must be assigned further to the remaining available and insertable ECU $u_1$, as shown in Figure 7. In this case, $n_8$’s actual reliability value is 0.999998 (calculated by Equation (3)), which has exceeded its reliability requirement of 0.981290.

(3) Although we can reduce the resource cost by iteratively assigning $n_8$’s replicas to available and insertable ECUs with the minimum resource costs until its reliability requirement is satisfied, we find that such a process may still cause additional redundancy. For example, we can remove $n_8^1$ from Figure 7 and its actual reliability value is reduced to 0.995012 (calculated by Equation (3)), which still satisfies its reliability requirement of 0.981290. The new assignment is shown in Figure 8, which filters out the replica $n_8^1$ for $n_8$ with less redundancy. The above filter process is implemented as follows: we reserve the selected replicas and ECUs, clear the previous assignments of the task, and then iteratively select replicas and ECUs with the maximum reliability values for each task from the reserved replicas and ECUs until its reliability requirement is satisfied. According to the above filter process, the $n_8^2$ and $u_1$ are selected and $n_8^1$ and $u_2$ are no longer needed, as shown in Figure 8. That is, the assigned ECU $u_{\text{max}}$ and corresponding reliability value $R(n_i, u_{\text{max}})$ for $n_i$ are determined using the following formula:

$$R(n_i, u_{\text{max}}) = \max_{u_k \in U, u_k \text{ is reserved}, u_k \text{ is reserved}} \{R(n_i, u_k)\}. \quad (23)$$

5.4 The RCFO Method

In the analysis of Sections 5.1 to 5.3, the RCFO method is presented to reduce the resource cost of an end-to-end distributed automotive function while satisfying its functional safety requirement in this subsection, as shown in Algorithm 2.
ALGORITHM 2: The RCFO method

Input: $G = (N, W, M, C, U, R_{req}(G), RT_{req}(G), \text{VSFR-generated assignments})$

Output: $RC(G)$ and related values

1: Order tasks in the ascending order of $rank_u(n_i, u_k)$ using Equation (7);
2: for ($y \leftarrow |N|; y \geq 1; y \leftarrow$) do
3:   $n_i \leftarrow n_{seq}(y)$;
4:   Calculate $R_{req}(n_i)$ using Equation (17);
5:   Clear the VSFR-generated assignments of $n_i$;
6:   $R(n_i) \leftarrow 0$; // initial value is 0
7:   while ($R(n_i) < R_{req}(n_i)$) do
8:     Assign $n_i$’s replica to the available and insertable ECU with the minimum resource cost by Equation (22);
9:     Calculate $R(n_i)$ using Equation (3);
10: end while
11: Reverse and clear and the previous assignments of $n_i$ in Lines 7–10;
12: $R(n_i) \leftarrow 0$; // reset the reliability value of $n_i$ to 0;
13: while ($R(n_i) < R_{req}(n_i)$) do
14:     Select $n_i$’s replica and ECU with the maximum reliability value from the reversed assignments by Equation (23);
15:     Calculate $R(n_i)$ using Equation (3);
16: end while
17: end for
18: Calculate $RC(G)$ using Equation (6);
19: Calculate $RT(G)$ using Equation (15);
20: Calculate $R(G)$ using Equation (4).

The main idea of RCFO is described as follows: (1) RCFO first iteratively assigns the current task $n_i$’s replicas to the available and insertable ECUs with the minimum resource costs until $n_i$’s reliability requirement is satisfied; (2) RCFO reserves the selected replicas and ECUs, clears the previous assignments of the task, and then iteratively selects replicas and ECUs with the maximum reliability values for $n_i$ from the reserved replicas and ECUs until its reliability requirement is satisfied. The main steps are explained as follows:

