WCRT Analysis and Evaluation for Sporadic Message-Processing Tasks in Multicore Automotive Gateways

Guoqi Xie, *Member, IEEE*, Gang Zeng, *Member, IEEE*, Ryo Kurachi, Hiroaki Takada, *Member, IEEE*, Zhetao Li, *Member, IEEE*, Renfa Li, *Senior Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—We study the worst case response time (WCRT) analysis and evaluation for sporadic message-processing tasks in a multicore automotive gateway of a controller area network (CAN) cluster. We first build a multicore automotive gateway on CAN clusters. Two WCRT analysis methods for message-processing tasks in the multicore gateway are subsequently presented based on global and partitioned scheduling paradigms. We evaluate the WCRT results of two analysis methods with real message sets provided by the automaker, and present the design optimization guide.**

***Index Terms*—Automotive gateway, global scheduling, multicore, partitioned scheduling, worse case response time (WCRT).**

I. Introduction

A. Background

TODAY's automotive electrical and electronic architecture has evolved into integrated architecture [1], [2], in which automotive systems are distributed over a multitude of subsystems (e.g., body subsystems, powertrain systems, and entertainment subsystems) because of the required constraints on size, weight, and power consumption for cost and performance benefit. A typical integrated system is a controller area networks (CANs) cluster where more than four or five CAN buses are integrated by a central gateway [3]–[5]. Large-scale CAN clusters are desirable to satisfy the increased bandwidth requirements of in-vehicle networks. CAN is a half duplex, static priority, and nonpreemptive scheduling communication protocol and is currently the most used in vehicles [6]–[8], and the gateway is an important node that connects and allows CAN clusters to pass messages from one bus to another [9], as it needs to copy thousands of CAN messages from one bus to the other bus(es) to complete many distributed cross-bus functionalities. For example, the engine controller of the powertrain subsystem should take input signals from the body subsystem through the gateway, and the dashboard of the body subsystem displays information from other subsystems [10]. However, the gateway becomes a bottleneck in message transmission because it introduces a large body of message-processing delay. Furthermore, most current commercial gateway products are only configured with a single-core CPU, and most studies also investigated message-processing delay based on a single-core gateway [11]–[13]. To eliminate or reduce the bottleneck in message transmission, the dual-core automotive gateway (e.g., MPC5668G is a dual-core system-on-chip 32-bit microcontroller [14]) has emerged with the development of advanced communication and processor technologies. With the increased distributed cross-bus functionalities causing message transmission delay in automobiles, using more CPU cores in the gateway is a solution to eliminate or reduce the bottleneck. However, adding more cores is not always feasible because the automotive industry is a highly cost-sensitive industry for the mass market. Therefore, we need to know the actual possible message transmission delay in a given multicore gateway in advance before designing the gateway.

Message transmission delay can be obtained by using worst case response time (WCRT). The WCRT of a message-processing task in the gateway is the maximum response time among all possible real response time values when the message-processing task is executed. The response time of the message represents the sum of its delay and executione time WCRT analysis is a typical method to find out the exact WCRT or derive a WCRT upper bound from all possible real response time values. Considering that automotive systems are highly safety-critical systems, the estimation of WCRT is necessary.

Manuscript received October 23, 2017; revised January 8, 2018; accepted February 27, 2018. Date of publication March 5, 2018; date of current version January 18, 2019. This work was supported in part by the National Key Research and Development Plan of China, under Grant 2016YFB0200405, in part by the National Natural Science Foundation of China, under Grant 61702172 and Grant 61672217, in part by the China Post-Doctoral Science Foundation under Grant 2016M592422, in part by the CCF-Venustech Open Research Fund under Grant CCF-VenustechRP2017012, in part by the Fundamental Research Funds for the Central Universities, and in part by the Natural Science Foundation of Hunan Province, China, under Grant 2018JJ3076. This paper was recommended by Associate Editor H. Li. *(Corresponding author: Zhetao Li.)*

G. Xie and R. Li are with the College of Computer Science and Electronic Engineering, Key Laboratory for Embedded and Network Computing of Hunan Province, Hunan University, Hunan 410082, China (e-mail: xgqman@hnu.edu.cn; lirenfa@hnu.edu.cn).

G. Zeng, R. Kurachi, and H. Takada are with the Center for Embedded Computing Systems, Nagoya University, Aichi 4648603, Japan (e-mail: sogo@ertl.jp; kurachi@nces.is.nagoya-u.ac.jp; hiro@ertl.jp).

Z. Li is with the College of Information Engineering, Xiangtan University, Hunan 411105, China (e-mail: liztchina@hotmail.com).

K. Li is with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China, and also with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the author. The total size of the file is 0.035 MB.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2018.2812119

B. Motivation

Experimental measurements are the approaches primarily used to analyze the message-processing delay in the gateway in recent years [11], [13]. The transmission time of one packet for a CAN bus with a speed of 500 Kb/s is approximately 250 μs [10], [11], and the execution time for each message-processing task in the gateway with a Freescale MPC555 microcontroller with two independent CAN 2.0 A/B communication controllers is approximately 40 μs when two buses are connected by a gateway [12]. The gateway task is interfered by at most one high-priority message-processing task because the speed of the gateway processing is faster than that of the message transmission process. As a result, the message can go through the gateway immediately with a negligible processing time. However, for a gateway with large-scale CAN clusters, the message-processing delays obtained by experimental measurements are unsafe, because the message-processing task can be interfered by other high-priority message-processing tasks from multiple buses. In this situation, the interfering delay would exceed the message-processing time.

