High performance real-time scheduling of multiple mixed-criticality functions in heterogeneous distributed embedded systems

Guoqi Xie\textsuperscript{a,b,*}, Gang Zeng\textsuperscript{c}, Liangjiao Liu\textsuperscript{a,b}, Renfa Li\textsuperscript{a,b}, Keqin Li\textsuperscript{a,d}

\textsuperscript{a} College of Computer Science and Electronic Engineering, Hunan University, China
\textsuperscript{b} Key Laboratory for Embedded and Network Computing of Hunan Province, China
\textsuperscript{c} Graduate School of Engineering, Nagoya University, Japan
\textsuperscript{d} Department of Computer Science, State University of New York, New Paltz, New York, USA

\begin{abstract}

The architectures of high-end embedded system have evolved into heterogeneous distributed integrated architectures. The scheduling of multiple distributed mixed-criticality functions in heterogeneous distributed embedded systems is a considerable challenge because of the different requirements of systems and functions. Overall scheduling length (i.e., makespan) is the main concern in system performance, whereas deadlines represent the major timing constraints of functions. Most algorithms use the fairness policies to reduce the makespan in heterogeneous distributed systems. However, these fairness policies cannot meet the deadlines of most functions. Each function has different criticality levels (e.g., severity), and missing the deadlines of certain high-criticality functions may cause fatal injuries to people under this situation. This study first constructs related models for heterogeneous distributed embedded systems. Thereafter, the criticality certification, scheduling framework, and fairness of multiple heterogeneous earliest finish time (F-MHEFT) algorithm for heterogeneous distributed embedded systems are presented. Finally, this study proposes a novel algorithm called the deadline-span of multiple heterogeneous earliest finish time (D-MHEFT), which is a scheduling algorithm for multiple mixed-criticality functions. The F-MHEFT algorithm aims at improving the performance of systems, while the D-MHEFT algorithm tries to meet the deadlines of more high-criticality functions by sacrificing a certain performance. The experimental results demonstrate that the D-MHEFT algorithm can significantly reduce the deadline miss ratio (DMR) and keep satisfactory performance over existing methods.

\end{abstract}

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

1.1. Background

High-end embedded system architectures have evolved into heterogeneous distributed architectures because of the size, weight, and power consumption (SWaP) for cost and high performance benefits. For example, automotive electronic architectures consist of many heterogeneous electronic control units (ECUs) that are distributed on multiple network buses, which are interconnected by a central gateway. Today, a luxury car comprises at least 70 heterogeneous ECUs with approximately 2500 signals [1]. The number of ECUs is expected to increase further in future automotive electronic systems.

The aforementioned distributed architecture leads to an increase in distributed functions (also called functionalities or applications in a few studies) with precedence-constrained tasks in automotive electronic systems [2]. Examples of active safety functions are x-by-wires and adaptive cruise control [3]. The integration of multiple functions in the same architecture is called “integrated architecture,” in which multiple functions can be supported by one ECU and one function can be distributed over multiple ECUs [3]. Integrated architectures are indeed an essential evolution to cope with the SWaP problems and seize the opportunity for cost reduction. This transition requires the development of new models and methods [3].

Integrated architecture drives the integration of several levels of safety-criticality and non-safety-criticality functions into the same platform; criticality levels and mixed-criticality systems have also been introduced [4]. Criticality level is represented by the automotive safety integrity level (ASIL) in the automotive functional safety standard ISO 26262 [5]. ASIL refers to a classification of inherent safety goals required by the standard to ensure the accomplish-
ment of goals in the system; ASIL D and ASIL A represent the highest and lowest criticality levels, respectively [5]. Mixed-criticality systems are new systems that attempt to combine multiple functions with different criticality levels on the same platform.

1.2. Motivations

To make full use of the numerous ECUs in automobiles, efficient scheduling policies are required to achieve substantially high performance improvement. However, scheduling multiple distributed mixed-criticality functions in heterogeneous distributed embedded systems involves the following challenges.

First, many scheduling methods for mixed-criticality systems have been developed in the past years, but such methods are mainly based on periodic and sporadic task models. Many distributed functions have apparent precedence constraints among tasks in high-end heterogeneous distributed embedded systems (e.g., automotive electronic systems). Evidence shows that models for mapping distributed functions are highly criticality to the analysis of automotive electronic systems. A few models, such as time chains [6] and task chains [7], have been employed in automobiles; however, these models are only suitable for simple distributed functions. With the increasing complexity and parallelization of automobile functions, a model that accurately reflects the distributed characteristics of automotive functions is desirable.

In heterogeneous distributed systems, a distributed function with precedence-constrained tasks at a high level is described as a directed acyclic graph (DAG), in which the nodes represent the tasks and the edges represent the communication messages between the tasks [18]. The DAG-based model has also been applied to automotive electronic systems [9,10].

Second, systems and functions in heterogeneous distributed embedded systems involve considerable conflicts. Overall scheduling length (makespan) is the main concern in system performance, whereas deadlines are the major timing constraints of functions. The deadlines of all functions cannot be met in heterogeneous distributed embedded systems, particularly in resource-constrained distributed embedded environments. A high-criticality function (i.e., a function with high criticality level) has a considerably important and strict timing constraint for a given deadline. Missing the deadlines of high-criticality functions results in fatal injuries to people. Most algorithms use fairness policies to reduce the overall makespan of systems in heterogeneous distributed systems; however, these policies could lead to the failure to meet the deadlines of high-criticality functions. Therefore, both performance and timing constraints should be considered to achieve a good makespan and low deadline miss ratio (DMR) [11].

1.3. Our contributions

Our contributions are summarized as follows. First, we construct a series of models for heterogeneous distributed embedded systems from the “distributed computing” and “functional safety” perspectives. Second, we propose a functional level scheduling algorithm with a round-robin fairness policy from the “system performance” perspective. Third, we further propose a functional level scheduling algorithm with a deadline-span-driven policy to achieve satisfactory system performance and low DMR.

