Journal of Systems Architecture 70 (2016) 3–14

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

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

Guoqi Xie[a,b,*], Gang Zeng[c], Liangjiao Liu[a,b], Renfa Li[a,b], Keqin Li[a,d]

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

ARTICLE INFO

Article history:
Received 8 September 2015
Revised 7 December 2015
Accepted 14 April 2016
Available online 28 April 2016

Keywords:
Criticality level
Deadline missed ratio (DMR)
Deadline-span
High performance
Real-time

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.

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-

* Corresponding author.
E-mail addresses: xgqman@gmail.com, xgqman@hnu.edu.cn (G. Xie), sogo@ertl.jp (G. Zeng), llj1984109@qq.com (L. Liu), lirenfa@hnu.edu.cn (R. Li), lik@newpaltz.edu (K. Li).

http://dx.doi.org/10.1016/j.sysarc.2016.04.008
1383-7621/

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 [1,8]. 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 (EFT) based on the insertion-based policy for task allocation. The aforementioned study further inspired substantial investigations and the development of other algorithms, including constrained EFT (CEFT) [12], predict EFT (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.

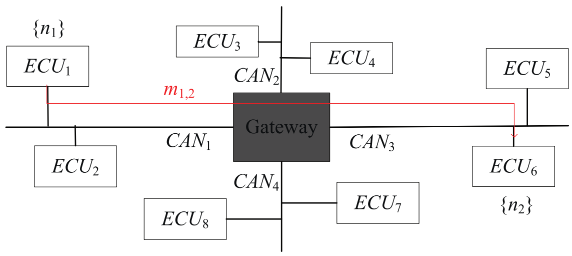

Fig. 1. Example of a CAN cluster with four buses interconnected by a central gateway.

In recent years, the functional level scheduling of multiple DAG-based distributed functions in heterogeneous distributed architectures has been studied. In [26], Wang et al. presented maximizing throughput of multi-DAGs with deadline algorithm to improve the ratio of functions which can be accomplished within deadline by timely abandoning the functions that exceed the deadline. However, some high-criticality functions cannot be abandoned in mixed-criticality systems such that the algorithm cannot be applied to such systems. Hu et al. investigated the scheduling of periodic functions on time-triggered FlexRay systems [27], the objective of which is to guarantee that all instances of all functions can meet their respective deadlines. A series of investigations about mixed-criticality functions were proposed by Tamas et al. [28–31]. In [28,29], the authors considered functions to be separated and used a temporal and spatial-partitioning scheme, in which safety-criticality functions are scheduled using static-cycling scheduling and non-safety-criticality functions are scheduled using fixed-priority preemptive scheduling. The major problems for [28,29] are as follows: 1) they only considered two criticality levels based on dual-criticality systems and 2) the communication times of messages for connecting precedence-constrained tasks are ignored. In [30], the authors considered processors interconnected by the TTEthernet-based protocol for mixed-criticality systems and in [31], the authors used a static-cyclic scheduling for multiple-criticality distributed functions on cost-constrained partitioned architectures. The main limitation of the above works is that the platform only supports partitioned scheduling and cannot be applied to global scheduling.

The objective of the above works is to meet the deadlines of all functions. However, as mentioned earlier, the deadlines of all functions cannot be met in heterogeneous distributed embedded systems, particularly in resource-constrained distributed embedded environments. This study will present high performance real-time scheduling approach that reduce the DMR when functions have various criticality levels and deadlines based on global non-preemptive scheduling, and keep satisfactory performance under significantly reducing the DMRs.

3. Modeling

3.1. System architecture

CAN bus is configured with the event-triggered non-preemptive mechanism. When a task is executed completely in one ECU, this task sends messages to all its successor tasks that may be located in the different ECUs of different buses. For example, task n_1 is executed on ECU ECU_1 of CAN_1. It then sends a message $m_{1,2}$ to its successor task n_2 located in ECU_6 of CAN_3 (See Fig. 1). The central gateway is a highly important node that connects CAN clusters and allows messages to be passed from one bus to another. We use $P = \{P_1, P_2, \ldots, P_{|P|}\}$ to represent a set of heterogeneous ECUs; here, $|P|$ represents the size of set P. Note that for any set X, this study uses $|X|$ to denote its size.

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 $F_m = (N, M, C, W, criticality,$ *lowerbound, deadline, makepsan*). F_m represents the mth functions in systems. N represents a 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 $e_{i,j} \in M$ represents the communication message from n_i to n_j. Accordingly, $c_{i,j}$ represents the worst-case transmitting time (WCTT) of $e_{i,j}$. Notice that the WCTT includes the gateway processing time of $e_{i,j}$. $pred(n_i)$ represents the set of the immediate predecessor tasks of n_i. $ind(n_i)$ represents the in-degree of n_i, which indicates the cardinality of $pred(n_i)$. $succ(n_i)$ represents the set of the immediate successor tasks of n_i. $outd(n_i)$ represents the out-degree of n_i, which indicates the cardinality of $succ(n_i)$. For simplicity, a function comprises only one entry task, which has no predecessor task and is denoted as n_{entry} and one exit task, which has no successor task and is denoted as n_{exit}. W is an $|N| \times |P|$ matrix, in which $w_{i,k}$ denotes the WCET 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 [1]).