The priority level-i busy period was used to calculate the WCRTs of static priority tasks. The busy period was also applied to the WCRT analysis of the CAN messages in the in-vehicle network [5], [6] and general tasks with multiple execution units in the multicore processor [15]–[18]. However, if the level-i busy period is applied to event-triggered sporadic tasks, the obtained WCRT could be very pessimistic because it treats the minimum interarrival time as the period to involve WCRT calculation. In fact, message-processing tasks in the gateway are event-triggered sporadic tasks because they are triggered by the messages from CAN buses.

C. Our Contributions

The main contributions are as follows.

1) We build a multicore automotive gateway model on CAN clusters based on the discrete channel gateway module, store-modify-forward switching, strict priority routing, and nonpreemptive scheduling.
2) We present the WCRT analysis methods of message-processing tasks based on partitioned and global scheduling paradigms, respectively.
3) We evaluate the WCRT results of the two analysis methods with experiments. We explain that which scheduling paradigm is remarkably suitable for the multicore automotive gateway and summarize the multicore gateway design guide based on experiment results.

II. Related Work

The related works are organized as the following: 1) gateway processing delay analysis using experimental measurements; 2) WCRT analysis for CAN messages with gateway in the in-vehicle network; and 3) WCRT analysis for general tasks in the multicore processor.

Several studies have investigated the gateway processing of automotive systems. Sommer and Blind [11] proposed a CAN/CAN gateway embedded system that investigated the processing delay measurement and buffer capacity. Schmidt *et al.* [19] studied the gateway processing delay and its performance evaluation in the CAN/FlexRay gateway. In [13], the authors measured the gateway processing delay in a Linux-based CAN/CAN gateway. Other related works also investigated the automotive gateways, such as the FlexRay/CAN gateway [20], the Ethernet/CAN gateway [21], [22], and the central gateway connecting more than two subsystems [23], [24]. The main limitation of the aforementioned studies is that the measured experimental results may dynamically change with time and environment.

WCRT analysis for CAN messages with a gateway, and end-to-end WCRT analysis for CAN messages with a gateway have been researched. The classical WCRT analysis proposed in [6] for a single CAN bus can be applied to gateway-integrated networks, but it can only obtain quite pessimistic results as it overlooks the influence of the gateway. Optimized WCRT is obtained by using the multiple first input first output queues in the gateway with experiments [25]. Azketa *et al.* [10] presented an optimized WCRT analysis method for CAN messages that considers the pipelining of the packets in the gateway. In [5], we examined in detail the various actual arriving orders of gateway messages and proposed an explorative WCRT computation method. Different from [5], this paper aims to analyze the WCRT of message-processing tasks in the gateway rather than the messages in CAN buses.

WCRT analysis for static priority tasks in the multicore processor based on global scheduling has been studied in [15]–[18], where WCRT upper bound is derived considering the carried-in and carried-out jobs with multiple execution units. However, the aforementioned works calculate the WCRT by treating the minimum interarrival time as the period. For event-triggered sporadic tasks, such treatment is very pessimistic. Partitioned scheduling is analogous to the bin packing problem and is known as NP-Hard; several heuristic algorithms, such as the next fit, first fit (FF), FF decreasing (FFD), best fit, and best FF [26].

In summary, existing studies either obtain the message-processing time in the gateway by the experimental measurements, or analyze the WCRTs of CAN messages in the in-vehicle network, or analyze the WCRTs of periodic general tasks in the multicore processor, but do not specifically analyze the WCRTs for the sporadic message-processing tasks in the multicore gateway, which will be systematically studied in the following.

III. Gateway Models

A. CAN Cluster Architecture

The CAN gateway architectures for automotive applications can be classified into three types, namely, the discrete channel gateway, the complex channel gateway, and the modular gateway [27]. Considering that the architecture is a CAN cluster, the discrete channel gateway is used in the present study because of its flexibility with regard to the number of CAN channels. The left side of Fig. 1 illustrates the architecture of the discrete channel gateway, which comprises three components (i.e., the CPU, the CPU peripheral bus, and several single-channel CAN modules). Message-processing is described as follows: the CPU must read all necessary control information and receive messages over the peripheral bus from one CAN module and subsequently writes the same data over the peripheral bus to some other CAN module(s).

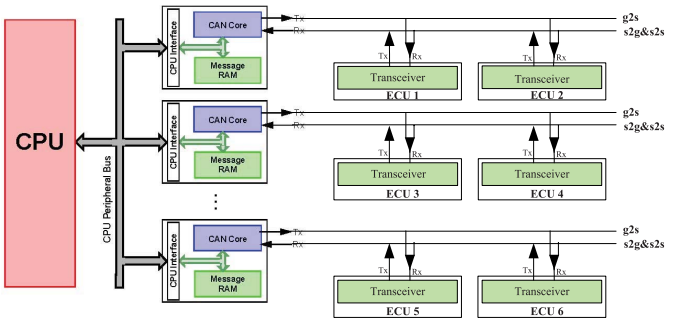


Fig. 1. CAN cluster architecture with discrete channel CAN gateway and subsystems.

We consider that the gateway-integrated CAN cluster comprises multiple CAN subsystems with the same or different bandwidths and that each subsystem contains several electronic control units (ECUs), as shown in the right side of Fig. 1. Note that two CAN buses (each bus is a two-wire twisted pair) exist in each subsystem in our CAN cluster architecture: One bus is in charge of receiving messages from the gateway (denoted with $g2s$), and another bus is responsible for sending messages to the gateway and communicating between ECUs in the same subsystem (denoted with $s2g\&s2s$). This architecture can be supported by several dual high-speed CAN transceivers (e.g., TJA1048, NCV7441, and AMIS-42700). We utilize this architecture because the messages sent from the gateway do not interfere with the messages released in the subsystem. Thus, each originally designed subsystem is still valid, and such architecture can simplify the WCRT analysis and then to guarantee real-time properties of message-processing tasks. Each CAN channel in the gateway is configured with two priority queues: one is an input queue (Rx-buffer) and the other is an output queue (Tx-buffer). Each ECU contains a message set that should be released and is also configured with two priority queues (i.e., Rx-buffer and Tx-buffer). Our objective is to obtain a possible maximum response time; thus, we assume that all the buffer sizes of queues are unlimited.