The rest of this paper is organized as follows. Section 2 reviews the related literature. Section 3 constructs a series of models for heterogeneous distributed embedded systems. Section 4 proposes the certification method, scheduling framework, and round-robin fairness scheduling. Section 5 proposes a scheduling algorithm with a deadline-span-driven policy. Section 6 verifies the performance ratios of all the proposed methods of this study. Section 7 concludes this study.

2. Related works

High performance is an important concern of heterogeneous distributed systems, whereas timing constraints represent an important requirement of high-criticality functions. This section first reviews the related research for high performance scheduling and then discusses real-time scheduling.

2.1. High performance scheduling

The scheduling of a single distributed function (also called single DAG-based function scheduling) is the basis of the scheduling of multiple distributed functions (also called multiple DAG-based function scheduling). Thus, we briefly introduce the single DAG-based function list scheduling. List scheduling includes two phases: the first phase orders tasks according to the descending order of priorities (task prioritizing), whereas the second phase allocates each task to a proper processor (task allocation). Scheduling tasks for a single DAG-based function with the fastest execution is a well-known NP-hard optimization problem [8]. In [8], Topcuoglu et al. proposed the popular algorithm called the heterogeneous earliest finish time (HEFT) for the single DAG-based function scheduling in heterogeneous distributed systems to reduce makespan to a minimum. The HEFT algorithm uses upward rank values for task ordering and the earliest finish time (EDF) based on the insertion-based policy for task allocation. The aforementioned study further inspired substantial investigations and the development of other algorithms, including constrained EDF (CEFT) [12], predict EDF (PEFT) [13], and heterogeneous selection value (HSV) [1].

The multiple DAG-based functions scheduling of heterogeneous systems also involves two steps, namely, task prioritizing and task allocation. In [14], Honig et al. first proposed a composition approach to merge multiple distributed functions into one new function and then used a single DAG-based function scheduling algorithm (e.g., HEFT) to schedule the new DAG-based function. However, apparent unfairness to functions with short makespans emerges because the upward rank values of these functions are significantly lower than those of functions with long makespans. This approach limits the execution opportunities of functions with short makespans, and such limitation results in an unfairness to them and in a considerably long overall makespan in systems. In [15], Zhao et al. first identified the fairness issue in the scheduling of multiple DAG-based functions. The authors proposed a fairness scheduling algorithm called Fairness with a slowdown-driven policy that ensures the fairness of different functions. Other related studies, such as those on online workflow management (OWM) [16] for overall makespan minimization and fairness dynamic workflow scheduling (FDWS) [17] for minimization of individual functions were conducted.

2.2. Real-time scheduling

The mixed-criticality scheduling problem was first identified and formalized by Vestal [18], whose work has been extended and has inspired further substantial investigations [19–22]. However, the models of these works are only periodic [19,20] and sporadic tasks models [21,22]. Hence, these works only considered mixed-criticality from the “task level” perspective and cannot reflect the distributed characteristics of functions in automobiles. For the functional safety of automobiles, scheduling should be considered at the “functional level” and not at the “task level.”

Some related researches are concerned about function scheduling with deadline constraints [23–25]. However, these solutions are merely for single DAG-based scheduling, and not suitable for multiple DAG-based scheduling issues.
3.2. Criticality level

ISO 26262 identifies four criticality levels denoted by ASILs (i.e., A, B, C, and D) for systematic failures with severity and random hardware failures with exposure (i.e., reliability) of automotive functions. Severity also involves four criticality levels, namely, S0, S1, S2, and S3, where S0 represents the lowest criticality level (i.e., no injuries) and S4 represents the highest criticality level (i.e., life-threatening to fatal injuries) [5,32]. Similar to [31], we do not address the issue of reliability (which is orthogonal to our problem), and we assume that the designer has developed the functions such that they provide the required level of fault tolerance [31]. Hence, $S = \{S_0, S_1, S_2, S_3\}$ is employed to represent a set of the criticality levels of a system. Our systems comprise more than two criticality levels (hence the name multiple-criticality systems) and are thus different from dual-criticality systems, which comprise only two criticality levels [28,29].

3.3. Mixed-criticality function model

A distributed mixed-criticality function is represented by a DAG $F_m = (N, M, C, W, \text{criticality, lowerbound, deadline, makespan})$. $F_m$ represents the mth functions in systems. $N$ represents the set of nodes in $F_m$, and each node $n_i \in N$ represents a task with different worst-case execution times (WCETs) on different ECUs. $M$ is a set of communication edges, and each edge $e_{ij} \in M$ represents the communication message from the node $n_i$ to $n_j$. Accordingly, $c_{ij}$ represents the worst-case transmitting time of $e_{ij}$. Notice that the WCTT includes the gateway processing time of $e_{ij}$. For each $e_{ij}$, $\text{pred}(n_i)$ represents the set of the immediate predecessor tasks of $n_i$, $\text{succ}(n_i)$ represents the set of the immediate successor tasks of $n_i$, $\text{ind}(n_i)$ represents the in-degree of $n_i$, which indicates the cardinality of $\text{pred}(n_i)$, and $\text{outd}(n_i)$ represents the out-degree of $n_i$, which indicates the cardinality of $\text{succ}(n_i)$. For simplicity, a function comprises only one entry task, which has no predecessor task and is denoted as $n_{\text{entry}}$ and one exit task, which has no successor task and is denoted as $n_{\text{exit}}$. $W$ is an $|N| \times |P|$ matrix, in which $w_{ik}$ denotes the WCTT of $n_i$ runs on $p_k$. The aforementioned parameters are the basic properties of the distributed mixed-criticality functions of heterogeneous distributed systems, and are used by several algorithms (e.g., HEFT [8] and HSV [11]).