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 certificated 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 $MS = \{\{F_1, F_2, \ldots, F_{|MS|}\}, criticality, makespan\}$ where *criticality* indicates the current criticality level of the system. In distinguishing the ambiguities, we use *MS.criticality* to express the *criticality* of *MS*, and use F_m*.criticality* to express the *criticality* of F_m. Other attributes use the same expression. *MS.criticality* can be changed to high-criticality levels and back to low-criticality levels. A change in *MS.criticality* indicates a switch in system mode. F_m can only be executed on the modes in which F_m*.criticality* is higher than or equal to *MS.criticality*. *MS.makespan* represents the overall makespan of *MS* and reflects system performance. *MS.makespan* is also generated with the proposed algorithm.

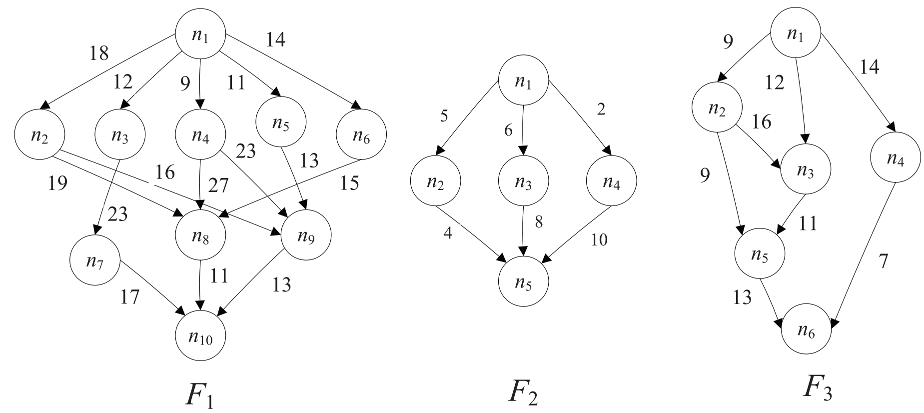

Fig. 2. Motivating example of mixed-criticality systems containing three distributed mixed-criticality functions ($F_1.criticality = S_3$, $F_2.criticality = S_2$, and $F_3.criticality = S_0$).

This study considers static scheduling and 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., WCTT 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, F_1, F_2, and F_3 , and with $F_1.criticality = S_3$, $F_2.criticality = S_2$, and $F_3.criticality = S_0$. The shapes of F_2 and F_3 functions are similar to the examples of [33], whereas F_1 is a relative complex function. Table 1 shows the WCETs of tasks for F_1, F_2, and F_3 in Fig. 2. The example shows ten tasks for F_1, five tasks for F_2, and six tasks for F_3. 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 $F_1.n_1$ and task $F_1.n_2$ represents the WCTT of $F_1.m_{1,2}$ if $F_1.n_1$ and $F_1.n_2$ are not assigned in the same ECU. The weight 14 of $F_1.n_1$ and p_1 in Table 1 represents the WCET and is denoted as $F_1.w_{1,1} = 14$. Section 4.1 explains the meaning of $rank_u$, *lowerbound*, and *deadline* of Table 1.

Table 1
WCETs for tasks of all functions in Fig. 2.

WCETs for tasks of F_1

Criticality	S_3									
ECU	Task									
	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
p_1	14	13	11	13	12	13	7	5	18	21
p_2	16	19	13	8	13	16	15	11	12	7
p_3	19	18	19	17	10	9	11	14	20	16
$rank_u$	109	78	81	81	70	64	43	36	45	15
Lower-bound	80									
Deadline	90									

WCETs for tasks of F_2

Criticality	S_2				
ECU	Task				
	n_1	n_2	n_3	n_4	n_5
p_1	4	9	18	21	7
p_2	5	10	17	15	6
p_3	6	11	16	19	5
$rank_u$	42	20	31	35	6
Lower-bound	36				
Deadline	46				

WCETs for tasks of F_3

Criticality	S_0					
ECU	Task					
	n_1	n_2	n_3	n_4	n_5	n_6
p_1	8	14	9	18	18	5
p_2	11	13	12	15	16	10
p_3	19	8	16	14	20	7
$rank_u$	110	91	63	31	39	8
Lower-bound	54					
Deadline	64					

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 ($rank_u$) 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 $rank_u$. Table 1 shows the upward rank values of all the tasks (Fig. 2), which are obtained with Eq. (1):

$$rank_u(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)} \{c_{i,j} + rank_u(n_j)\}. \quad (1)$$

Second, the attributes $EST(n_j, p_k)$ and $EFT(n_j, p_k)$ represent the earliest start time (EST) and the EFT, respectively, of task n_j on processor p_k. $EFT(n_j, p_k)$ 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:

$$\begin{cases} EST(n_{\mathrm{entry}}, p_k) = 0; \\ EST(n_j, p_k) = max\left(avail[p_k], \max_{n_i \in pred(n_j)} \{AFT(n_i) + c'_{i,j}\}\right); \end{cases} \tag{2}$$

and

$$EFT(n_j, p_k) = EST(n_j, p_k) + w_{j,k}. \tag{3}$$

$avail[p_k]$ is 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

$$F_m.lowerbound = AFT(F_m.n_{\mathrm{exit}}), \tag{4}$$

where $F_m.n_{\mathrm{exit}}$ 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 $rank_{\mathrm{u}}(n_i)$.
(2) In the common ready queue (*common_ready_queue*) of systems for storing ready tasks (selecting one ready task with maximum $rank_{\mathrm{u}}$ from each function), tasks are also ordered according to decreasing $rank_{\mathrm{u}}(n_i)$.
(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 $rank_{\mathrm{u}}(n_i)$.

