Abstract—Both response time and reliability are important functional safety properties that must be simultaneously satisfied 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 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.

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,
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 optimization problem. In Fig. 1, each point \( x^5 \) represents a solution of a bicriteria minimization problem [10], [11]. The points \( x^5, x^3, x^2, x^1, \) and \( x^0 \) are Pareto optima; the points \( x^1 \) and \( x^2 \) are weak optima, whereas the points \( x^3, x^4, \) and \( x^5 \) 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.

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 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.

![Fig. 3. Integration architecture of automotive electronic systems [1].](image)

### Table I
**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 between the tasks $n_i$ and $n_j$</td>
</tr>
<tr>
<td>$\text{rank}_i(n_i)$</td>
<td>Upward rank value of the task $n_i$</td>
</tr>
<tr>
<td>$\mu_{\text{ECU}(i)}$</td>
<td>Assigned ECU of the task $n_i$</td>
</tr>
<tr>
<td>$</td>
<td>X</td>
</tr>
<tr>
<td>$\lambda_k$</td>
<td>Failure rate of the ECU $u_k$</td>
</tr>
<tr>
<td>$\mu_{\text{assign}(i)}$</td>
<td>$i$th assigned task of the application</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>$R_{\text{max}}(n_i)$</td>
<td>Reliability requirement of the task $n_i$</td>
</tr>
<tr>
<td>$R(G)$</td>
<td>Reliability of the application $G$</td>
</tr>
<tr>
<td>$R_{\text{max}}(G)$</td>
<td>Maximum Reliability of the application $G$</td>
</tr>
<tr>
<td>$R_{\text{req}}(G)$</td>
<td>Reliability requirement of the application $G$</td>
</tr>
<tr>
<td>$RR(G)$</td>
<td>Reliability ratio of the application $G$</td>
</tr>
<tr>
<td>$RR_{\text{req}}(G)$</td>
<td>Ratio-based reliability pre-assignment of the application $G$</td>
</tr>
<tr>
<td>$RR_{\text{assign}}(n_i)$</td>
<td>Ratio-based reliability pre-assignment of the task $n_i$</td>
</tr>
<tr>
<td>$\text{LB}(G)$</td>
<td>Lower bound of the application $G$</td>
</tr>
<tr>
<td>$\text{RT}(G)$</td>
<td>Response time of the application $G$</td>
</tr>
<tr>
<td>$\text{EST}(n_i,u_k)$</td>
<td>Earliest start time of the task $n_i$ on the ECU $u_k$</td>
</tr>
<tr>
<td>$\text{EFT}(n_i,u_k)$</td>
<td>Earliest finish time of the task $n_i$ on the ECU $u_k$</td>
</tr>
<tr>
<td>$\text{AST}(n_i)$</td>
<td>Actual start time of the task $n_i$</td>
</tr>
<tr>
<td>$\text{AFT}(n_i)$</td>
<td>Actual finish time of the task $n_i$</td>
</tr>
<tr>
<td>$\text{RT}_{\text{req}}(n_i)$</td>
<td>Response time requirement of the task $n_i$</td>
</tr>
<tr>
<td>$\text{RT}_{\text{assign}}(G)$</td>
<td>Response time requirement of the application $G$</td>
</tr>
</tbody>
</table>

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 $u_2$. $U = \{u_1, u_2, \ldots, u_G\}$ 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 $i$th task of $G$. $\text{pred}(n_i)$ represents the set of
immediate predecessor tasks of \( n_i \), whereas \( \text{succ}(n_i) \) represents the set of the immediate successor tasks of \( n_i \). The task with no predecessor task is denoted by \( n_{\text{entry}} \), whereas the task with no successor task is denoted by \( n_{\text{exit}} \). 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 \( n_{\text{entry}} \) or multiple \( n_{\text{exit}} \) 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 probability of an event in unit time \( t \) is denoted by \( R(t) = e^{-\lambda t} \), where \( \lambda \) is the constant failure rate per time unit (i.e., failure rate) for an ECU [16]. We let \( \lambda_{uk} \) 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

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

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

\[
R(G) = \prod_{n_i \in N} R(n_i, u_{pr(n_i)})
\]

where \( u_{pr(n_i)} \) 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
traversing all the ECUs, and it is calculated by
\[
R_{\text{max}}(n_i) = \max_{u_k \in U} R(n_i, u_k). \tag{3}
\]
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
\[
R_{\text{max}}(G) = \prod_{n_i \in G} R_{\text{max}}(n_i). \tag{4}
\]