For a distributed mixed-criticality function, the remaining attributes (criticality, lowerbound, deadline, and makespan) need to be used. criticality $\in S$ represents the criticality level of $F_m$, lowerbound and deadline represent the lower-bound and deadline of $F_m$, respectively. criticality, lowerbound, and deadline must be certified by a certification authority (CA) (refer to Section 4.1 for concrete certification). makespan represents the actual makespan of $F_m$ and is generated with the proposed algorithm.

3.4. Mixed-criticality systems model

A mixed-criticality system comprises of multiple distributed mixed-criticality functions and is denoted as $MS = \{F_1, F_2, \ldots , F_{|MS|} \}$, where criticality indicates the current criticality level of the system. In distinguishing the ambiguities, we use $MS.\text{criticality}$ to express the criticality of $MS$, and use $F_m.\text{criticality}$ to express the criticality of $F_m$. Other attributes use the same expression. $MS.\text{criticality}$ can be changed to high-criticality levels and back to low-criticality levels. A change in $MS.\text{criticality}$ indicates a switch in system mode. $F_m$ can only be executed on the modes in which $F_m.\text{criticality}$ is higher than or equal to $MS.\text{criticality}$. $MS.\text{makespan}$ represents the overall makespan of $MS$ and reflects system performance. $MS.\text{makespan}$ is also generated with the proposed algorithm.
This study considers static scheduling and does not dynamic scheduling. The reason is that we can observe the results for an optimistic system analysis and design. In static scheduling, all functions are released simultaneously. This type of scheduling is thus widely used in the many functions of automobiles. For example, integrated safety systems include the functions of anti-lock braking system (ABS), acceleration slip regulation (ASR), and electronic stability program (ESP). To avoid possible collision in emergent state, these functions will released simultaneously. If a task is allocated to different ECUs with partitioned scheduling, then such task generates heavy communication cost (i.e., WCCT of messages) between any two ECUs. Hence, different from the partitioned scheduling in [28–31], this study considers the global non-preemptive scheduling.

3.5 Motivating example

**Fig. 2** shows a motivating example of mixed-criticality systems with three functions, namely, F1, F2, and F3, and with F1.criticality = S2, F2.criticality = S2, and F3.criticality = S0. The shapes of F2 and F3 functions are similar to the examples of [33], whereas F1 is a relative complex function. **Table 1** shows the WCETs of tasks for F1, F2, and F3 in Fig. 2. The example shows ten tasks for F1, five tasks for F2, and six tasks for F3. This study assumes three ECUs for the system in this motivating example. Although the example is simple, it involves three ECUs, three functions, and three criticality levels. Hence, this example can reflect the characteristics of multiple ECUs, multiple functions, and multiple criticality levels for heterogeneous distributed embedded systems. The weight 18 of the edge between task F1,n1 and task F1,n2 represents the WCET of F1,n1,2 if F1,n1 and F1,n2 are not assigned in the same ECU. The weight 14 of F1,n1 and p1 in **Table 1** represents the WCET and is denoted as F1,n1 = 14. **Section 4.1** explains the meaning of ranku, lowerbound, and deadline of **Table 1**.

### 4. Certification and framework

#### 4.1. Lower-bound and deadline

The HEFT algorithm is the most popular single DAG-based function scheduling algorithm for reducing makespan to a minimum while achieving low complexity and high performance in heterogeneous distributed systems [8]. The two-phase HEFT algorithm has two important contributions.

First, the HEFT algorithm uses the upward rank value (ranku) of a task given by **Eq. (1)** as the common task priority standard. In this case, the tasks are ordered according to the decreasing order of ranku. **Table 1** shows the upward rank values of all the tasks (Fig. 2), which are obtained with **Eq. (1)**:

\[
\text{rank}_u(n_i) = \text{WCET} + \max_{n_j \text{succeed}(n_i)} \{ c_{ij} + \text{rank}_u(n_j) \}. \tag{1}
\]

Second, the attributes EST(ni, pk) and EFT(ni, pk) represent the earliest start time (EST) and the EFT, respectively, of task ni on processor pk. EFT(ni, pk) is considered the common task allocation criterion because it can meet the local optimal of each precedence-constrained task by using the greedy policy. The aforementioned
attributes are calculated as follows:

\[
\begin{align*}
EST(n_{entry}, p_k) &= 0; \\
EST(n_j, p_k) &= \max \left( \text{avail}[p_k], \max_{n_i \in \text{pred}(n_j)} \{ AFT(n_i) + c_{i,j} \} \right),
\end{align*}
\]

and

\[
EFT(n_j, p_k) = EST(n_j, p_k) + w_{j,k}.
\]

The earliest available time when processor \( p_k \) is ready for task execution. \( AFT(n_i) \) is the actual finish time of task \( n_i \). \( n_j \) is allocated to the processor with the minimum \( EFT \) by using the insertion-based scheduling policy that \( n_j \) can be inserted into the slack with the minimum \( EFT \).

High-criticality functions have strict real-time requirements. Therefore, a convincing standard algorithm needs to be employed by CAs to assess the lower-bound of a distributed mixed-criticality function. The lower-bound refers to the minimum makespan of a function when all processors are monopolized by the function by using the standard single DAG-based function scheduling algorithm. As a known algorithm with low complexity and high performance, the HEFT algorithm can be and should be selected as the standard algorithm for certifying distributed mixed-criticality functions. The present study uses the HEFT algorithm as the certification algorithm to explain the certification process (other algorithms can be easily selected and employed as an alternative to the HEFT algorithm). The lower-bound of function \( F_m \) is calculated as

\[
F_m^{\text{lowerbound}} = AFT(F_m, n_{exit}).
\]

where \( F_m \), \( n_{exit} \) represents the exit task of \( F_m \). Moreover, the CA also provides a deadline for each function on the basis of the lower-bound and actual physical time requirement obtained after the hazard analysis and risk assessment. Note that the concrete hazard analysis and risk assessment are not discussed in this paper because our main focus is scheduling; that is, the deadline of each function has been obtained in advance. Table 1 lists the lower-bound and deadline of each function of the motivating example.