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

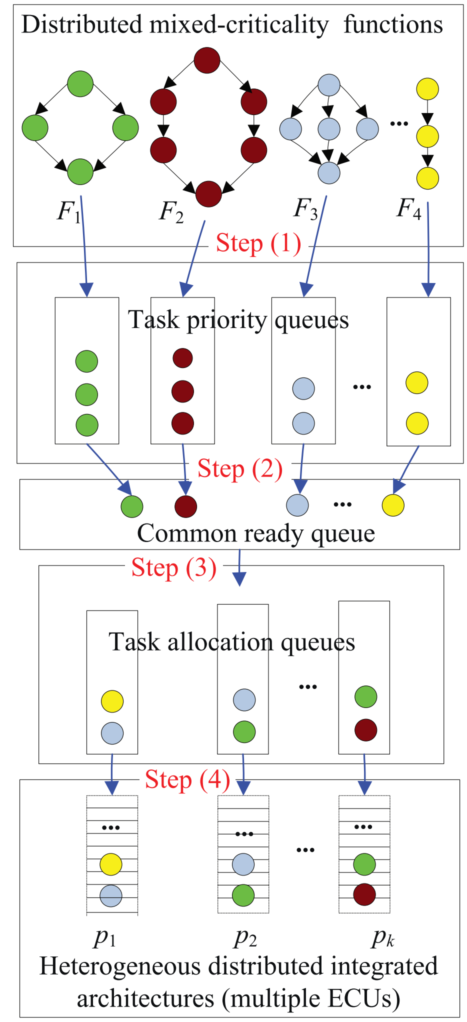

Fig. 3. Scheduling framework of multiple distributed mixed-criticality functions.

Step (3) Task allocation with fairness policy: Select a task with the highest $rank_{\mathrm{u}}(n_i)$ 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(|*MS*|) time. All the tasks of a function can be scheduled in $O(N_{\max})$ time ($N_{\max} = \max(|F_1.N|, |F_2.N|, ..., |F_{|MS|}.N|)$). 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 $O(|MS| \times N_{\max}^2 \times |P|)$.

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

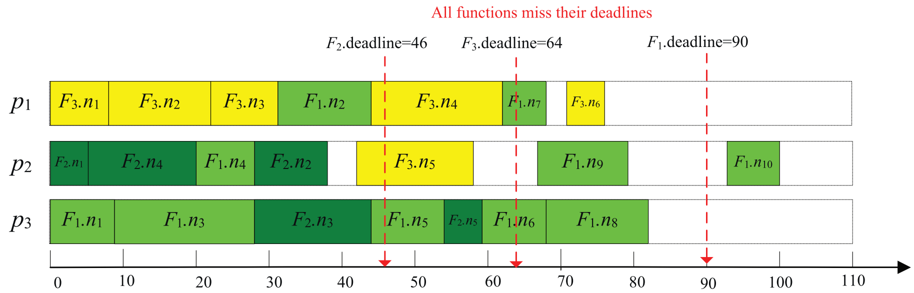

Fig. 4. Scheduling result of the F_MHEFT algorithm for the motivating example.

Algorithm 1 F_MHEFT Algorithm

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

Calculate $rank_u$ for all tasks of all functions in MS, and put these tasks into corresponding $task\_priority\_queue(F_m)$;
while (there are tasks to be allocated) **do**
  for $(m = 1; m <= |MS|; m++)$ **do**
    $n_i = task\_priority\_queue(F_m).out()$;
    $common\_ready\_queue.put(n_i)$; //select one task from each task priority queue, and put it into the common ready queue;
  end for
  while $(!common\_ready\_queue.empty())$ **do**
    $n_i = common\_ready\_queue.out()$; //select one task from the common ready queue to be allocated
    assign n_i to $task\_allocation\_queue(p_k)$ the minimum EFT using the insertion-based scheduling policy;
  end while
end while

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

Step	Task allocation
1	$F_3.n_1$, $F_1.n_1$, $F_2.n_1$
2	$F_3.n_2$, $F_1.n_3$, $F_2.n_4$
3	$F_1.n_4$, $F_3.n_3$, $F_2.n_3$
4	$F_1.n_2$, $F_3.n_5$, $F_2.n_2$
5	$F_1.n_5$, $F_3.n_4$, $F_2.n_5$
6	$F_1.n_6$, $F_3.n_6$
7	$F_1.n_9$
8	$F_1.n_7$
9	$F_1.n_8$
10	$F_1.n_{10}$