1. In Line 1, contrary to VSFR, RCFO orders tasks in the ascending order of $rank_u(n_i)$ using Equation (7). In Lines 2 to 17, RCFO heuristically assigns the tasks of the function.
2. In Lines 5 to 10, RCFO clears the VSFR-generated assignments of $n_i$ and iteratively assigns the current task $n_i$’s replicas to the available and insertable ECUs with the minimum resource costs until $n_i$’s reliability requirement is satisfied.
3. In Lines 11 to 16, RCFO reserves the assigned replicas and ECUs, clears the previous assignments of $n_i$, and then iteratively selects replicas and ECUs with the maximum reliability values for $n_i$ from the reserved replicas and ECUs until its reliability requirement is satisfied.
4. In Lines 18 to 20, RCFO calculates the resource cost $RC(G)$, response time $RT(G)$, and reliability value $R(G)$ of the function.

Similar to $O(|N|^2 \times |U|)$, RCFO has the same time complexity analysis as VSFR. Its time complexity is also $O(|N|^2 \times |U|)$.

The same parameter values ($\lambda_1 = 0.001, \lambda_2 = 0.002, \lambda_3 = 0.003, r_1 = 3, r_2 = 1, r_3 = 2$, and $R_{req}(G) = 0.9$) as the aforementioned example are used. Figure 9 shows the task mapping of the motivating function $G$ using RCFO, where $RT(G) = 94$, $R(G) = 0.911706$, and $RC(G) = 236$. The
Fig. 9. Task assignments of the motivating function using RCFO.

Table 5. Parameter Values of ACPS

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Scopes</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCETs of ECU tasks</td>
<td>100μs–400μs</td>
</tr>
<tr>
<td>WCRTs of CAN messages</td>
<td>100μs–400μs</td>
</tr>
<tr>
<td>Failure rates of ECUs</td>
<td>10⁻⁶/μs–10⁻⁵/μs</td>
</tr>
<tr>
<td>Resource cost ratios of ECUs</td>
<td>0.5Kb/μs–1.5Kb/μs</td>
</tr>
</tbody>
</table>

response time of the function is calculated by

\[
RT(G) = RT_{req}(G) - \min_{\beta \in [1, num_{entry}]} \left\{ AFT\left(n_{entry}^\beta, u_{pr}(n_{entry}^\beta)\right)\right\},
\]

because the minimum AST of the entry task is larger than or equal to 0. Compared with VFSR, RCFO can reduce the resource cost of 285-236=49 while satisfying the functional safety requirement.

6 EXPERIMENTS

6.1 Experimental Metrics

Considering that this study aims to optimize the resource cost of a distributed automotive function while satisfying its functional safety requirement, performance metrics selected for comparison should be the actual reliability value (calculated by Equation (4)), actual response time value (calculated by Equation (15)), and final resource cost (calculated by Equation (6)) of the function.

Algorithms compared with the proposed VFSR and RCFO algorithms are the DRR [32], MRCRG [23], and HRRM [24] algorithms. DRR is the state-of-the-art method to minimize the resource cost for an end-to-end distributed function while satisfying its functional safety requirement on heterogeneous distributed systems using fault tolerance. MRCRG is the state-of-the-art method aiming to minimize the resource cost for an end-to-end distributed function while satisfying its reliability requirement on heterogeneous distributed systems without using fault tolerance. HRRM is the state-of-the-art method aiming to minimize the redundancy for an end-to-end distributed function while satisfying its reliability requirement on heterogeneous distributed systems using fault tolerance. Therefore, DRR, MRCRG, and HRRM should be used to compare with the proposed VFSR and RCFO methods 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 ACPSs as experimental data. The parameter
values of the function used in this study are summarized in Table 5 [7, 26]. The aforementioned values are generated with uniform distribution.

The distributed functions will be tested on simulated ACPSs based on the above 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 guides 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 2.6GHz Intel CPU and 4GB 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 performed 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.

ISO 26262 provides the duration/probability of exposure in Table B.2, Annex B of Part 3 of [12], as shown in Table 6. Exposure refers to the relative expected frequency of the operational conditions, in which hazardous events may occur and cause hazards and injuries. It has five levels of E0, E1, E2, E3, and E4. Exposure is related to random hardware failures of hazardous events [12]. That is, reliability is merely the inverse expression of exposure.