4.2. Scheduling framework

We present the scheduling framework of multiple distributed mixed-criticality functions for heterogeneous distributed embedded systems (Fig. 3). The scheduling framework comprises three priority queues, namely, task priority, common ready, and task allocation queues.

1. In the task priority queue (task_priority_queue) of each function, tasks are ordered according to decreasing \( \text{rank}_k(n_i) \).
2. In the common ready queue (common_ready_queue) of systems for storing ready tasks (selecting one ready task with maximum \( \text{rank}_k \) from each function), tasks are also ordered according to decreasing \( \text{rank}_k(n_i) \).
3. The task allocation queue (task_allocation_queue) of each processor is for storing allocated tasks.

We present a round-robin fairness policy on the basis of the scheduling framework. Each step in the proposed fairness policy (Fig. 3) is described as follows:

Step (1) Task priority: Put the tasks of each function into the corresponding task priority queue task_priority_queue according to the decreasing order of \( \text{rank}_k(n_i) \).

Step (2) Task ready with fairness policy: Select the ready tasks with the highest \( \text{rank}_k(n_i) \) from each function, and put them into the common_ready_queue according to the decreasing order of \( \text{rank}_k(n_i) \).

Step (3) Task allocation with fairness policy: Select a task with the highest \( \text{rank}_k(n_i) \) from the common_ready_queue, and put it into the task allocation queue of the processor \( p_k \) (denoted as task_allocation_queue \( p_k \)) with minimum \( EFT(n_i, p_k) \) using the insertion-based scheduling policy.

Step (4) Task execution: Execute these tasks on their corresponding processors after assigning them to the task allocation queues.

This study proposes a scheduling algorithm called the fairness of multiple heterogeneous earliest finish time (F_MHEFT) for heterogeneous distributed embedded systems. The steps are described in Algorithm 1. The time complexity of the F_MHEFT algorithm is analyzed as follows. All functions must be traversed during scheduling. This requirement can be fulfilled in \( O(\text{IMS}) \) time. All the tasks of a function can be scheduled in \( O(N_{max}) \) time (\( N_{max} = \max(|F_1.N|, |F_2.N|, ..., |F_{\text{MSI}}.N|) \)). The EFT values of all tasks can be computed in \( O(N_{max} \times |P|) \) time. Thus, the complexity of the F_MHEFT algorithm is \( O(\text{IMS} \times N_{max} \times |P|) \).

Fig. 4 shows the scheduling process of the F_MHEFT algorithm for the motivating example. Accordingly, Table 2 shows the task allocation steps, each of which indicates a fairness for all functions. The overall makespan of the system is 100; however, all functions miss their deadlines.

The similar method for minimization of individual functions is FDWS [17]. The main difference between FDWS and F_MHEFT in
Algorithm 1 F_MHEFT Algorithm

Input: $P = \{p_1, p_2, ..., p_{|P|}\}$, $MS = \{F_1, F_2, ..., F_{|MS|}\}$
Output: $\{F_1.makespan, F_2.makespan, ..., F_{|MS|}.makespan\}$

1: Calculate $rank_i$ for all tasks of all functions in MS, and put these tasks into corresponding $task.priority\_queue(F_m)$;
2: while (there are tasks to be allocated) do
3: for $(m = 1; m <= |MS|; m++)$ do
4: $n_i = task.priority\_queue(F_m).out()$;
5: $common.ready\_queue.put(n_i)$; //select one task from each task priority queue, and put it into the common ready queue;
6: end for
7: while ($common.ready\_queue.empty()$) do
8: $n_i = common.ready\_queue.out()$; //select one task from the common ready queue to be allocated
9: assign $n_i$ to $task.allocation\_queue(p_k)$ the minimum EFT using the insertion-based scheduling policy;
10: end while
11: end while

Table 2
Task allocation steps of the F_MHEFT algorithm for the motivating example.

<table>
<thead>
<tr>
<th>Step</th>
<th>Task allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$F_1.n_1, F_1.n_2, F_2.n_1$</td>
</tr>
<tr>
<td>2</td>
<td>$F_1.n_2, F_1.n_3, F_2.n_4$</td>
</tr>
<tr>
<td>3</td>
<td>$F_1.n_4, F_2.n_3, F_2.n_5$</td>
</tr>
<tr>
<td>4</td>
<td>$F_1.n_2, F_1.n_5, F_2.n_1$</td>
</tr>
<tr>
<td>5</td>
<td>$F_1.n_6, F_2.n_4, F_2.n_3$</td>
</tr>
<tr>
<td>6</td>
<td>$F_2.n_6, F_3.n_1$</td>
</tr>
<tr>
<td>7</td>
<td>$F_3.n_1$</td>
</tr>
<tr>
<td>8</td>
<td>$F_1.n_7$</td>
</tr>
<tr>
<td>9</td>
<td>$F_2.n_8$</td>
</tr>
<tr>
<td>10</td>
<td>$F_3.n_{10}$</td>
</tr>
</tbody>
</table>

static scheduling is FDWS and F_MHEFT select the task with highest $rank_i(n_i)$ (Eq. (5)) and $rank_{n_i}(n_i)$ from the common ready queue, respectively, in Step (3) of Fig. 3:

$$rank_i(F_m.n_i) = \frac{1}{PR_T(F_m)} \times \frac{1}{CPL(F_m)}.$$  

where $PR_T(F_m)$ and $CPL(F_m)$ represent the percentage of remaining task (PR_T) number and the critical path length (CPL) of the function $F_m$, respectively. We will see that F_MHEFT would outperform FDWS on large-scale function sets in experiments. Moreover, F_MHEFT is the basis of our subsequent works to reduce DMR.

5. Mixed-criticality scheduling

We can use the F_MHEFT algorithm (Algorithm 1) to schedule all functions with different criticality levels and achieve a short makespan. However, the deadlines of many high-criticality functions may be missed. To meet the deadlines of high-criticality functions and reduce the DMRs of systems, a novel solution based on F_MHEFT is proposed and discussed in this section.