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

$$rank_r(F_m.n_i) = \frac{1}{PRT(F_m)} \times \frac{1}{CPL(F_m)}, \quad (5)$$

where $PRT(F_m)$ and $CPL(F_m)$ represent the percentage of remaining task (PRT) 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,

$$F_m.deadlinespan = F_m.deadline - F_m.lowerbound. \quad (6)$$

$F_m.deadline$ 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,

$$deadline(F_m.n_i) = lowerbound(F_m.n_i) + F_m.deadlinespan, \quad (7)$$

where $lowerbound(F_m.n_i) = AFT(F_m.n_i)$ 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). Table 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 mutliple 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($|MS| \times N_{\max}^2 \times |P|$), 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 complected. Finally, the remaining tasks of low-criticality functions are scheduled.

Table 3
Lower-bounds and deadlines of tasks in the F_1 function.

Tasks	$F_1.n_1$	$F_1.n_2$	$F_1.n_3$	$F_1.n_4$	$F_1.n_5$	$F_1.n_6$	$F_1.n_7$	$F_1.n_8$	$F_1.n_9$	$F_1.n_{10}$
Lower-bound	9	40	28	26	38	42	49	62	68	80
Deadline	19	50	38	36	48	52	59	72	78	90

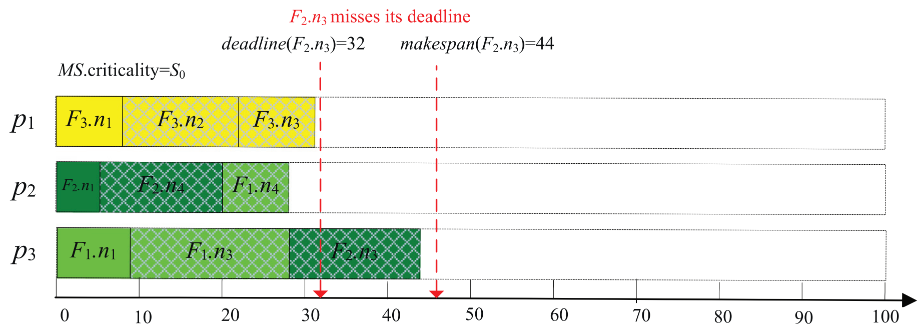

Fig. 5. $MS.criticality = S_0$; the tasks of functions F_1, F_2, and F_3 are scheduled with the fairness policy until $makespan(F_2.n_3) > deadline(F_2.n_3)$.

Algorithm 2 D_MHEFT Algorithm

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

Calculate $rank_u$ for all tasks of all functions in MS, and put these tasks into corresponding $task\_priority\_queue(F_m)$;
$MS.criticality = S_0$;
while (there are tasks to be allocated) **do**
  for $(m = 1; m <= |MS|; m++)$ **do**
    if $(F_m.criticality < MS.criticality)$ **then**
      continue;
    end if
    $n_i = task\_priority\_queue(F_m).out()$;
    $common\_ready\_queue.put(n_i)$; //select one task from each task priority queue, and put it into the common ready queue;
  end for
  while $(!common\_ready\_queue.empty())$ **do**
    $F_m.n_i = common\_ready\_queue.out()$; //select one task from the common ready queue to be allocated;
    Assign $F_m.n_i$ to $task\_allocation\_queue(p_k)$ with the minimum EFT using the insertion-based scheduling policy;
    if $(makespan(F_m.n_i) > deadline(F_m.n_i)$ && $F_m.criticality > MS.criticality)$ **then**
      Cancel the allocation of tasks in this round except for the tasks in scheduled completed functions;
      Put the cancelled tasks back to individual task priority queues;
      Clear the tasks in the common ready queue and put them back to individual task priority queues;
      Change the criticality of the system to the criticality of F_m, namely, $MS.criticality = F_m.criticality$.
    end if
    if (F_m is the function causing the criticality of the system to be changed up and F_m is scheduled completed) **then**
      Clear the tasks in the common ready queue and put them back to individual task priority queues.
      Change the criticality of the system to S_0, namely, $MS.criticality = S_0$.
    end if
  end while
end while

Table 4
Lower-bounds and deadlines of tasks in the F_2 function.

Task	$F_2.n_1$	$F_2.n_2$	$F_2.n_3$	$F_2.n_4$	$F_2.n_5$
Lower-bound	4	20	22	21	36
Deadline	14	30	32	31	46

Table 5
Lower-bounds and deadlines of tasks in the F_3 function.