B. Gateway Architecture

Fig. 2 shows the gateway architecture of this paper, which contains eight main components.

1) *Filters:* The filters determine whether the incoming messages from the source subsystem should be forwarded to the destination subsystems with the help of the look-up table.
2) *Receiving Queue (Rx-Buffer):* Each CAN controller contains one such priority queue. It receives the incoming messages from source subsystems.
3) *Receiver:* The receiver is in charge of receiving the highest priority messages from each receiving queue.
4) *Router:* Two basic switching policies exist in the gateway, namely, cut-through switching and store-and-forward switching [11]. We use the same policy as in [11], which employs the store-modify-forward switching policy for automotive gateways. This policy works as follows: when the receiver receives a message from a source subsystem, the router modifies it and transmits it to the destination subsystem(s) with the help of the scheduler. Moreover, two routing policies exist in the gateway, namely, the strict priority and the cyclic polling routing policies (also known as round-Robin routing policy [28]). We use the strict priority policy, which means that the message-processing tasks inherit the priorities of corresponding incoming messages. Consequently, the timing constraints of high-priority messages are still valid in the gateway. The work in [11] uses the round-Robin policy, but it destroys the importance of high-priority messages.
5) *Scheduler:* The scheduler routes messages from receiving queues to the gateway for processing. Scheduling policies can be either preemptive (e.g., OSEKTime) or nonpreemptive (e.g., eCos) [4], and both of them can be supported by AUTOSAR [29]. This paper uses the nonpreemptive scheduling to keep consistent with the message scheduling of the CAN buses.
6) *Message-Processing Tasks:* The message-processing task is executed to copy the message from the scheduler to the dispatcher. All message-processing tasks have equal execution time from execution to termination, as shown in [24], because the execution is a message copy process. Notably, other nonmessage-processing tasks (e.g., I/O and redundancy checks) may run on the gateway [27]. In this paper, these tasks take lower priorities than any message-processing task such that only blocking is their effect on message-processing tasks and they show no interference on the static priority nonpreemptive scheduling. We assume that a homogeneous multicore CPU is utilized in the gateway to achieve high performance. Consequently, concurrent message transfer requests from different CAN channels can be processed simultaneously by multiple cores.
7) *Dispatcher:* The dispatcher routes the copied messages to corresponding transmission queues.
8) *Transmission Queue (Tx-Buffer):* Each CAN controller contains one such priority queue. It sends the messages to the destination subsystems.

In summary, we have specified the message-processing policy of the gateway as store-modify-forward switching, strict priority routing, nonpreemptive scheduling, and priority queues.

C. CAN Message Model

Given that a message-processing task is triggered by a corresponding incoming message, the CAN message model should be defined before determining the message-processing task model. Let $U = \{u_1, u_2, \ldots, u_{|U|}\}$ represent a multicore gateway CPU with $|U|$ cores. Remarkably, for any set X, this paper uses $|X|$ to denote its size. Let $S = \{s_1, s_2, \ldots, s_{|S|}\}$ represent a CAN cluster with $|S|$ subsystems. Let $M = \{m_1, m_2, \ldots, m_{|M|}\}$ represent a message set. Let m_i=($P_{m_i}, T_{m_i}, C_{m_i}, D_{m_i}, s_{\text{src},i}, s_{\text{dest},i}$) represent the ith message of the M, i is the distinct identifier of a message, and P_{m_i} represents the priority of m_i. When two messages m_i and m_j satisfy $i < j$, then m_i has a higher priority than m_j. T_{m_i} is considered as the period or minimum interarrival time of m_i, and C_{m_i} and D_{m_i} are the maximum transmission time and deadline, respectively, of m_i. Finally, $s_{\text{src},i}$ and $s_{\text{dest},i}$ represent the source- and destination-end subsystems, respectively, of m_i. No queuing jitter and offset exist for each message, which

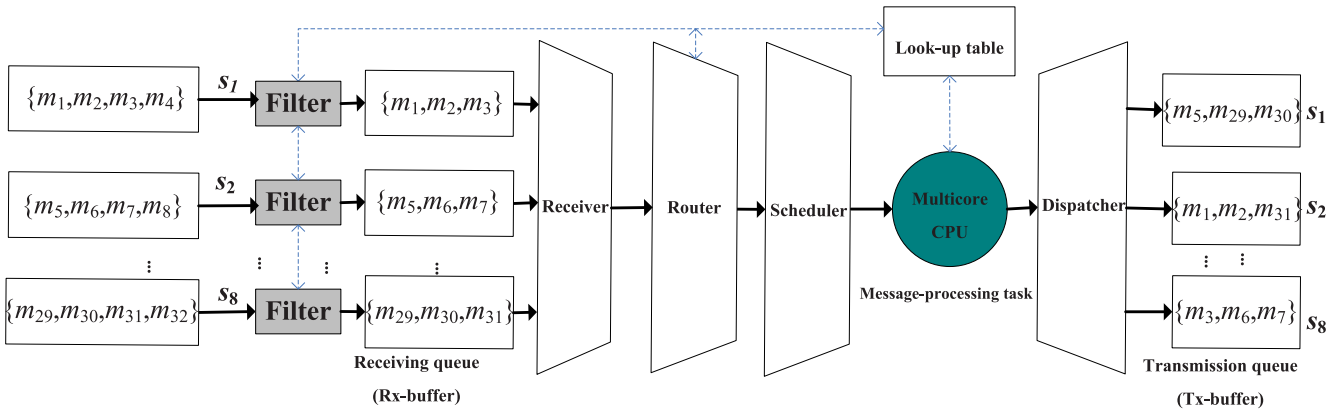


Fig. 2. Multicore automotive gateway architecture.