5.1. Deadline-span

The CA uses the HEFT algorithm to generate the lower-bounds of functions and provides a deadline for each function on the basis of the lower-bound of the function obtained after the hazard analysis and risk assessment. Each task should have a lower-bound and a deadline. The following definition is provided as an explanation.

Definition 1. (Deadline-span) The deadline-span of a function represents the value of the deadline minus the lower-bound of the function, that is,

$$F_m.deadlinespan = F_m.deadline - F_m.lowerbound.$$  

$F_m.deadline$ and $F_m.lowerbound$ are provided and calculated by the CA; thus, $F_m.deadlinespan$ can be obtained easily. The deadline of task $n_i (n_i \in F_m)$ can then be generated. Thus,

$$deadline(F_m.n_i) = lowerbound(F_m.n_i) + F_m.deadlinespan.$$  

where $lowerbound(F_m.n_i) = AF_T(F_m.n_i)$ for certification. The deadline-spans of all functions are obtained using Eq. (6) ($F_1.deadlinespan = 10$, $F_2.deadlinespan = 10$, and $F_3.deadlinespan = 10$) of the motivating example. Thereafter, the deadlines of all tasks are calculated using Eq. (7). Tables 3, 4, and 5 show all the values.

5.2. The D_MHEFT algorithm

On the basis of the analysis in Section 5.1, we propose a scheduling algorithm called the deadline-span of multiple heterogeneous earliest finish time (D_MHEFT) to meet the deadlines of high-criticality functions and consequently achieve low DMR and satisfactory system performance. The steps are described in Algorithm 2.

The time complexity of the D_MHEFT algorithm should be $O(|MS| \times N_{max} \times |P|)$, which is equal to that of the F_MHEFT algorithm. In other words, changing the systems criticality does not increase the time complexity. A function can be scheduled only when its criticality is higher than or equal to the criticality of the system. The D_MHEFT algorithm is driven by the change in the criticality of the system. The main idea of the D_MHEFT algorithm is that when the deadline of any high-criticality function cannot be met, the system's criticality is changed up to the criticality of the function. Then, only functions whose criticality levels are equal to or larger than the system's criticality are scheduled. The system's criticality is changed down to the lowest criticality level after the function is scheduled completed. Finally, the remaining tasks of low-criticality functions are scheduled.
Table 3
Lower-bounds and deadlines of tasks in the F1 function.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>F1,n1</th>
<th>F1,n2</th>
<th>F1,n3</th>
<th>F1,n4</th>
<th>F1,n5</th>
<th>F1,n6</th>
<th>F1,n7</th>
<th>F1,n8</th>
<th>F1,n9</th>
<th>F1,n10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower-bound</td>
<td>9</td>
<td>40</td>
<td>28</td>
<td>26</td>
<td>38</td>
<td>42</td>
<td>49</td>
<td>62</td>
<td>68</td>
<td>80</td>
</tr>
<tr>
<td>Deadline</td>
<td>19</td>
<td>50</td>
<td>38</td>
<td>36</td>
<td>48</td>
<td>52</td>
<td>59</td>
<td>72</td>
<td>78</td>
<td>90</td>
</tr>
</tbody>
</table>

Fig. 5. MS.criticality = S0; the tasks of functions F1, F2, and F3 are scheduled with the fairness policy until makespan(F2,n1) > deadline(F2,n1).

Algorithm 2 D_MHEFT Algorithm

Input: P = \{p1, p2, ..., p|P|\}, MS = \{F1, F2, ..., F_M\}, MS.makespan
Output: \{F1.makespan, F2.makespan, ..., F_M.makespan\}, MS.makespan

1: Calculate ranku for all tasks of all functions in MS, and put these tasks into corresponding task_priority_queue(Fm);
2: MS.criticality = S0;
3: while (there are tasks to be allocated) do
4:     (m = 1; m <= |MS|; m++)
5:         if (Fm.criticality < MS.criticality) then
6:             continue;
7:         end if
8:     end for
9:     while (common_ready_queue.empty()) do
10:         Fm,n1 = common_ready_queue.out(); //select one task from each task priority queue, and put it into the common ready queue;
11:         end for
12:     while (not (common_ready_queue.empty())) do
13:         Fm,n1 = common_ready_queue.out(); //select one task from the common ready queue to be allocated;
14:         Assign Fm,n1 to task_allocation_queue(p1) with the minimum EFT using the insertion-based scheduling policy;
15:         if (makespan(Fm,n1) > deadline(Fm,n1) and Fm.criticality > MS.criticality) then
16:             Cancel the allocation of tasks in this round except for the tasks in scheduled completed functions;
17:             Put the cancelled tasks back to individual task priority queues;
18:             Clear the tasks in the common ready queue and put them back to individual task priority queues;
19:             Change the criticality of the system to the criticality of Fm, namely, MS.criticality = Fm.criticality.
20:         end if
21:     end while
22:     if (Fm is the function causing the criticality of the system to be changed up and Fm is scheduled completed) then
23:         Clear the tasks in the common ready queue and put them back to individual task priority queues;
24:         Change the criticality of the system to S0, namely, MS.criticality = S0.
25:     end if
26: end while

Table 4
Lower-bounds and deadlines of tasks in the F2 function.

<table>
<thead>
<tr>
<th>Task</th>
<th>F2,n1</th>
<th>F2,n2</th>
<th>F2,n3</th>
<th>F2,n4</th>
<th>F2,n5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower-bound</td>
<td>4</td>
<td>20</td>
<td>22</td>
<td>21</td>
<td>36</td>
</tr>
<tr>
<td>Deadline</td>
<td>18</td>
<td>32</td>
<td>41</td>
<td>46</td>
<td>59</td>
</tr>
</tbody>
</table>

Table 5
Lower-bounds and deadlines of tasks in the F3 function.