Note that ISO 26262 does not define the concept of reliability requirement, but we can deduce the corresponding reliability requirements for given exposure levels. For example, the probability of exposure E2 is less than 1% of average operating time. That is, the lowest probability of an 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 requirement in this case. The reliability requirements for other exposures can also be obtained according to the above rule. Finally, the reliability requirements for exposures are shown in Table 6.

### 6.2 Real-Life End-to-End Distributed Automotive Function

We first use the real-life end-to-end distributed automotive function of an automotive case study adopted from [7], as shown in Figure 10. 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 two tasks ($n_{18}$–$n_{19}$), suspension controller with five tasks ($n_{20}$–$n_{24}$), and body work with seven tasks ($n_{25}$–$n_{31}$).

Given that this study considers the functional safety requirement consisting of reliability and response time requirements, we first fix the response time requirement for varying reliability requirements, and then fix the reliability requirement for varying response time requirements.

**Experiment 1.** This experiment is conducted to compare the actual reliability values, response time values, and final resource costs of the real-life end-to-end distributed automotive function for varying reliability requirements. The response time requirement of the function is 1,000μs. The reliability requirement is changed from 0.9 to 0.99 with a 0.01 increment, because it falls in

<table>
<thead>
<tr>
<th>Exposure Level</th>
<th>Probability of Exposure</th>
<th>Reliability Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1 Very low probability</td>
<td>Not specified</td>
<td>At least exceeds 0.99</td>
</tr>
<tr>
<td>E2 Low probability</td>
<td>&lt;1%</td>
<td>0.99</td>
</tr>
<tr>
<td>E3 Medium probability</td>
<td>[1%, 10%]</td>
<td>&gt;0.9</td>
</tr>
<tr>
<td>E4 High probability</td>
<td>&gt;10%</td>
<td>&lt;=0.9</td>
</tr>
</tbody>
</table>
Table 7. Reliability Values of Real-Life Automotive Function for Varying Reliability Requirements

<table>
<thead>
<tr>
<th>$R_{req}(G) = x$</th>
<th>$x=0.9$</th>
<th>$x=0.91$</th>
<th>$x=0.92$</th>
<th>$x=0.93$</th>
<th>$x=0.94$</th>
<th>$x=0.95$</th>
<th>$x=0.96$</th>
<th>$x=0.97$</th>
<th>$x=0.98$</th>
<th>$x=0.99$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRR</td>
<td>0.998742</td>
<td>0.998742</td>
<td>0.998742</td>
<td>0.998742</td>
<td>0.998742</td>
<td>0.998742</td>
<td>0.998742</td>
<td>0.998742</td>
<td>0.998742</td>
<td>0.998742</td>
</tr>
<tr>
<td>MRCRG</td>
<td>0.977661</td>
<td>0.977661</td>
<td>0.977661</td>
<td>0.977661</td>
<td>0.977661</td>
<td>0.977661</td>
<td>0.977661</td>
<td>0.977661</td>
<td>0.980006</td>
<td>0.990020</td>
</tr>
<tr>
<td>HRRM</td>
<td>0.995060</td>
<td>0.995060</td>
<td>0.995060</td>
<td>0.995060</td>
<td>0.995060</td>
<td>0.995060</td>
<td>0.995060</td>
<td>0.995060</td>
<td>0.995060</td>
<td>0.995060</td>
</tr>
<tr>
<td>VFSR</td>
<td>0.976756</td>
<td>0.976756</td>
<td>0.976756</td>
<td>0.976756</td>
<td>0.976756</td>
<td>0.976756</td>
<td>0.976756</td>
<td>0.976756</td>
<td>0.980668</td>
<td>0.990329</td>
</tr>
<tr>
<td>RCFO</td>
<td>0.977487</td>
<td>0.977487</td>
<td>0.977487</td>
<td>0.977487</td>
<td>0.977487</td>
<td>0.977487</td>
<td>0.977487</td>
<td>0.977487</td>
<td>0.981681</td>
<td>0.990195</td>
</tr>
</tbody>
</table>