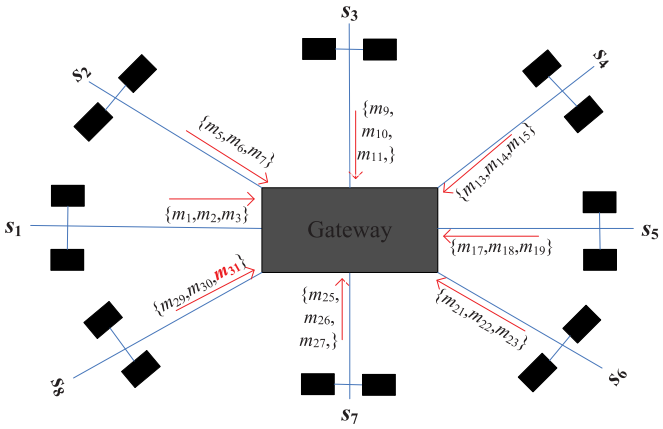


Fig. 3. Example of the CAN cluster with eight subsystems.

TABLE I
MOTIVATING EXAMPLE OF MESSAGE SET IN FIG. 3

m_i	$s_{\text{src},i}$	$s_{\text{dest},i}$	P_{m_i}	T_{m_i}	C_{m_i}	D_{m_i}	m_i	$s_{\text{src},i}$	$s_{\text{dest},i}$	P_{m_i}	T_{m_i}	C_{m_i}	D_{m_i}
m_1	s_1	s_2	1	24	4	24	m_{17}	s_5	s_6	17	10	5	10
m_2	s_1	s_2	2	12	4	12	m_{18}	s_5	s_6	18	30	5	30
m_3	s_1	s_8	3	24	3	24	m_{19}	s_5	s_6	19	30	4	30
m_4	s_1	s_1	4	24	5	24	m_{20}	s_5	s_5	20	30	5	30
m_5	s_2	s_1	5	12	4	12	m_{21}	s_6	s_7	21	10	4	10
m_6	s_2	s_8	6	24	4	24	m_{22}	s_6	s_7	22	30	3	30
m_7	s_2	s_8	7	24	3	24	m_{23}	s_6	s_7	23	15	5	15
m_8	s_2	s_2	8	24	5	24	m_{24}	s_6	s_6	24	30	5	30
m_9	s_3	s_4	9	24	4	24	m_{25}	s_7	s_4	25	10	4	10
m_{10}	s_3	s_4	10	12	4	12	m_{26}	s_7	s_4	26	30	5	30
m_{11}	s_3	s_4	11	12	3	12	m_{27}	s_7	s_4	27	30	4	30
m_{12}	s_3	s_3	12	24	4	24	m_{28}	s_7	s_7	28	30	5	30
m_{13}	s_4	s_5	13	8	4	8	m_{29}	s_8	s_1	29	30	5	30
m_{14}	s_4	s_5	14	24	4	24	m_{30}	s_8	s_1	30	30	3	30
m_{15}	s_4	s_5	15	24	3	24	m_{31}	s_8	s_2	31	15	4	15
m_{16}	s_4	s_4	16	24	4	24	m_{32}	s_8	s_8	32	30	5	30

means that all messages in the same ECU can be released simultaneously [5]. Fig. 3 shows an example of the CAN cluster with eight subsystems ($|S| = 8$). Each subsystem contains two CAN buses. Table I lists the corresponding message set ($|M| = 32$) of Fig. 3. This message set is considered as a motivating example in this paper.

Let the message set that released in the subsystem s_y be $M(s_y)$, which contains two types of messages: 1) the messages transmitted only via the subsystem s_y are called nongateway messages, and their set is denoted by $\text{NGM}(s_y)$ and 2) the messages released on the subsystem s_y forwarding to other subsystems through the gateway are called gateway messages, and their set is denoted by $\text{GM}(s_y)$. These two types of message sets are transmitted in bus *s2g&s2s* according to the architecture of Fig. 1. For example, as shown in Fig. 3, the messages released in the subsystem s_1 are $M(s_1) = \{m_1, m_2, m_3, m_4\}$, and $\text{NGM}(s_1) = \{m_4\}$ and $\text{GM}(s_1) = \{m_1, m_2, m_3\}$ are obtained.

D. Message-Processing Task Model

The gateway activates message-processing tasks when it receives gateway messages from subsystems [24]. The actual finish instants (AFI) of messages are sporadic; thus, a message-processing task model should be sporadic rather than periodic. Let τ_i=(P_{τ_i}, C_{τ_i}, s_{src,τ_i}, s_{dest,τ_i}) represent a message-processing task according to priority, execution time, source subsystem, and destination subsystem. P_{τ_i}, s_{src,τ_i}, and s_{dest,τ_i} inherit those of the corresponding incoming message. Although different messages exhibit different transmission time values [24], any two message-processing tasks have equal execution time values because the execution is a message copy process. In other words, for any two message-processing tasks τ_i and τ_j, we have $C_{\tau_i} = C_{\tau_j}$.