The reliability requirement \(R_{\text{req}}(G)\) must be less than or equal to \(R_{\text{max}}(G)\). Hence, \(R_{\text{req}}(G)\) must have the following constraint:
\[
R_{\text{req}}(G) \leq R_{\text{max}}(G) \tag{5}
\]
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 \(R_{\text{max}}(G) = 0.974335\). We let reliability requirement \(R_{\text{req}}(G) = 0.96\), which can pass the assessment, because
\[
0.96 \leq 0.974335
\]
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 (\(\text{rank}_{\text{u}}(n_i)\)) of a task [see (6)] as the task priority standard. In this case, the tasks are arranged according to the descending order of \(\text{rank}_{\text{u}}(n_i)\), which is obtained by
\[
\text{rank}_{\text{u}}(n_i) = \overline{w_i} + \max_{n_j \in \text{succ}(n_i)} \{c_{i,j} + \text{rank}_{\text{u}}(n_j)\} \tag{6}
\]
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 \(\text{rank}_{\text{u}}(n_i) > \text{rank}_{\text{u}}(n_j)\); 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 \(\{n_1, n_2, n_4, n_5, n_6, n_3, n_7, n_8, n_9, n_{10}\}\).

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:
\[
\begin{align*}
\text{EST}(n_i, u_k) & = 0; \\
\text{EST}(n_i, u_k) & = \max \left\{ \text{EST}(n_i, u_k) + w_{i,k} \right\}
\end{align*}
\]
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. \(\text{AFT}(n_i, u_k)\) is the actual finish time (AFT) of task \(n_i\), \(c_{h,i}'\) 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.

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 \(n_{\text{exit}}\) represents the exit task as mentioned earlier.

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

TABLE III

UPWARD RANK VALUES FOR TASKS OF THE MOTIVATING DISTRIBUTED APPLICATION

<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_9)</th>
<th>(n_{10})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{rank}_{\text{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>
<td>14.7</td>
</tr>
</tbody>
</table>

Fig. 5. Task scheduling for lower bound calculation.
and risk assessment (see Fig. 5). \( \text{RT}_{\text{req}}(G) \) must be larger than or equal to \( \text{LB}(G) \). Hence, \( \text{RT}_{\text{req}}(G) \) must derive the following constraint:

\[
\text{LB}(G) \leq \text{RT}_{\text{req}}(G) \tag{10}
\]

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

\[
80 \leq 90
\]

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:

\[
\text{RT}(G) \leq \text{RT}_{\text{req}}(G) \tag{11}
\]

\[
\text{R}(G) \geq \text{R}_{\text{req}}(G) \tag{12}
\]

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 \( y \)-th 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. 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:

\[
\text{R}_{s(y)}(G) = \prod_{z=1}^{y-1} \text{R}(n_{s(z)}) \\
\times \text{R}(n_{s(y)}) \times \prod_{z=y+1}^{|N|} \text{R}_{\text{max}}(n_{s(z)}) \geq \text{R}_{\text{req}}(G)
\]

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

\[
\text{R}(n_{s(y)}) \geq \frac{\text{R}_{\text{req}}(G)}{\prod_{z=1}^{y-1} \text{R}(n_{s(z)}) \times \prod_{z=y+1}^{|N|} \text{R}_{\text{max}}(n_{s(z)})}. \tag{13}
\]

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

\[
\text{R}_{\text{req}}(n_{s(y)}) = \frac{\text{R}_{\text{req}}(G)}{\prod_{z=1}^{y-1} \text{R}(n_{s(z)}) \times \prod_{z=y+1}^{|N|} \text{R}_{\text{max}}(n_{s(z)})}. \tag{14}
\]

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

\[
\text{R}(n_{s(y)}) \geq \text{R}_{\text{req}}(n_{s(y)}).
\]

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:

\[
\text{RR}(G) = \frac{|N|}{\sqrt{\text{R}_{\text{req}}(G)}}, \tag{15}
\]

We obtain \( \text{RR}(G) \leq 1 \) because \( \text{R}_{\text{req}}(G) \leq \text{R}_{\text{max}}(G) \) 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:

\[
\text{R}_{\text{trp}}(n_{s(y)}) = \text{R}_{\text{max}}(n_{s(y)}) \times \text{RR}(G). \tag{16}
\]

Obviously, we have

\[
\text{R}_{\text{trp}}(n_{s(y)}) \leq \text{R}_{\text{max}}(n_{s(y)}),
\]

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

\[
\text{R}_{\text{req}}(n_{s(y)}) = \frac{\text{R}_{\text{req}}(G)}{\prod_{z=1}^{y-1} \text{R}(n_{s(z)}) \times \prod_{z=y+1}^{|N|} \text{R}_{\text{trp}}(n_{s(z)})}. \tag{17}
\]

The sole difference between (13) and (17) is that \( \text{R}_{\text{max}}(n_{s(y)}) \) in the former is replaced with \( \text{R}_{\text{trp}}(n_{s(y)}) \) 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 \( \text{R}_{\text{trp}}(n_{s(y)}) \) [see (15)] is used, each task \( n_{s(y)} \) in the distributed automotive application \( G \) can always find an available ECU to satisfy the reliability requirement \( \text{R}_{\text{req}}(G) \) 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

\[ R_{\text{pr}}(G) = \prod_{n_i \in N} R_{\text{pr}}(n_i). \]  