(1) Table 7 shows that actual reliability values using all the methods are always larger than the corresponding reliability requirements. DRR always rejects the schedule in all the cases because the response time requirements of certain tasks cannot be satisfied. That is, DRR is not suitable for safe-critical embedded functions. To observe the final results, we let DRR continue to schedule in the case of schedule rejection. We can see that DRR and HRRM always obtain individual equal reliability values; the reason is that both DRR and HRRM implement resource cost reduction by iteratively assigning a task’s replicas to available ECUs with the maximum reliability values until its reliability requirement is satisfied. By contrast, a small function can easily obtain a high reliability value by the above assignments to satisfy the reliability requirement. Similarly, MRCRG, VFSR, and RCFO can obtain high reliability values for a small function even if they do not iteratively assign replicas to available ECUs with the maximum reliability.

(2) Table 8 shows that DRR, MRCRG, and HRRM cannot satisfy the response time requirement of 1,000μs in all the cases, whereas VFSR and RCFO can always satisfy the response...
Table 8. Response Time Values (Unit: \(\mu s\)) of Real-Life Automotive Function for Varying Reliability Requirements

<table>
<thead>
<tr>
<th>(R_{req}(G) = x)</th>
<th>(x=0.9)</th>
<th>(x=0.91)</th>
<th>(x=0.92)</th>
<th>(x=0.93)</th>
<th>(x=0.94)</th>
<th>(x=0.95)</th>
<th>(x=0.96)</th>
<th>(x=0.97)</th>
<th>(x=0.98)</th>
<th>(x=0.99)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRR</td>
<td>2,370</td>
<td>2,370</td>
<td>2,370</td>
<td>2,370</td>
<td>2,370</td>
<td>2,370</td>
<td>2,370</td>
<td>2,370</td>
<td>2,370</td>
<td>2,370</td>
</tr>
<tr>
<td>MRCRG</td>
<td>1,768</td>
<td>1,768</td>
<td>1,768</td>
<td>1,768</td>
<td>1,768</td>
<td>1,768</td>
<td>1,768</td>
<td>1,768</td>
<td>1,748</td>
<td>1,600</td>
</tr>
<tr>
<td>HRRM</td>
<td>1,134</td>
<td>1,134</td>
<td>1,134</td>
<td>1,134</td>
<td>1,134</td>
<td>1,134</td>
<td>1,134</td>
<td>1,134</td>
<td>1,134</td>
<td>1,134</td>
</tr>
<tr>
<td>VFSR</td>
<td>837</td>
<td>837</td>
<td>837</td>
<td>837</td>
<td>837</td>
<td>837</td>
<td>837</td>
<td>956</td>
<td>956</td>
<td>956</td>
</tr>
<tr>
<td>RCFO</td>
<td>978</td>
<td>978</td>
<td>978</td>
<td>978</td>
<td>978</td>
<td>978</td>
<td>978</td>
<td>969</td>
<td>988</td>
<td></td>
</tr>
</tbody>
</table>

Table 9. Resource Cost (Unit: Kb) of Real-Life Automotive Function for Varying Reliability Requirements