Considering that no offset exists for each message, all the messages in $M(s_y)$ are released simultaneously. The bus *g2s* is specifically responsible for receiving messages from the gateway; therefore, these messages do not influence all the messages in $\text{GM}(s_y)$ that are transmitted on the bus *s2g&s2s*. Subsequently, we can obtain the actual start instants (ASIs) and AFIs for each CAN message job in $\text{GM}(s_y)$ based on static nonpreemptive scheduling of the CAN bus. Each message-processing job is activated by the incoming message job triggered from the subsystem. Consequently, the released instant (RI) of the message-processing job $\tau_{i,k}$ should be the AFI of $m_{i,k}$. Considering that we aim to analyze message-processing tasks, we assume that all subsystems are designed well in advance and that all messages are schedulable. Therefore, all these message jobs in $\text{GM}(s_y)$ would repeat transmission in their hyper period $H(s_y)$, which is also called the least common multiple. The number of message jobs for m_i is $H(s_y)/T_{m_i}$. Finally, we can use the schedulability verification (SV) algorithm (see supplement materials) to obtain the RI of each message-processing job.

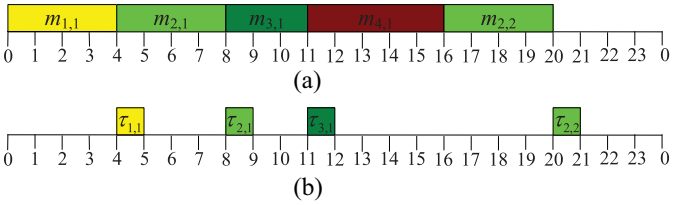


Fig. 4. ASIs and AFIs of message jobs released in s_1, and RIs of message-processing jobs triggered from s_1 of Fig. 3.

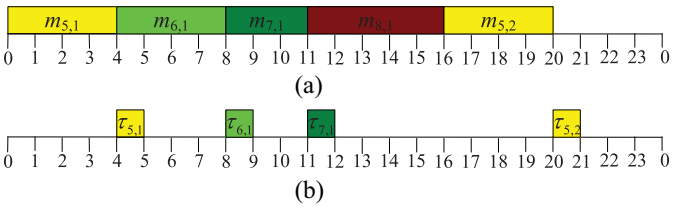


Fig. 5. ASIs and AFIs of message jobs released in s_2, and RIs of message-processing jobs triggered from s_2 of Fig. 3.

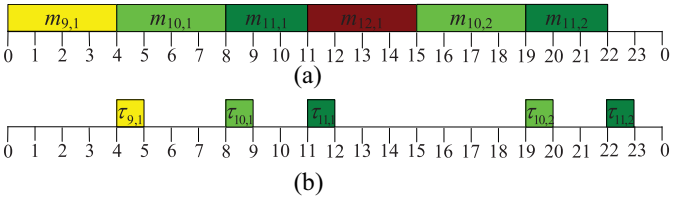


Fig. 6. ASIs and AFIs of message jobs released in s_3, and RIs of message-processing jobs triggered from s_3 of Fig. 3.

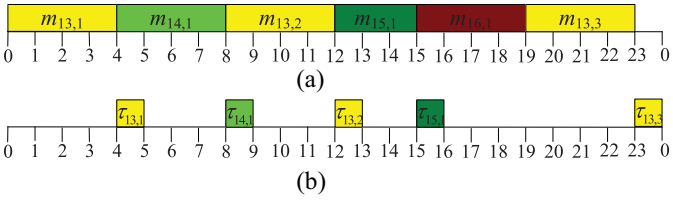


Fig. 7. ASIs and AFIs of message jobs released in s_4, and RIs of message-processing jobs triggered from s_4 of Fig. 3.

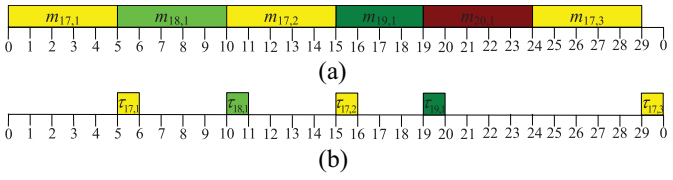


Fig. 8. ASIs and AFIs of message jobs released in s_5, and RIs of message-processing jobs triggered from s_5 of Fig. 3.

The SV algorithm can be used to verify the schedulability of the message set of each subsystem, and we discover that all the message sets of the eight subsystems of Table I are schedulable. Fig. 4(a) shows the ASIs and AFIs of message jobs released in s_1, and Fig. 4(b) shows the RIs of message-processing jobs triggered from s_1. The hyper period of the message set $M(s_1) = \{m_1, m_2, m_3, m_4\}$ is $H(s_1) = 24$. Notably, m_4 is not included in Fig. 4(b) because it is a non-gateway message, which cannot activate a message-processing job. Figs. 5–11 also show the ASIs and AFIs of message jobs and the RIs of message-processing jobs in (from) s_2 and s_3-s_8, respectively. Considering that gateway processing speed is faster than the transmission speed of a CAN bus, we regard the execution time of each message-processing task as 1 time unit in this paper.

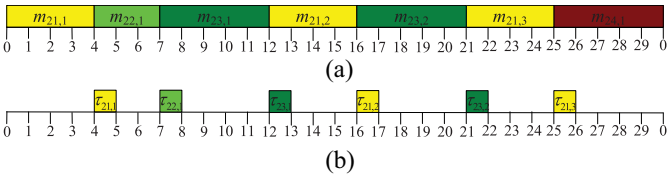


Fig. 9. ASIs and AFIs of message jobs released in s_6, and RIs of message-processing jobs triggered from s_6 of Fig. 3.