(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

\[
R_{\text{pr}}(G) = \prod_{n_i \in N} R_{\text{pr}}(n_i) = \prod_{n_i \in N} (R_{\text{max}}(n_i) \times RR(G))
\]

\[= \prod_{n_i \in N} R_{\text{max}}(n_i) \times \prod_{n_i \in N} RR(G). \]

(19)

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

\[
R_{\text{pr}}(G) = \prod_{n_i \in N} R_{\text{max}}(n_i) \times \prod_{n_i \in N} \sqrt[\text{max}]{\frac{R_{\text{req}}(G)}{R_{\text{max}}(G)}}
\]

\[= R_{\text{max}}(G) \times \frac{R_{\text{req}}(G)}{R_{\text{max}}(G)} = R_{\text{req}}(G). \]  

(20)

Finally, given that \( R_{\text{pr}}(G) \) 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:

\[ R(n_{s(y)}) \geq R_{\text{req}}(n_{s(y)}). \]

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 \( u_{pr(i)} \) for \( n_i \) is determined by

\[ \text{AFT}(n_i, u_{pr(i)}) = \min_{u_k \in U, R(n_i, u_k) \geq R_{\text{req}}(n_i)} \{ \text{AFT}(n_i, u_k) \}. \]  

(21)

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

Algorithm 1: The FFSV1 Algorithm.

Input: \( U = \{ u_1, u_2, \ldots, u_{|U|} \}, G, R_{\text{req}}(G), R_{\text{RT}}(G) \)

Output: \( R(G), RT(G), \) verification result

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

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

\[ \text{AFT}(n_i) = \text{AFT}(n_i, u_{pr(i)}) \]

(22)

\[ \text{AST}(n_i) = \text{AFT}(n_i) - w_{i,pr(i)}. \]

(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 \( \text{task}_{\text{list}} \) by descending order of rank 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 \( R(n_i, u_k) \geq R_{\text{req}}(n_i) \).
4) Verifying functional safety. In lines 16 and 17, FFSV1 calculates the final reliability \( R(G) \) and the response time.
Algorithm is $O(n)$ time. Therefore, 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.

<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>0.995329</td>
<td>16</td>
<td>-</td>
<td>0.996805</td>
<td></td>
</tr>
<tr>
<td>$n_2$</td>
<td>0.994451</td>
<td>39</td>
<td>29</td>
<td>-</td>
<td>0.997403</td>
</tr>
<tr>
<td>$n_3$</td>
<td>0.993972</td>
<td>-</td>
<td>37</td>
<td>-</td>
<td>0.998401</td>
</tr>
<tr>
<td>$n_4$</td>
<td>0.990318</td>
<td>47</td>
<td>56</td>
<td>-</td>
<td>0.993521</td>
</tr>
<tr>
<td>$n_5$</td>
<td>0.992716</td>
<td>59</td>
<td>50</td>
<td>-</td>
<td>0.997403</td>
</tr>
<tr>
<td>$n_6$</td>
<td>0.991933</td>
<td>60</td>
<td>66</td>
<td>39</td>
<td>0.991933</td>
</tr>
<tr>
<td>$n_7$</td>
<td>0.994838</td>
<td>-</td>
<td>75</td>
<td>-</td>
<td>0.997603</td>
</tr>
<tr>
<td>$n_8$</td>
<td>0.992769</td>
<td>59</td>
<td>90</td>
<td>-</td>
<td>0.996506</td>
</tr>
<tr>
<td>$n_9$</td>
<td>0.992588</td>
<td>69</td>
<td>86</td>
<td>-</td>
<td>0.997503</td>
</tr>
<tr>
<td>$n_{10}$</td>
<td>0.992209</td>
<td>-</td>
<td>87</td>
<td>-</td>
<td>0.998601</td>
</tr>
</tbody>
</table>

$RT(G) = 87, R(G) = 0.966185 > R_{req}(G) = 0.96$

### 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 $RT_{req}(G)$ 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, 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 $RT_{req}(n_i)$ calculations as follows:

1. The EST defined in (7) considers the earliest available time $avail[k]$ 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 $avail[k]$, but can be migrated; thus, we update EST as follows:

   $$
   \begin{align*}
   \text{EST}(n_i, u_k) &= 0 \\
   \text{EST}(n_i, u_k) &= \max_{n_j \in \text{pred}(n_i)} \{ \text{EST}(n_j, u_k) + c_{i,j} \} \quad (24)
   \end{align*}
   $$