Task	$F_3.n_1$	$F_3.n_2$	$F_3.n_3$	$F_3.n_4$	$F_3.n_5$	$F_3.n_6$
Lower-bound	8	22	31	36	49	54
Deadline	18	32	41	46	59	64

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

In Fig. 5 with $MS.criticality = S_0$, the tasks ($F_3.n_1$, $F_1.n_1$, $F_2.n_1$, $F_3.n_2$, $F_1.n_3$, $F_2.n_4$, $F_1.n_4$, $F_3.n_3$, and $F_2.n_3$) of F_1, F_2, and F_3 functions are scheduled with the fairness policy until $makespan(F_2.n_3) > deadline(F_2.n_3)$. Thereafter, the allocated tasks (i.e., $F_3.n_2$, $F_1.n_3$, $F_2.n_4$, $F_1.n_4$, $F_3.n_3$, and $F_2.n_3$) (denoted as shadowgraphs) in the current and previous rounds is 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 $MS.criticality = S_2$ because $F_2.criticality = S_2$. Thereafter, the tasks (i.e., $F_1.n_3$, $F_2.n_4$, $F_1.n_4$, $F_2.n_3$, $F_1.n_2$, $F_2.n_2$, $F_1.n_5$, and $F_2.n_5$) of F_1 and F_2 functions are scheduled with the fairness policy until all the scheduled tasks of F_2 are allocated. In this mode, $F_2.makespan = 44$, which is less than $F_2.deadline = 46$. Hence, F_2 meets its deadline and is safe.

Fig. 7 shows that the system criticality is changed down to $MS.criticality = S_0$. Thereafter, the tasks (i.e., $F_3.n_2$ and $F_1.n_6$) of F_1 and F_3 functions are scheduled with the fairness policy until $makespan(F_1.n_6) > deadline(F_1.n_6)$. Thereafter, the allocated tasks (e.g., $F_1.n_5$, $F_3.n_2$, and $F_1.n_6$) (denoted as shadowgraphs) in the current and previous rounds is cancelled. Note that F_2 has been completed, and its tasks cannot be cancelled.

Fig. 8 shows that the criticality of the system is changed up to $MS.criticality = S_3$ because $F_1.criticality = S_3$. Thereafter, the tasks (i.e., $F_1.n_5$, $F_1.n_6$, $F_1.n_9$, $F_1.n_7$, $F_1.n_8$, and $F_1.n_{10}$) of F_1 function are scheduled with the fairness policy until all the scheduled tasks of F_1 are allocated. In this mode, $F_1.makespan = 89$, which is less than $F_1.deadline = 90$. Hence, F_1 meets its deadline and is safe.

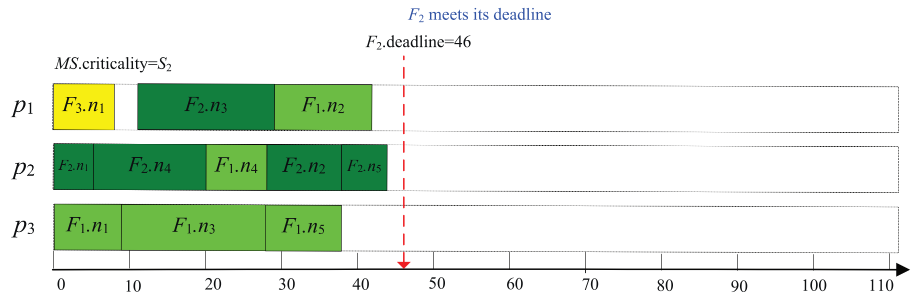

Fig. 6. The criticality level of the system is changed up to $MS.criticality = S_2$. The tasks of the F_1 and F_2 functions are scheduled with the fairness policy until all the scheduled tasks of F_2 are allocated.

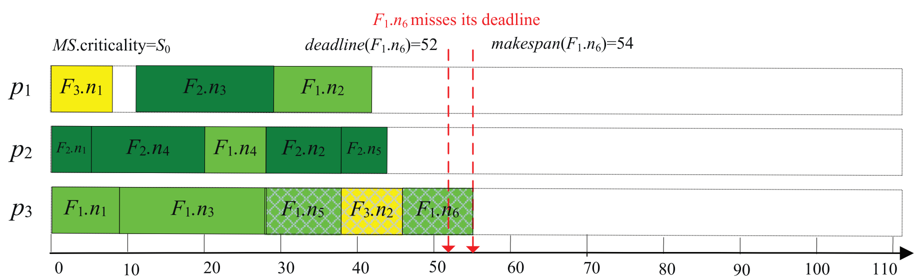

Fig. 7. System criticality is changed down to $MS.criticality = S_0$. The tasks of the F_1 and F_3 functions are scheduled with the fairness policy until $makespan(F_1.n_6) > deadline(F_1.n_6)$.

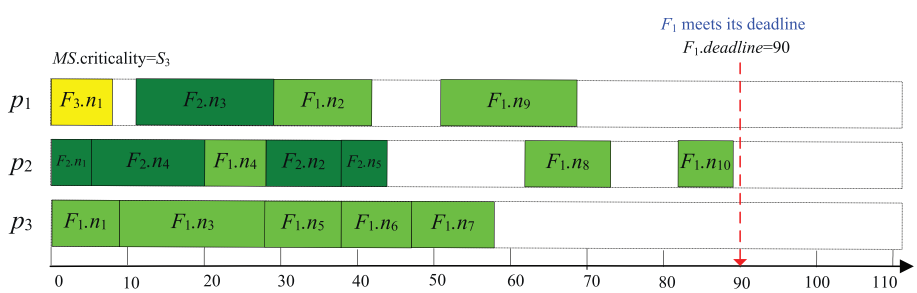

Fig. 8. The criticality level of the system is changed up to $MS.criticality = S_3$. The tasks of the F_1 function are scheduled until all the scheduled tasks of F_1 are allocated.