<table>
<thead>
<tr>
<th>(R_{req}(G) = x)</th>
<th>(x=0.9)</th>
<th>(x=0.91)</th>
<th>(x=0.92)</th>
<th>(x=0.93)</th>
<th>(x=0.94)</th>
<th>(x=0.95)</th>
<th>(x=0.96)</th>
<th>(x=0.97)</th>
<th>(x=0.98)</th>
<th>(x=0.99)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRR</td>
<td>7,711.5</td>
<td>7,711.5</td>
<td>7,711.5</td>
<td>7,711.5</td>
<td>7,711.5</td>
<td>7,711.5</td>
<td>7,711.5</td>
<td>7,711.5</td>
<td>7,711.5</td>
<td>7,711.5</td>
</tr>
<tr>
<td>MRCRG</td>
<td>2,175.1</td>
<td>2,175.1</td>
<td>2,175.1</td>
<td>2,175.1</td>
<td>2,175.1</td>
<td>2,175.1</td>
<td>2,175.1</td>
<td>2,258</td>
<td>2,773.0</td>
<td></td>
</tr>
<tr>
<td>HRRM</td>
<td>3,517.1</td>
<td>3,517.1</td>
<td>3,517.1</td>
<td>3,517.1</td>
<td>3,517.1</td>
<td>3,517.1</td>
<td>3,517.1</td>
<td>3,517.1</td>
<td>3,517.1</td>
<td>3,517.1</td>
</tr>
<tr>
<td>VFSR</td>
<td>4,202.6</td>
<td>4,202.6</td>
<td>4,202.6</td>
<td>4,202.6</td>
<td>4,202.6</td>
<td>4,202.6</td>
<td>4,202.6</td>
<td>4,399.1</td>
<td>5,010.1</td>
<td></td>
</tr>
<tr>
<td>RCFO</td>
<td>2,440.4</td>
<td>2,440.4</td>
<td>2,440.4</td>
<td>2,440.4</td>
<td>2,440.4</td>
<td>2,440.4</td>
<td>2,440.4</td>
<td>2,431.2</td>
<td>3,263.3</td>
<td></td>
</tr>
</tbody>
</table>

The combination of Tables 7 and 8 reveals that only VFSR and RCFO can simultaneously satisfy the functional safety requirements of the function. However, VFSR is only used to verify the functional safety requirement, and it is not a resource-cost-aware method. Based on VFSR, RCFO considers the resource cost optimization while satisfying the functional safety requirement. In addition, RCFO can reduce 22% to 39% of the resource costs compared with VFSR, as shown in Table 9.

Experiment 2. This experiment is conducted to compare the actual reliability values, response time values, and final resource costs of the real-life automotive function for varying response time requirements. The response time requirement is changed from 1,000\(\mu s\) to 1,900\(\mu s\) with a 100\(\mu s\) increment. The reliability requirement is fixed at 0.99, which is the reliability requirement of exposure E2 in ISO 26262. The results are shown in Tables 10 through 12.
Table 10. Reliability Values of Real-Life Automotive Function for Varying Response Time Requirements (Unit: μs)

<table>
<thead>
<tr>
<th>R&lt;sub&gt;req&lt;/sub&gt;(G) (μs)</th>
<th>x=1,000</th>
<th>x=1,100</th>
<th>x=1,200</th>
<th>x=1,300</th>
<th>x=1,400</th>
<th>x=1,500</th>
<th>x=1,600</th>
<th>x=1,700</th>
<th>x=1,800</th>
<th>x=1,900</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRR</td>
<td>0.998742</td>
<td>0.998742</td>
<td>0.998742</td>
<td>0.998742</td>
<td>0.998742</td>
<td>0.998742</td>
<td>0.998742</td>
<td>0.998742</td>
<td>0.998742</td>
<td></td>
</tr>
<tr>
<td>MRCRG</td>
<td>0.999020</td>
<td>0.999020</td>
<td>0.999020</td>
<td>0.999020</td>
<td>0.999020</td>
<td>0.999020</td>
<td>0.999020</td>
<td>0.999020</td>
<td>0.999020</td>
<td></td>
</tr>
<tr>
<td>HRRM</td>
<td>0.995060</td>
<td>0.995060</td>
<td>0.995060</td>
<td>0.995060</td>
<td>0.995060</td>
<td>0.995060</td>
<td>0.995060</td>
<td>0.995060</td>
<td>0.995060</td>
<td></td>
</tr>
<tr>
<td>VFSR</td>
<td>0.990329</td>
<td>0.990329</td>
<td>0.990329</td>
<td>0.990329</td>
<td>0.990329</td>
<td>0.990329</td>
<td>0.990329</td>
<td>0.990329</td>
<td>0.990329</td>
<td></td>
</tr>
<tr>
<td>RCFO</td>
<td>0.990195</td>
<td>0.990179</td>
<td>0.990179</td>
<td>0.990179</td>
<td>0.990179</td>
<td>0.990179</td>
<td>0.990179</td>
<td>0.990179</td>
<td>0.990179</td>
<td></td>
</tr>
</tbody>
</table>