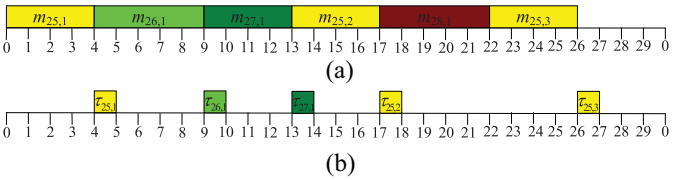


Fig. 10. ASIs and AFIs of message jobs released in s_7, and RIs of message-processing jobs triggered from s_7 of Fig. 3.

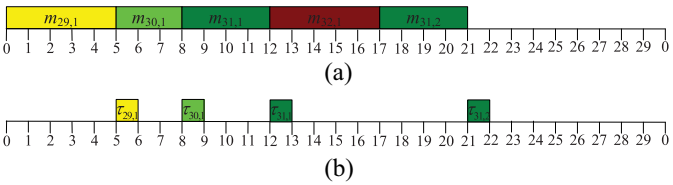


Fig. 11. ASIs and AFIs of message jobs released in s_8, and RIs of message-processing jobs triggered from s_8 of Fig. 3.

IV. WCRT Analysis Based on Global Scheduling

This section presents the WCRT analysis based on global scheduling.

A. Critical Instant Candidates

To analyze the WCRT of $\tau_{i,k}$ ($\tau_{i,k}$ is the message-processing job under analysis), the RIs of all message-processing jobs, which have higher priority than $\tau_{i,k}$ from all subsystems, should be obtained. We use $\mathrm{HP}(\tau_{i,k}, s_y)$ to represent the set of high-priority gateway messages for $\tau_{i,k}$ released in s_y. For example, to analyze the WCRT of $\tau_{31,1}$ triggered from s_8 in Fig. 11(b), the high-priority message-processing jobs from all subsystems denoted as follows would cause interference to $\tau_{31,1}$:

$$\begin{cases} \mathrm{HP}(\tau_{31,1}, s_1) = \{\tau_{1,1}, \tau_{2,1}, \tau_{3,1}, \tau_{2,2}\} \\ \mathrm{HP}(\tau_{31,1}, s_2) = \{\tau_{5,1}, \tau_{6,1}, \tau_{7,1}, \tau_{5,2}\} \\ \mathrm{HP}(\tau_{31,1}, s_3) = \{\tau_{9,1}, \tau_{10,1}, \tau_{11,1}, \tau_{10,2}, \tau_{11,2}\} \\ \mathrm{HP}(\tau_{31,1}, s_4) = \{\tau_{13,1}, \tau_{14,1}, \tau_{13,2}, \tau_{15,1}, \tau_{13,3}\} \\ \mathrm{HP}(\tau_{31,1}, s_5) = \{\tau_{17,1}, \tau_{18,1}, \tau_{17,2}, \tau_{19,1}, \tau_{17,3}\} \\ \mathrm{HP}(\tau_{31,1}, s_6) = \{\tau_{21,1}, \tau_{22,1}, \tau_{23,1}, \tau_{21,2}, \tau_{23,2}, \tau_{21,3}\} \\ \mathrm{HP}(\tau_{31,1}, s_7) = \{\tau_{25,1}, \tau_{26,1}, \tau_{27,1}, \tau_{25,2}, \tau_{25,3}\} \\ \mathrm{HP}(\tau_{31,1}, s_8) = \{\tau_{29,1}, \tau_{30,1}\}. \end{cases}$$

The critical instant (CI) for the job under analysis is defined as an instant at which a request for that job will have the

WCRT, and it occurs whenever the job is requested simultaneously with requests for all high-priority message-processing jobs [30]. The CI in global scheduling can be obtained because any message-processing task has the same execution time (i.e., 1 time unit). Hence, our objective is to determine the CI of each subsystem for the message-processing job under analysis and subsequently calculate the WCRT according to the obtained CIs. Each subsystem can be started at any instant after the in-vehicle network is initialized. Consequently, each released instant in a subsystem could be the CI. For example, we continue to assume that the message-processing job under analysis is $\tau_{31,1}$. Then one of the time instants in 4, 8, 11, or 20 of the subsystem S_1 [Fig. 4(b)] may be the CI. We first define the following terms before obtaining the CI.

Definition 1 [CI Candidates (CICs)] : The CICs for $\tau_{i,k}$ are the instants at which $\tau_{i,k}$ is released simultaneously with any of the high-priority message-processing jobs from the other subsystems and that may cause the WCRT. These CICs are grouped by subsystems, and the CICs from s_y are denoted as $\text{CICs}(\tau_{i,k}, s_y)$.

Assuming that $\tau_{i,k}$ is triggered from the subsystem s_a, then $\text{CICs}(\tau_{i,k}, s_a) = \{RI(\tau_{i,k})\}$. For any other subsystem s_y, the $\text{CICs}(\tau_{i,k}, s_y)$ are initialized as the released instants of $\text{HP}(\tau_{i,k}, s_y)$. For example, the CICs for $\tau_{31,1}$ are

$$\begin{cases} \text{CICs}(\tau_{31,1}, s_1) = \{4, 8, 11, 20\} \\ \text{CICs}(\tau_{31,1}, s_2) = \{4, 8, 11, 20\} \\ \text{CICs}(\tau_{31,1}, s_3) = \{4, 8, 11, 19, 22\} \\ \text{CICs}(\tau_{31,1}, s_4) = \{4, 8, 12, 15, 23\} \\ \text{CICs}(\tau_{31,1}, s_5) = \{5, 10, 15, 19, 29\} \\ \text{CICs}(\tau_{31,1}, s_6) = \{4, 7, 12, 16, 21, 25\} \\ \text{CICs}(\tau_{31,1}, s_7) = \{4, 9, 13, 17, 26\} \\ \text{CICs}(\tau_{31,1}, s_8) = \{12\}. \end{cases} \tag{1}$$

B. Our Approach

Our approach aims to verify whether an expected delay (EL) (i.e., expected interval) of each core can be interfered by high-priority message-processing jobs from all subsystems. Our main objective is to fill this EL with possible high-priority message-processing jobs from all CAN subsystems. The EL of each CPU core is explored from 1 to 2, 3, ... in increments of 1 time unit until it cannot be reachable by high-priority message-processing jobs. The concrete steps are as follows.