<table>
<thead>
<tr>
<th>Task</th>
<th>F3,n1</th>
<th>F3,n2</th>
<th>F3,n3</th>
<th>F3,n4</th>
<th>F3,n5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower-bound</td>
<td>8</td>
<td>22</td>
<td>31</td>
<td>36</td>
<td>49</td>
</tr>
<tr>
<td>Deadline</td>
<td>18</td>
<td>32</td>
<td>41</td>
<td>46</td>
<td>59</td>
</tr>
</tbody>
</table>

Figs. 5–9 show the scheduling steps and the results of the D_MHEFT algorithm for the motivating example.

In Fig. 5 with MS.criticality = S0, the tasks (F1,n1, F1,n1, F2,n1, F2,n2, F2,n3, F2,n4, F2,n5, F2,n6, and F2,n7) of F1, F2, and F3 functions are scheduled with the fairness policy until makespan(F2,n3) > deadline(F2,n3). Thereafter, the allocated tasks (i.e., F1,n3, F1,n3, F2,n4, F2,n5, F2,n5, F2,n7, and F2,n7) denoted as shadowgraphs in the current and previous rounds of the current round are cancelled. Given that the current round is not completed, the next round would still be the allocation of current round if the allocation of current round is merely cancelled. Hence, in this case, the two rounds need to be cancelled.

Fig. 6 shows that the criticality of the system is changed up to MS.criticality = S2 because F2.criticality = S2. Thereafter, the tasks (i.e., F1,n1, F1,n1, F2,n2, F2,n3, F1,n4, F2,n5, F2,n6, F1,n5, and F2,n7) of F1 and F2 functions are scheduled with the fairness policy until all the scheduled tasks of F2 are allocated. In this mode, F2.makespan = 44, which is less than F2.deadline = 46. Hence, F2 meets its deadline and is safe.

Fig. 7 shows that the system criticality is changed down to MS.criticality = S0. Thereafter, the tasks (i.e., F1,n2 and F1,n6) of F1 and F2 functions are scheduled with the fairness policy until makespan(F1,n6) > deadline(F1,n6). Thereafter, the allocated tasks (e.g., F1,n5, F2,n2, and F1,n5) denoted as shadowgraphs in the current and previous rounds of the current round are cancelled. Note that F2 has been completed, and its tasks cannot be cancelled.

Fig. 8 shows that the criticality of the system is changed up to MS.criticality = S3 because F1.criticality = S3. Thereafter, the tasks (i.e., F1,n5, F1,n6, F1,n7, F1,n8, and F1,n10) of F1 function are scheduled with the fairness policy until all the scheduled tasks of F1 are allocated. In this mode, F1.makespan = 89, which is less than F1.deadline = 90. Hence, F1 meets its deadline and is safe.
Fig. 6. The criticality level of the system is changed up to $MS_{\text{criticality}} = S_2$. The tasks of the $F_1$ and $F_2$ functions are scheduled with the fairness policy until all the scheduled tasks of $F_2$ are allocated.

Fig. 7. System criticality is changed down to $MS_{\text{criticality}} = S_0$. The tasks of the $F_1$ and $F_3$ functions are scheduled with the fairness policy until $\text{makespan}(F_1,n_6) > \text{deadline}(F_1,n_6)$.

Fig. 8. The criticality level of the system is changed up to $MS_{\text{criticality}} = S_3$. The tasks of the $F_1$ function are scheduled until all the scheduled tasks of $F_1$ are allocated.

Fig. 9. System criticality is changed down to $MS_{\text{criticality}} = S_0$. The tasks of the $F_3$ function are scheduled with the fairness policy.

Fig. 9 shows that the system criticality is changed down to $MS_{\text{criticality}} = S_0$. Thereafter, the tasks (i.e., $F_3,n_2, F_3,n_3, F_3,n_5, F_3,n_6$) of $F_3$ function are scheduled with the fairness policy. Considering that the criticality of $F_3$ is $S_0$, it cannot be changed up. Thereafter, for $F_3$, $F_3\_\text{makespan} = 105$, which is larger than $F_3\_\text{deadline} = 64$. Hence, $F_3$ misses its deadline; however, it is a non-safety function and will not cause fatal injuries to people in this situation.

On the basis of the results of the F_MHEFT and D_MHEFT algorithms for the motivating example, we can make the following observations. (1) The F_MHEFT algorithm has a short system makespan value of 100, but all functions miss their deadlines. (2) The D_MHEFT algorithm meets the deadline of $F_1$ and $F_2$ and still has a satisfactory system makespan of 105. (3) The D_MHEFT algorithm has the same time complexity as the F_MHEFT algorithm.
6. Performance evaluation

6.1. Experimental metrics

The performance metrics selected for comparison are the DMR of functions [11] and overall makespan of systems [17]. Overall makespan is given by

$$\text{MS.makespan} = \max_{f \in \text{MS}} \text{makespan}.$$  \hfill (8)

DMR is calculated using

$$\text{DMR}(S) = \frac{\text{MS}^{\text{miss}}(S)}{\text{MS}(S)},$$  \hfill (9)

where $\text{MS}^{\text{miss}}(S_i)$ represents the number of the functions with criticality level $S_i$ missing their deadlines and $\text{MS}(S_i)$ represents the number of all the functions with criticality level $S_i$.

We implemented the simulated CAN clusters with four buses using Java on a standard desktop computer. Considering that there are at least about 70 ECUs in a luxury car [1], this platform contains 100 ECUs and can generate and run a variety of functions samples with different criticality levels (including active safety, passive safety, and non-safety functions). Function samples are randomly generated depending on the following realistic parameters of automotive functions. $100\mu s \leq w_{i,k} \leq 400\mu s$, $100\mu s \leq c_{i,j} \leq 400\mu s$, $8 \leq |N| \leq 30$. Three algorithms (i.e., FDWS [17], F_MHEFT, and D_MHEFT) are used for the experiment and then compared for verification. The reason for choosing the FDWS algorithm is that it is a state-of-the-art fairness algorithm that minimizes individual makespans of functions.