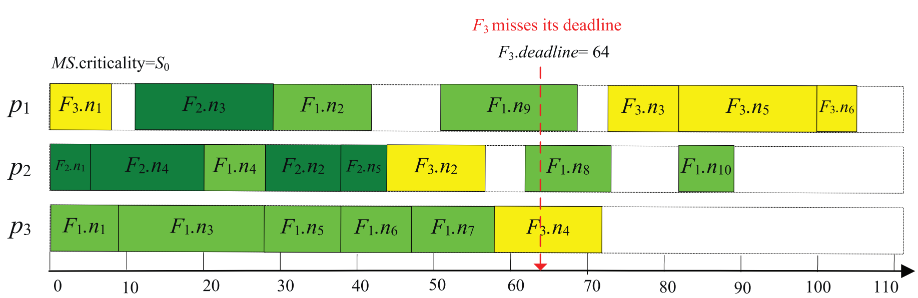

Fig. 9. System criticality is changed down to $MS.criticality = S_0$. The tasks of the F_3 function are scheduled with the fairness policy.

Fig. 9 shows that the system criticality is changed down to $MS.criticality = S_0$. Thereafter, the tasks (i.e., $F_3.n_2$, $F_3.n_3$, $F_3.n_5$, $F_3.n_4$, and $F_3.n_6$) 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, $F_3.makespan = 105$, which is larger than $F_3.deadline = 64$. 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.

Table 6
DMRs for varying numbers of functions using FDWS, F_MHEFT, and D_MHEFT.

Algorithms	FDWS				F_MHEFT				D_MHEFT			
Criticality levels	S_0	S_1	S_2	S_3	S_0	S_1	S_2	S_3	S_0	S_1	S_2	S_3
$\|MS\| = 40$	0.8	0.4	0.6	0.8	0.7	0.7	0.6	0.5	0.7	0.6	0.7	0.2
$\|MS\| = 80$	0.6	0.65	0.7	0.55	0.75	0.65	0.85	0.6	0.85	0.5	0.5	0.2
$\|MS\| = 160$	1.0	0.975	0.95	0.95	0.975	0.975	1.0	1.0	1.0	0.925	0.775	0.375
$\|MS\| = 320$	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.55
$\|MS\| = 640$	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9875

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

$$MS.makespan = \max_{F_m \in MS}\{F_m.makespan\}. \tag{8}$$

DMR is calculated using

$$DMR(S_x) = \frac{|MS^{\text{miss}}(S_x)|}{|MS(S_x)|}, \tag{9}$$

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

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 $F_m.deadlinespan = F_m.lowerbound/40$. 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. Specificality, 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 $|MS| = 160$ and $|MS| = 320$, 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 7
Overall makespans (μs) for varying numbers of functions using FDWS, F_MHEFT, and D_MHEFT.

Algorithms	FDWS	F_MHEFT	D_MHEFT
$\|MS\| = 20$	7286	7450	7166
$\|MS\| = 40$	7256	7090	7179
$\|MS\| = 60$	8306	7879	9721
$\|MS\| = 80$	10,058	9599	13,717
$\|MS\| = 100$	15,265	14,936	22,269

Table 7 shows the overall makespans for varying numbers of functions using the three algorithms. Except for the small-scale function sets ($|MS| = 20$), F_MHEFT outperform FDWS in middle and large-scale function sets. Similarly, except for the case of small-scale ($|MS| = 20$ and $|MS| = 40$), 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.

Experiment 2. Given that the system cannot meet the deadlines of all the functions with S_3 in Experiment 1, the number of such functions should be reduced. In this experiment, the total number of functions is fixed with 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 $F_m.deadlinespan = F_m.lowerbound/40$. 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 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_3.

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 μs. D_MHEFT generate longer makespan than F_MHEFT, and the differences are relative values (3808- 5355μs).

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 S_0 and S_3 using F_MHEFT and D_MHEFT.

Algorithms	F_MHEFT				D_MHEFT			
Criticality levels	S_0	S_1	S_2	S_3	S_0	S_1	S_2	S_3
$\|MS(S_0)\| = 80, \|MS(S_1)\| = 80, \|MS(S_2)\| = 80, \|MS(S_3)\| = 80$	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.6625
$\|MS(S_0)\| = 100, \|MS(S_1)\| = 80, \|MS(S_2)\| = 80, \|MS(S_3)\| = 60$	1.0	1.0	1.0	1.0	1.0	1.0	0.975	0.5666
$\|MS(S_0)\| = 120, \|MS(S_1)\| = 80, \|MS(S_2)\| = 80, \|MS(S_3)\| = 40$	1.0	1.0	1.0	1.0	1.0	1.0	0.9375	0.35
$\|MS(S_0)\| = 140, \|MS(S_1)\| = 80, \|MS(S_2)\| = 80, \|MS(S_3)\| = 20$	1.0	1.0	1.0	1.0	1.0	1.0	0.8	0.1
$\|MS(S_0)\| = 150, \|MS(S_1)\| = 80, \|MS(S_2)\| = 80, \|MS(S_3)\| = 10$	1.0	1.0	1.0	1.0	1.0	1.0	0.6875	0.0

Table 9
Overall makespans (μs) for varying numbers of functions with S_0 and S_3 using F_MHEFT and D_MHEFT.