1) Let $\text{CL}(\tau_{i,k}, u_x)$ represent the current delay (CL) of the job $\tau_{i,k}$ in the core u_x. $\text{CL}(\tau_{i,k}, u_x)$ is the reachable delay that has been filled by the blocking of low-priority nonmessage-processing tasks and interference of high-priority message-processing jobs from all subsystems. Considering that all nonmessage-processing tasks take lower priorities and have larger execution time (denoted by B_{τ_i}) than (i.e., $B_{\tau_i} > C_{\tau_i}$) in the gateway, the initial CL for $\tau_{i,k}$ in each core is B_{τ_i}.
2) Select the core u_x with the minimum CL in all cores, and let $EL(\tau_{i,k}, u_x)$ represent the EL that will be verified in the core u_x and is calculated by
$$EL\big(\tau_{i,k}, u_x\big) = \text{CL}\big(\tau_{i,k}, u_x\big) + C_{\tau_{i,k}}.$$
3) Iteratively verify the EL $EL(\tau_{i,k}, u_x)$ from s_1 to $s_{|S|}$. If $EL(\tau_{i,k}, u_x)$ of the core u_x can be filled from subsystem s_y, then $\text{CL}(\tau_{i,k}, u_x)$ is updated to $EL(\tau_{i,k}, u_x)$. The core is selected according to step 2 to verify the new EL from s_{y+1} to $s_{|S|}$ if $y < |S|$.
4) If $y = |S|$, then the new EL is verified from s_1 to $s_{|S|}$.
5) If the verified EL $EL(\tau_{i,k}, u_x)$ cannot be filled by traversing all subsystems (from s_1 to $s_{|S|}$), then the final response time $R(\tau_{i,k})$ is $EL(\tau_{i,k}, u_x)$. $R(\tau_{i,k})$ is either a lower bound on response time (LBRT) $R^-(\tau_{i,k})$ or an upper bound on response time (UBRT) $R^+(\tau_{i,k})$ depending on different search conditions. We will first introduce how to obtain the LBRT and UBRT in the subsequent sections.

The search process in step 3 is referred to as the "one round search" because all subsystems are searched once. The whole process (steps 1–5) is referred to as the "round search" because it involves multiple rounds to search for CICs and increase the ELs.

The final reachable latencies of all cores are filled by the blocking of low-priority nonmessage-processing tasks and high-priority message-processing jobs from multiple subsystems; thus, each subsystem should contribute a certain amount of latencies, and the following is obtained:

$$\sum_{u_x \in U} \text{CL}\big(\tau_{i,k}, u_x\big) = \sum_{u_x \in U} B_{\tau_{i,k}} + \sum_{s_y \in S} \big(\text{CN}\big(\tau_{i,k}, s_y\big) \times C_{\tau_{i,k}}\big)$$

where $\text{CN}(\tau_{i,k}, s_y)$ is the contribution number (CN) of the high-priority message-processing jobs triggered from the subsystem s_y. The initial CNs for all subsystems are 0. If $EL(\tau_{i,k}, u_x)$ is going to be verified in the subsystem s_y, the following relationship is obtained:

$$\begin{aligned} EL\big(\tau_{i,k}, u_x\big) \; + \sum_{u_e \in U, e \neq x} \text{CL}\big(\tau_{i,k}, u_e\big) \\ = \sum_{u_e \in U} B_{\tau_{i,k}} + \text{EN}\big(\tau_{i,k}, s_y\big) \times C_{\tau_{i,k}} \\ + \sum_{s_f \in S, f \neq y} \big(\text{CN}\big(\tau_{i,k}, s_f\big) \times C_{\tau_{i,k}}\big). \end{aligned}$$

where $\text{EN}(\tau_{i,k}, s_y)$ represents the expected number (EN) of the high-priority message-processing jobs filling the $EL(\tau_{i,k}, u_x)$ from the subsystem s_y and is calculated as follows:

$$\text{EN}\big(\tau_{i,k}, s_y\big) = \text{CN}\big(\tau_{i,k}, s_y\big) + 1.$$

C. Multicore Round Search for LBRT

We first let $IS(\tau_{i,k}, s_y, \text{EN}(\tau_{i,k}, s_y), EL(\tau_{i,k}, u_x))$ represent whether at least $\text{EN}(\tau_{i,k}, s_y)$ high-priority message-processing jobs triggered from the subsystem s_y are available in the expected interval $EL(\tau_{i,k}, u_x)$ on the core u_x. After that, we explain the whole process of calculating the LBRT for $\tau_{31,1}$ before presenting the algorithm. The results are shown in Table II.

In initialization, the initial CICs for all subsystems are the same as shown in (1). The initial CL for each core is the blocking 2, and the CN for each subsystem is 0 (see initialization of Table II).

In Round 1, the operations are as follows.