6.2. Experimental results

**Experiment 1.** This experiment is to compare the overall makespans and DMRs on different scale function sets. Function samples are randomly selected from the sample space. The number of functions is changed from 40 to 640. The number of functions reflects the workload of the systems. These functions are evenly distributed to four criticality levels ($S_0$, $S_1$, $S_2$, and $S_3$). The deadline-span of each function $F_m$ is calculated as $F_m.\text{deadline-span} = F_m.\text{lowerbound}/40$. Three algorithms (i.e., FDWS [17], F_MHEFT, and D_MHEFT) are used for the experiment and then compared for verification.

Table 6 shows the DMRs for varying numbers of functions using the three algorithms. In overall, D_MHEFT generates considerably lower DMRs than FDWS and F_MHEFT in all cases. Specificity, the DMRs are extremely high (at least 0.95 for all the criticality levels) for FDWS and F_MHEFT in middle and large-scale function sets (more than 160 functions). Moreover, all the results reach 1.0 when the function number reaches or exceeds 320. The DMRs of functions with $S_3$ generated by D_MHEFT are always much lower than FDWS and F_MHEFT. In other words, D_MHEFT can meet the deadlines of more active-safety functions. For example, when $|S| = 160$ and $|S| = 320$, the DMRs generated by FDWS and F_MHEFT are 1.0, whereas those generated by D_MHEFT are merely 0.375 and 0.5, respectively.

### Table 6

<table>
<thead>
<tr>
<th>Criticality levels</th>
<th>Algorithms</th>
<th>FDWS</th>
<th>F_MHEFT</th>
<th>D_MHEFT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$S_0$</td>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$S_3$</td>
</tr>
<tr>
<td>$</td>
<td>S</td>
<td>= 40$</td>
<td>0.8</td>
<td>0.4</td>
</tr>
<tr>
<td>$</td>
<td>S</td>
<td>= 80$</td>
<td>0.6</td>
<td>0.65</td>
</tr>
<tr>
<td>$</td>
<td>S</td>
<td>= 160$</td>
<td>1.0</td>
<td>0.975</td>
</tr>
<tr>
<td>$</td>
<td>S</td>
<td>= 320$</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>$</td>
<td>S</td>
<td>= 640$</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

**Experiment 2.** Given that the system cannot meet the deadlines of all the functions with $S_1$ in Experiment 1, the number of such functions should be reduced. In this experiment, the total number of functions is fixed at 320. These functions are first evenly distributed to four criticality levels ($S_0$, $S_1$, $S_2$, and $S_3$), then partial functions with the criticality level $S_3$ is changed to $S_0$. The deadline-span of each function $F_m$ is still fixed as $F_m.\text{deadline-span} = F_m.\text{lowerbound}/40$. Considering that F_MHEFT outperforms FDWS in the previous experiments, only F_MHEFT and D_MHEFT are used for the experiment and then compared for verification.

**Table 7** shows the overall makespans for varying numbers of functions using the three algorithms. Except for the small-scale function sets ($|S| = 20$), F_MHEFT outperform FDWS in middle and large-scale function sets. Similarly, except for the case of small-scale ($|S| = 20$ and $|S| = 40$), both FDWS and F_MHEFT exhibits better performance than D_MHEFT. In other words, with the large number of functions exist in systems, F_MHEFT is superior than FDWS and D_MHEFT.

According to the results of Table 6 and 7, it is verified that D_MHEFT can significantly reduce the DMR by sacrificing certain performance.

**Table 8** shows the DMRs for varying numbers of functions with $S_0$ and $S_3$ using the F_MHEFT and D_MHEFT algorithms. The DMRs using F_MHEFT are always 1.0 in all different criticality levels. However, the DMR of functions with $S_3$ using D_MHEFT is reduced step by step. When the number is reduced to 10, the DMR is 0. Meanwhile, the DMR of functions with $S_2$ using D_MHEFT are also reduced from 1.0 to 0.6875. That is, we implement the objective that the system meet the deadlines of all the functions with $S_1$.

**Table 9** shows the overall makespans for varying numbers of functions with $S_0$ and $S_3$ using the F_MHEFT and D_MHEFT algorithms. We can see that the overall makespan using F_MHEFT is always fixed as 9686 $\mu$s. D_MHEFT generate longer makespan than F_MHEFT, and the differences are relative values (3808-5355 $\mu$s).

This experiment indicates that D_MHEFT can significantly reduce the DMR and keep satisfactory performance.