Table 11. Response Time Values (Unit: μs) of Real-life Automotive Function for Varying Response Time Requirements (Unit: μs)

<table>
<thead>
<tr>
<th>R&lt;sub&gt;req&lt;/sub&gt;(G) (μs)</th>
<th>x=1,000</th>
<th>x=1,100</th>
<th>x=1,200</th>
<th>x=1,300</th>
<th>x=1,400</th>
<th>x=1,500</th>
<th>x=1,600</th>
<th>x=1,700</th>
<th>x=1,800</th>
<th>x=1,900</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRR</td>
<td>2,370</td>
<td>2,370</td>
<td>2,370</td>
<td>2,370</td>
<td>2,370</td>
<td>2,370</td>
<td>2,370</td>
<td>2,370</td>
<td>2,370</td>
<td></td>
</tr>
<tr>
<td>MRCRG</td>
<td>1,600</td>
<td>1,600</td>
<td>1,600</td>
<td>1,600</td>
<td>1,600</td>
<td>1,600</td>
<td>1,600</td>
<td>1,600</td>
<td>1,600</td>
<td></td>
</tr>
<tr>
<td>HRRM</td>
<td>1,134</td>
<td>1,134</td>
<td>1,134</td>
<td>1,134</td>
<td>1,134</td>
<td>1,134</td>
<td>1,134</td>
<td>1,134</td>
<td>1,134</td>
<td></td>
</tr>
<tr>
<td>VFSR</td>
<td>837</td>
<td>837</td>
<td>837</td>
<td>837</td>
<td>837</td>
<td>837</td>
<td>837</td>
<td>837</td>
<td>837</td>
<td></td>
</tr>
<tr>
<td>RCFO</td>
<td>988</td>
<td>989</td>
<td>1,159</td>
<td>1,278</td>
<td>1,384</td>
<td>1,499</td>
<td>1,566</td>
<td>1,630</td>
<td>1,768</td>
<td></td>
</tr>
</tbody>
</table>

Table 12. Resource Cost (Unit: Kb) of Real-Life Automotive Function for Varying Response Time Requirements (Unit: μs)

<table>
<thead>
<tr>
<th>R&lt;sub&gt;req&lt;/sub&gt;(G) (μs)</th>
<th>x=1,000</th>
<th>x=1,100</th>
<th>x=1,200</th>
<th>x=1,300</th>
<th>x=1,400</th>
<th>x=1,500</th>
<th>x=1,600</th>
<th>x=1,700</th>
<th>x=1,800</th>
<th>x=1,900</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRR</td>
<td>7,711.5</td>
<td>7,711.5</td>
<td>7,711.5</td>
<td>7,711.5</td>
<td>7,711.5</td>
<td>7,711.5</td>
<td>7,711.5</td>
<td>7,711.5</td>
<td>7,711.5</td>
<td></td>
</tr>
<tr>
<td>MRCRG</td>
<td>2,773.0</td>
<td>2,773.0</td>
<td>2,773.0</td>
<td>2,773.0</td>
<td>2,773.0</td>
<td>2,773.0</td>
<td>2,773.0</td>
<td>2,773.0</td>
<td>2,773.0</td>
<td></td>
</tr>
<tr>
<td>HRRM</td>
<td>3,517.1</td>
<td>3,517.1</td>
<td>3,517.1</td>
<td>3,517.1</td>
<td>3,517.1</td>
<td>3,517.1</td>
<td>3,517.1</td>
<td>3,517.1</td>
<td>3,517.1</td>
<td></td>
</tr>
<tr>
<td>VFSR</td>
<td>5,010.1</td>
<td>5,010.1</td>
<td>5,010.1</td>
<td>5,010.1</td>
<td>5,010.1</td>
<td>5,010.1</td>
<td>5,010.1</td>
<td>5,010.1</td>
<td>5,010.1</td>
<td></td>
</tr>
<tr>
<td>RCFO</td>
<td>3,263.3</td>
<td>3,007.6</td>
<td>3,212.89</td>
<td>3,057.8</td>
<td>3,057.6</td>
<td>3,105.2</td>
<td>3,081.7</td>
<td>3,121.2</td>
<td>2,988.49</td>
<td>3,067.5</td>
</tr>
</tbody>
</table>