Algorithms	F_MHEFT	D_DMHEF	DIFFERENCES
$\|MS(S_0)\| = 80, \|MS(S_1)\| = 80, \|MS(S_2) = \|80, \|MS(S_3)\| = 80$	9686	14,089	4403
$\|MS(S_0)\| = 100, \|MS(S_1)\| = 80, \|MS(S_2) = \|80, \|MS(S_3)\| = 60$	9686	15,041	5355
$\|MS(S_0)\| = 120, \|MS(S_1)\| = 80, \|MS(S_2) = \|80, \|MS(S_3)\| = 40$	9686	14,593	4907
$\|MS(S_0)\| = 140, \|MS(S_1)\| = 80, \|MS(S_2) = \|80, \|MS(S_3)\| = 20$	9686	14,586	4900
$\|MS(S_0)\| = 150, \|MS(S_1)\| = 80, \|MS(S_2) = \|80, \|MS(S_3)\| = 10$	9686	13,494	3808

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

Algorithm	F_MHEFT				D_MHEFT			
Criticality	S_0	S_1	S_2	S_3	S_0	S_1	S_2	S_3
$F_m.deadlinespan = F_m.lowerbound/40$	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.7125
$F_m.deadlinespan = F_m.lowerbound/30$	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.6125
$F_m.deadlinespan = F_m.lowerbound/20$	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.55
$F_m.deadlinespan = F_m.lowerbound/10$	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.3
$F_m.deadlinespan = F_m.lowerbound/5$	1.0	1.0	1.0	1.0	1.0	1.0	0.8625	0.1125

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

Algorithm	F_DMHEFT	D_DMHEFT	DIFFERENCES
$F_m.deadlinespan = F_m.lowerbound/40$	9805	13,966	4161
$F_m.deadlinespan = F_m.lowerbound/30$	9805	13,822	4017
$F_m.deadlinespan = F_m.lowerbound/20$	9805	13,980	4175
$F_m.deadlinespan = F_m.lowerbound/10$	9805	14,089	4284
$F_m.deadlinespan = F_m.lowerbound/5$	9805	14,089	4284

each function F_m is gradually changed from $F_m.lowerbound/40$ to $F_m.lowerbound/5$. Two algorithms (i.e., F_MHEFT and D_MHEFT) are used for the experiment and then compared for verification.

Table 10 shows the DMRs for varying deadline-spans using the F_MHEFT and D_MHEFT algorithms. The DMR values using F_MHEFT are always 1.0 in all the cases. However, the DMRs of functions with S_3 using D_MHEFT are reduced step by step. When the deadline-span is reduced to $F_m.deadlinespan = F_m.lowerbound/5$, the DMR of functions with S_3 is merely 0.1125; meanwhile, the DMR of functions with S_2 is also reduced to 0.8625. By expanding the deadline-span, we implement the objective that the system meet the deadlines of more functions with S_3.

Table 11 shows the overall makespans for varying deadline-spans using the F_MHEFT and D_MHEFT algorithms. The overall makespan using F_MHEFT is always fixed as 9805 μs. We can see that D_MHEFT generate longer makespan than F_MHEFT. The differences (4017-4284μs) are also relatively stable.

This experiment also indicates that D_MHEFT can significantly reduce the DMR and keep satisfactory performance. Considering that changing the deadline-span of a function is actually changing its deadline, D_MHEFT can provide certain design guideline to deadline certification in actual system design.

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 up 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

[1] G. Xie, R. Li, K. Li, Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems, J. Parallel Distr. Comput. 83 (2015) 1–12.

[2] D. Goswami, R. Schneider, A. Masrur, M. Lukasiewycz, S. Chakraborty, H. Voit, A. Annaswamy, Challenges in automotive cyber-physical systems design, in: Embedded Computer Systems (SAMOS), 2012 International Conference on, IEEE, 2012, pp. 346–354.

[3] M.D. Natale, A. Sangiovanni-Vincentelli, Moving from federated to integrated architectures in automotive: the role of standards, methods and tools, Proc. IEEE 98 (4) (2010) 603–620.

[4] S. Baruah, H. Li, L. Stougie, Towards the design of certifiable mixed-criticality systems, in: Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE, IEEE, 2010, pp. 13–22.

[5] I. ISO, 26262–road vehicles-functional safety, ISO Standard (2011).

[6] M. Zeller, C. Prehofer, G. Weiss, D. Eilers, R. Knorr, Towards self-adaptation in real-time, networked systems: efficient solving of system constraints for automotive embedded systems, in: Self-Adaptive and Self-Organizing Systems (SASO), 2011 Fifth IEEE International Conference on, IEEE, 2011, pp. 79–88.

[7] P. Heinrich, C. Prehofer, Network-wide energy optimization for adaptive embedded systems, ACM SIGBED Rev. 10 (1) (2013) 33–36.