**Experiment 3.** Another method for meeting the deadlines of all the functions with $S_3$ is modifying the deadline-span. Hence, this experiment aims to modify the deadline-span of each function to observe the results. In this experiment, the number of functions is fixed with 320. These functions are also evenly distributed to four criticality levels ($S_0$, $S_1$, $S_2$, and $S_3$). The deadline-span of
Table 8
DMRs for varying numbers of functions with S₀ and S₁ using F_MHEFT and D_MHEFT.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>F_MHEFT</th>
<th>D_MHEFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₀</td>
<td>S₁</td>
<td>S₂</td>
</tr>
<tr>
<td>[MS(S₀)] = 80, [MS(S₁)] = 80, [MS(S₂)] = 80, [MS(S₃)] = 80</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>[MS(S₀)] = 100, [MS(S₁)] = 80, [MS(S₂)] = 80, [MS(S₃)] = 60</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>[MS(S₀)] = 120, [MS(S₁)] = 80, [MS(S₂)] = 80, [MS(S₃)] = 40</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>[MS(S₀)] = 140, [MS(S₁)] = 80, [MS(S₂)] = 80, [MS(S₃)] = 20</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>[MS(S₀)] = 150, [MS(S₁)] = 80, [MS(S₂)] = 80, [MS(S₃)] = 10</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 9
Overall makespans (μs) for varying numbers of functions with S₀ and S₁ using F_MHEFT and D_MHEFT.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>F_MHEFT</th>
<th>D_MHEFT</th>
<th>DIFFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₀</td>
<td>S₁</td>
<td>S₂</td>
<td>S₃</td>
</tr>
<tr>
<td>[MS(S₀)] = 80, [MS(S₁)] = 80, [MS(S₂)] = 80, [MS(S₃)] = 80</td>
<td>9686</td>
<td>14,089</td>
<td>4403</td>
</tr>
<tr>
<td>[MS(S₀)] = 100, [MS(S₁)] = 80, [MS(S₂)] = 80, [MS(S₃)] = 60</td>
<td>9686</td>
<td>15,041</td>
<td>5355</td>
</tr>
<tr>
<td>[MS(S₀)] = 120, [MS(S₁)] = 80, [MS(S₂)] = 80, [MS(S₃)] = 40</td>
<td>9686</td>
<td>14,593</td>
<td>4907</td>
</tr>
<tr>
<td>[MS(S₀)] = 140, [MS(S₁)] = 80, [MS(S₂)] = 80, [MS(S₃)] = 20</td>
<td>9686</td>
<td>14,586</td>
<td>4900</td>
</tr>
<tr>
<td>[MS(S₀)] = 150, [MS(S₁)] = 80, [MS(S₂)] = 80, [MS(S₃)] = 10</td>
<td>9686</td>
<td>13,494</td>
<td>3808</td>
</tr>
</tbody>
</table>

Table 10
DMRs for varying deadline-spans using F_MHEFT and D_MHEFT.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>F_MHEFT</th>
<th>D_MHEFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₀</td>
<td>S₁</td>
<td>S₂</td>
</tr>
<tr>
<td>Fₘ deadlinespan = Fₘ lowerbound/40</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Fₘ deadlinespan = Fₘ lowerbound/30</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Fₘ deadlinespan = Fₘ lowerbound/20</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Fₘ deadlinespan = Fₘ lowerbound/10</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Fₘ deadlinespan = Fₘ lowerbound/5</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 11
Overall makespans (μs) for varying deadline-spans using F_MHEFT and D_MHEFT.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>F_D_HFT</th>
<th>D_D_HFT</th>
<th>DIFFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₀</td>
<td>S₁</td>
<td>S₂</td>
<td>S₃</td>
</tr>
<tr>
<td>Fₘ deadlinespan = Fₘ lowerbound/40</td>
<td>9805</td>
<td>13,966</td>
<td>4161</td>
</tr>
<tr>
<td>Fₘ deadlinespan = Fₘ lowerbound/30</td>
<td>9805</td>
<td>13,822</td>
<td>4017</td>
</tr>
<tr>
<td>Fₘ deadlinespan = Fₘ lowerbound/20</td>
<td>9805</td>
<td>13,898</td>
<td>4175</td>
</tr>
<tr>
<td>Fₘ deadlinespan = Fₘ lowerbound/10</td>
<td>9805</td>
<td>14,089</td>
<td>4284</td>
</tr>
<tr>
<td>Fₘ deadlinespan = Fₘ lowerbound/5</td>
<td>9805</td>
<td>14,089</td>
<td>4284</td>
</tr>
</tbody>
</table>

7. Conclusions

We develop a novel functional level scheduling algorithm called D_MHEFT with a deadline-span-driven policy to achieve satisfactory system performance and low DMR of multiple distributed mixed-criticality functions in heterogeneous distributed embedded systems. The D_MHEFT algorithm is implemented by changing or down the system’s criticality to achieve fair scheduling of functions whose criticality levels are larger than or equal to the system’s criticality. The extensive experiments conducted in this study demonstrate that the D_MHEFT algorithm achieves satisfactory overall makespan when meeting the deadlines of more high-criticality functions compared with existing methods. Moreover, D_MHEFT can provide certain design guideline to deadline certification in actual system design.

Acknowledgments

The authors would like to express their gratitude to the anonymous reviewers for their constructive comments which have helped to improve the quality of the paper. This study was partially supported by the National High-Tech Research and Development.
Plan of China with Grant No. 2012AA01A301-01, the Key Program of National Natural Science Foundation of China with Grant No. 61432005, and the National Natural Science Foundation of China with Grant Nos. 61173036 and 61370095.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.sysarc.2016.04.008

References


Guoqi Xie received his Ph.D. degree in computer science from Hunan University, China, in 2014. He was a postdoctoral researcher at Nagoya University, Japan, from 2014 to 2015. He is currently a postdoctoral researcher at Hunan University, China, since 2015. His major interests include embedded systems, distributed systems, in-vehicle networks, real-time systems, and cyber-physical systems.

Gang Zeng is an associate professor at the Graduate School of Engineering, Nagoya University. He received his Ph.D. degree in Information Science from Chiba University in 2006. From 2006 to 2010, he was a researcher, and then assistant professor at the Center for Embedded Computing Systems (NCES), the Graduate School of Information Science, Nagoya University. His research interests mainly include power-aware computing and real-time embedded system design. He is a member of IEEE and IPSJ.

Liangjiao Liu received his master degree from Hunan University, China, in 2009. He is currently working toward the Ph.D. degree at Hunan University, China. His research interests include embedded systems, distributed systems, and cyber-physical systems.

Renfa Li is a full professor of computer science and electronic engineering, and the dean of College of Computer Science and Electronic Engineering, Hunan University, China. He is the director of the Key Laboratory for Embedded and Network Computing of Hunan Province, China. He is also an expert committee member of National Supercomputing Center in Changsha, China. His major research includes embedded systems, distributed systems, and cyber-physical systems. He is a senior member of IEEE, and a senior member of ACM.

Keqin Li is a SUNY Distinguished professor of computer science. 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. He has published over 370 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, IEEE Transactions on Cloud Computing, Journal of Parallel and Distributed Computing. He is an IEEE fellow.