(1) Similar to Table 7, all the methods can satisfy the reliability requirement in all the cases in Table 10. Note that DRR still constantly rejects the schedule in all the cases, but we let it continue to schedule in the case of schedule rejection.

(2) Similar to Table 8, VFSR and RCFO still can always satisfy the response time requirements in all the cases in Table 11. With the increment of response time requirements, RCFO-generated response time values are always less than and close to the response time requirements. HRRM and MRCRG always generate fixed response time values of 1,600μs and 1,134μs, respectively; the reason for the above fixed results is that HRRM and MRCRG only focus on the reliability requirement and this requirement is fixed at 0.99 in this experiment. Therefore, when the response time requirements exceed certain values, HRRM and MRCRG can also satisfy the response time requirements. DRR still cannot satisfy the response time requirements in all the cases because, in DRR, the reliability requirements of tasks are unfair, as explained in Section 4.2.

(3) Similar to Table 9, MRCRG still generates the minimum resource costs followed by RCFO, HRRM, VFSR, and DRR in Table 12. Although MRCRG can generate less resource cost than RCFO by using non-fault tolerance, it cannot always satisfy the response time requirements shown in Table 11, such that the functional safety requirements cannot always be
satisfied. In a word, RCFO is the best choice among the five methods in reducing resource costs while satisfying the functional safety requirements.

6.3 Synthetic End-to-End Distributed Function

Given the increasing complexity of ACPSs, future automotive functions are likely to reach 100 tasks. To further validate the effectiveness, we use additional simulated functions with the same actual parameter values of the real-life function to observe the results. Heterogeneity can also affect the response time and reliability values of the function because the automotive E/E architecture consists of heterogeneous ECUs. Heterogeneity can be easily implemented for randomly generated end-to-end distributed functions by providing a selective heterogeneity factor. Randomly generated end-to-end distributed functions can be obtained by the task graph generator [10]. The ECU number, failure rates of ECUs, WCETs of the tasks, and WCRTs of the messages are still similar to the real-life functions.

Experiment 3. This experiment shows the actual reliability values, response time values, and final resource costs of a synthetic function for various reliability requirements. The function parameters are set as follows: 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 to 1 scope in the task graph generator, where 0.1 and 1 are the lowest and highest heterogeneity factors, respectively. Task reliability requirements are changed from 0.9 to 0.99 with 0.01 increments. The response time requirement is fixed at $3,035 \times 1.2 = 4,235 \mu s$, where 3,035 is the response time value obtained by VFSR when $R_{\text{req}}(G) = 0.9$. The reason of such setting is to ensure that the verification is passed by VFSR. The results are shown in Tables 13 through 15.

(1) Table 13 shows that except for MRCRG, other methods can always satisfy the reliability requirements. When the reliability requirement exceeds 0.95, MRCRG no longer satisfies the reliability requirements (denoted with ") because the maximum reliability value without using fault tolerance is 0.958434 for the function with 100 tasks, such that MRCRG is not suitable for large functions, and fault-tolerant methods should be used to satisfy high-reliability requirements.

(2) Similar to the results of the small function in Table 8, VFSR generates the minimum response time values followed by RCFO, HRRM, MRCRG, and DRR for the large function in Table 14. Clearly, only VFSR and RCFO can still satisfy the response time requirement of 4,235$\mu s$, whereas HRRM, MRCRG, and DRR still cannot. As MRCRG cannot complete the schedule when the reliability requirement exceeds 0.95, the actual response time values are denoted with "-".