[8] H. Topcuoglu, S. Hariri, M.-y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, Parallel Distr. Syst IEEE Trans. 13 (3) (2002) 260–274.

[9] H. Zeng, M. Di Natale, P. Giusto, A. Sangiovanni-Vincentelli, Stochastic analysis of can-based real-time automotive systems, Indus. Inf. IEEE Trans. 5 (4) (2009) 388–401.

[10] H. Zeng, M.D. Natale, A. Ghosal, A. Sangiovanni-Vincentelli, Schedule optimization of time-triggered systems communicating over the flexray static segment, Indus. Inf. IEEE Trans. 7 (1) (2011) 1–17.

[11] S. Manolache, P. Eles, Z. Peng, Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints, ACM Trans. Embedded Comput. Syst. (TECS) 7 (2) (2008) 19.

[12] M.A. Khan, Scheduling for heterogeneous systems using constrained critical paths, Parallel Comput. 38 (4) (2012) 175–193.

[13] H. Arabnejad, J.G. Barbosa, List scheduling algorithm for heterogeneous systems by an optimistic cost table, Parallel Distr. Sys. IEEE Trans. 25 (3) (2014) 682–694.

[14] U. Hönig, W. Schiffmann, A meta-algorithm for scheduling multiple dags in homogeneous system environments, in: Proceedings of the Eighteenth IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS06), 2006.

[15] H. Zhao, R. Sakellariou, Scheduling multiple dags onto heterogeneous systems, in: Parallel and Distributed Processing Symposium, 2006 (IPDPS 2006) 20th International, IEEE, 2006, pp. 14–pp.

[16] C.-C. Hsu, K.-C. Huang, F.-J. Wang, Online scheduling of workflow applications in grid environments, Future Gener. Comput. Syst. 27 (6) (2011) 860–870.

[17] H. Arabnejad, J. Barbosa, Fairness resource sharing for dynamic workflow scheduling on heterogeneous systems, in: Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on, IEEE, 2012, pp. 633–639.

[18] S. Vestal, Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance, in: Real-Time Systems Symposium, 2007 (RTSS 2007) 28th IEEE International, IEEE, 2007, pp. 239–243.

[19] J. Anderson, S. Baruah, B.B. Brandenburg, Multicore operating-system support for mixed criticality, in: Proceedings of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification, Citeseer, 2009.

[20] M.S. Mollison, J.P. Erickson, J.H. Anderson, S.K. Baruah, J.A. Scoredos, Mixed–criticality real-time scheduling for multicore systems, in: Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on, IEEE, 2010, pp. 1864–1871.

[21] N. Guan, P. Ekberg, M. Stigge, W. Yi, Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems, in: Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd, IEEE, 2011, pp. 13–23.

[22] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, L. Stougie, The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems, in: Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on, IEEE, 2012, pp. 145–154.

[23] S. Abrishami, M. Naghibzadeh, D.H. Epema, Cost-driven scheduling of grid workflows using partial critical paths, Parallel Distr. Syst. IEEE Trans. 23 (8) (2012) 1400–1414.

[24] S. Abrishami, M. Naghibzadeh, D.H. Epema, Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds, Future Gener. Comput. Syst. 29 (1) (2013) 158–169.

[25] M. Mao, M. Humphrey, Auto-scaling to minimize cost and meet application deadlines in cloud workflows, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2011, p. 49.

[26] W. Wang, Q. Wu, Y. Tan, F. Wu, Maximize throughput scheduling and cost-fairness optimization for multiple dags with deadline constraint, in: Algorithms and Architectures for Parallel Processing, Springer, Cham, Switzerland, 2015, pp. 621–634.

[27] M. Hu, J. Luo, Y. Wang, B. Veeravalli, Scheduling periodic task graphs for safety-critical time-triggered avionic systems, IEEE Trans. Aerosp. Electron. Syst. 51 (2015).

[28] D. Tămaş-Selicean, P. Pop, Optimization of time-partitions for mixed-criticality real-time distributed embedded systems, in: Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on, IEEE, 2011, pp. 1–10.

[29] D.T. Selicean, P. Pop, Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures, in: Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd, IEEE, 2011, pp. 24–33.

[30] D. Tămaş-Selicean, P. Pop, W. Steiner, Design optimization of ttethernet-based distributed real-time systems, Real-Time Systems 51 (1) (2015) 1–35.

[31] D. Tămaş-Selicean, P. Pop, Design optimization of mixed-criticality real-time embedded systems, ACM Trans. Embedded Comput. Syst. (TECS) 14 (3) (2015) 50.

[32] V. Scheuch, G. Kaiser, M. Korte, P. Grabs, F. Kreft, F. Holzmann, A safe torque vectoring function for an electric vehicle, in: Electric Vehicle Symposium and Exhibition (EVS27), 2013 World, IEEE, 2013, pp. 1–10.

[33] P. Pop, P. Eles, Z. Peng, Analysis and optimisation of heterogeneous real-time embedded systems, IEE Proc. Comput. Digital Tech. 152 (2) (2005) 130–147.

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.