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

   $$
   \begin{align*}
   RT_{req}(n_i, u_k) &= RT_{req}(G) \\
   RT_{req}(n_i, u_k) &= \min_{n_j \in \text{succ}(n_i)} \{ \text{EST}(n_j) - e_{i,j} \} \quad (25)
   \end{align*}
   $$

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

$$
\begin{align*}
\text{EST}(n_{10}, u_1) &= 81 \\
\text{EST}(n_{10}, u_2) &= 73 \\
\text{EST}(n_{10}, u_3) &= 81
\end{align*}
$$

$$
\begin{align*}
RT_{req}(n_{10}, u_1) &= 90 \\
RT_{req}(n_{10}, u_2) &= 90 \\
RT_{req}(n_{10}, u_3) &= 90
\end{align*}
$$

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
ECU \( u_k \) is defined as follows:

\[
S_{i,k} = \{S_{i,k,1}, S_{i,k,2}, S_{i,k,3}|S_{i,k}|\}
\]

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 \( y \)th slack \( S_{i,k,y} \) as follows:

\[
S_{i,k,y} = \{t_y(S_{i,k,y}), t_e(S_{i,k,y})\}
\]

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

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

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 RTreq constraints:

\[
\begin{align*}
\text{EST}(n_i, u_k) &= \max\{\text{EST}(n_i, u_k), t_y(S_{i,k,1})\} \\ 
\text{RTreq}(n_i, u_k) &= \min\{\text{RTreq}(n_i, u_k), t_e(S_{i,k,1})\}
\end{align*}
\]

For example, the new EST and RTreq values of \( n_{10} \) on all ECUs are updated to

\[
\begin{align*}
\text{EST}(n_{10}, u_1) &= 81 \\
\text{EST}(n_{10}, u_2) &= 73 \\
\text{EST}(n_{10}, u_3) &= 81
\end{align*}
\]

\[
\begin{align*}
\text{RTreq}(n_{10}, u_1) &= 90 \\
\text{RTreq}(n_{10}, u_2) &= 90 \\
\text{RTreq}(n_{10}, u_3) &= 90
\end{align*}
\]

### B. Maximizing Reliability

This section iteratively assigns each task to the ECU with the maximum reliability under the response time requirement, such that FFSV2 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:

\[
\text{MET}(n_i, u_k) = \text{RTreq}(n_i, u_k) - \text{EST}(n_i, u_k). \tag{28}
\]

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

\[
\begin{align*}
\text{MET}(n_{10}, u_1) &= 9 < 21 = w_{10,1} \\
\text{MET}(n_{10}, u_2) &= 17 > 7 = w_{10,2} \\
\text{MET}(n_{10}, u_3) &= 9 < 16 = w_{10,3}
\end{align*}
\]

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

\[
\text{MET}(n_i, u_k) \geq w_{i,k}. \tag{30}
\]

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

\[
R(n_i, u_{\text{pr}(i)}) = \max_{u_k \in U} \{R(n_i, u_k)\}. \tag{31}
\]

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

\[
R(n_i, u_{\text{pr}(i)}) \geq R_{\text{heft}}(n_i)
\]

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:

\[
\begin{align*}
\text{AFT}(n_i) &= \text{RTreq}(n_i, u_{\text{pr}(i)}) \tag{32} \\
\text{AST}(n_i) &= \text{RTreq}(n_i, u_{\text{pr}(i)}) - w_{i_{\text{pr}(i)}} \tag{33}
\end{align*}
\]

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 \( \text{taskList} \) in ascending order of rank \( n_i \) 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 \( \text{MET}(n_i, u_k) \geq w_{i,k} \).

4) Verifying functional safety. In lines 16 and 17, FFSV2 calculates the final reliability \( R(G) \) and the response time \( RT(G) \) 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 \( \text{task}_\text{list} \) by ascending order of rank values;
3. while (there are tasks in \( \text{task}_\text{list} \)) do
   4. \( n_i \leftarrow \text{task}_\text{list}.\text{out}() \);
   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

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.

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 \((r_1 \ldots r_7)\), automatic gear box with four tasks \((r_{18} \ldots r_{21})\), antilocking brake system with six tasks \((r_{12} \ldots r_{17})\), wheel angle sensor with two tasks \((r_{18} \ldots r_{19})\), suspension controller with five tasks \((r_{20} \ldots r_{24})\), and body work with seven tasks \((r_{25} \ldots r_{31})\). The ECU number of the system is 16, and the failure rate of each task is within the range of \(10^{-6} \sim 10^{-6}/μs\). 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.

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 trades off 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 \(LB(G) = 736 μs\). Therefore, the response time requirement is changed from 736 to 1536 μs with 100 μs increments, as shown in the x-axis in Fig. 11. Considering that actual response time
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 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. The ECU number and failure rates 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 0.9 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 10 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[N] × 2^{[N/2]}). 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


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 Post-doctoral 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.

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.