(3) As we expect, MRCRG still generates the minimum resource cost among all the methods when it can generate correct schedule results owing to it using a non-fault-tolerant method, as shown in Table 15. However, MRCRG cannot always guarantee that the reliability and response time requirements are satisfied; this method is not suitable for a
Table 14. Response Time Values (Unit: μs) of Real-Life Function for Varying Reliability Requirements

<table>
<thead>
<tr>
<th>$R_{req}(G) = x$</th>
<th>$x=0.9$</th>
<th>$x=0.91$</th>
<th>$x=0.92$</th>
<th>$x=0.93$</th>
<th>$x=0.94$</th>
<th>$x=0.95$</th>
<th>$x=0.96$</th>
<th>$x=0.97$</th>
<th>$x=0.98$</th>
<th>$x=0.99$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRR</td>
<td>15.357</td>
<td>15.357</td>
<td>15.357</td>
<td>15.357</td>
<td>15.723</td>
<td>17.018</td>
<td>17.384</td>
<td>18.879</td>
<td>20.896</td>
<td>22.394</td>
</tr>
<tr>
<td>MRCRG</td>
<td>5.455</td>
<td>5.656</td>
<td>6.912</td>
<td>8.053</td>
<td>11.292</td>
<td>14.031</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>HRRM</td>
<td>4.647</td>
<td>4.647</td>
<td>4.647</td>
<td>4.647</td>
<td>4.647</td>
<td>4.931</td>
<td>4.935</td>
<td>5.323</td>
<td>5.199</td>
<td>-</td>
</tr>
<tr>
<td>VFSR</td>
<td>3.035</td>
<td>3.040</td>
<td>3.096</td>
<td>3.057</td>
<td>3.153</td>
<td>3.394</td>
<td>3.427</td>
<td>3.496</td>
<td>3.496</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 15. Resource Cost (Unit: Kb) of Real-Life Function for Varying Reliability Requirements

<table>
<thead>
<tr>
<th>$R_{req}(G) = x$</th>
<th>$x=0.9$</th>
<th>$x=0.91$</th>
<th>$x=0.92$</th>
<th>$x=0.93$</th>
<th>$x=0.94$</th>
<th>$x=0.95$</th>
<th>$x=0.96$</th>
<th>$x=0.97$</th>
<th>$x=0.98$</th>
<th>$x=0.99$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRR</td>
<td>41.870</td>
<td>41.870</td>
<td>41.870</td>
<td>41.870</td>
<td>41.870</td>
<td>41.870</td>
<td>42.221</td>
<td>47.430</td>
<td>54.556</td>
<td>-</td>
</tr>
<tr>
<td>MRCRG</td>
<td>11.352</td>
<td>11.352</td>
<td>11.352</td>
<td>11.821</td>
<td>13.973</td>
<td>17.512</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

large function. VFSR and RCFO are the methods that can guarantee correct schedules for automotive functional safety. RCFO still generates less resource costs than VFSR, and it outperforms VFSR by 11.5% to 15.4%. In summary, RCFO shows satisfactory resource cost reduction compared with other counterparts while satisfying the functional safety requirement.

7 CONCLUSION

We propose a resource-cost-aware fault-tolerant design methodology for end-to-end functional safety computation on ACPSs during the design phase. Our design methodology first verifies whether the functional safety requirement of a given end-to-end distributed automotive function can be satisfied on an ACPS platform by proposing the VFSR method during the early design phase. Based on the results of VFSR, the proposed RCFO method can reduce resource cost by as much as 39% while satisfying the functional safety requirement during the late design phase. Our proposed design methodology is efficient for both small real-life and synthetic automotive functions based on the experiments. Considering that ACPSs dynamically interact with the physical world, and dynamics is the inherent property of ACPSs, responding to changes in physical environments will be studied in our future work.

REFERENCES


Received April 2017; revised October 2017; accepted November 2017