1) The first selected core is u_1, and the verified EL is $EL(\tau_{31,1}, u_1) = 3$. $EL(\tau_{31,1}, u_1) = 3$ is verified in the

TABLE II
PROCESS AND RESULTS OF THE LBRT COMPUTATION FOR $\tau_{31,1}$ (THE VALID RESULTS ARE IN BOLD TEXT) USING MRSLR

Initialization	$\begin{cases} CL(\tau_{31,1}, u_1) = 2 \\ CL(\tau_{31,1}, u_2) = 2 \end{cases}$	$\begin{cases} CN(\tau_{31,1}, s_1) = 0 \\ CN(\tau_{31,1}, s_2) = 0 \\ CN(\tau_{31,1}, s_3) = 0 \\ CN(\tau_{31,1}, s_4) = 0 \\ CN(\tau_{31,1}, s_5) = 0 \\ CN(\tau_{31,1}, s_6) = 0 \\ CN(\tau_{31,1}, s_7) = 0 \\ CN(\tau_{31,1}, s_8) = 0 \end{cases}$	$\begin{cases} CICs(\tau_{31,1}, s_1) = \{4, 8, 11, 20\} \\ CICs(\tau_{31,1}, s_2) = \{4, 8, 11, 20\} \\ CICs(\tau_{31,1}, s_3) = \{4, 8, 11, 19, 22\} \\ CICs(\tau_{31,1}, s_4) = \{4, 8, 12, 15, 23\} \\ CICs(\tau_{31,1}, s_5) = \{5, 10, 15, 19, 29\} \\ CICs(\tau_{31,1}, s_6) = \{4, 7, 12, 16, 21, 25\} \\ CICs(\tau_{31,1}, s_7) = \{4, 9, 13, 17, 26\} \\ CICs(\tau_{31,1}, s_8) = \{12\}. \end{cases}$

Round	u_x	Expected delay	s_y	Current number	Search	CICs
Round 1	u_1	$EL(\tau_{31,1}, u_1)$=3	s_1	$EN(\tau_{31,1}, s_1)$=1	$IS(\tau_{31,1}, s_1, 1, 3)$	$CICs(\tau_{31,1}, s_1)$={**4,8,11,20**}
	u_2	$EL(\tau_{31,1}, u_2)$=3	s_2	$EN(\tau_{31,1}, s_2)$=1	$IS(\tau_{31,1}, s_2, 1, 3)$	$CICs(\tau_{31,1}, s_2)$={**4,8,11,20**}
	u_1	$EL(\tau_{31,1}, u_1)$=4	s_3	$EN(\tau_{31,1}, s_3)$=1	$IS(\tau_{31,1}, s_3, 1, 4)$	$CICs(\tau_{31,1}, s_3)$={**4, 8, 11, 19, 22**}
	u_2	$EL(\tau_{31,1}, u_2)$=4	s_4	$EN(\tau_{31,1}, s_4)$=1	$IS(\tau_{31,1}, s_4, 1, 4)$	$CICs(\tau_{31,1}, s_4)$={**4, 8, 12, 15, 23**}
	u_1	$EL(\tau_{31,1}, u_1)$=5	s_5	$EN(\tau_{31,1}, s_5)$=1	$IS(\tau_{31,1}, s_5, 1, 5)$	$CICs(\tau_{31,1}, s_5)$={**5, 10, 15, 19, 29**}
	u_2	$EL(\tau_{31,1}, u_2)$=5	s_6	$EN(\tau_{31,1}, s_6)$=1	$IS(\tau_{31,1}, s_6, 1, 5)$	$CICs(\tau_{31,1}, s_6)$={**4, 7, 12, 16, 21, 25**}
	u_1	$EL(\tau_{31,1}, u_1)$=6	s_7	$EN(\tau_{31,1}, s_7)$=1	$IS(\tau_{31,1}, s_7, 1, 6)$	$CICs(\tau_{31,1}, s_7)$={**4, 9, 13, 17, 26**}
	u_2	$EL(\tau_{31,1}, u_2)$=6	s_8	$EN(\tau_{31,1}, s_8)$=1	$IS(\tau_{31,1}, s_8, 1, 6)$	$CICs(\tau_{31,1}, s_8)$={}
Round 2	u_2	$EL(\tau_{31,1}, u_2)$=6	s_1	$EN(\tau_{31,1}, s_1)$=2	$IS(\tau_{31,1}, s_1, 2, 6)$	$CICs(\tau_{31,1}, s_1)$={**4,8**}
	u_1	$EL(\tau_{31,1}, u_1)$=7	s_2	$EN(\tau_{31,1}, s_2)$=2	$IS(\tau_{31,1}, s_2, 2, 7)$	$CICs(\tau_{31,1}, s_2)$={**4,8**}
	u_2	$EL(\tau_{31,1}, u_2)$=7	s_3	$EN(\tau_{31,1}, s_3)$=2	$IS(\tau_{31,1}, s_3, 2, 7)$	$CICs(\tau_{31,1}, s_3)$={**4, 8, 19, 22**}
	u_1	$EL(\tau_{31,1}, u_1)$=8	s_4	$EN(\tau_{31,1}, s_4)$=2	$IS(\tau_{31,1}, s_4, 2, 8)$	$CICs(\tau_{31,1}, s_4)$={**4, 8, 12, 23**}
	u_2	$EL(\tau_{31,1}, u_2)$=8	s_5	$EN(\tau_{31,1}, s_5)$=2	$IS(\tau_{31,1}, s_5, 2, 8)$	$CICs(\tau_{31,1}, s_5)$={**5, 10, 15, 29**}
	u_1	$EL(\tau_{31,1}, u_1)$=9	s_6	$EN(\tau_{31,1}, s_6)$=2	$IS(\tau_{31,1}, s_6, 2, 9)$	$CICs(\tau_{31,1}, s_6)$={**4, 7, 12, 16, 21**}
	u_2	$EL(\tau_{31,1}, u_2)$=9	s_7	$EN(\tau_{31,1}, s_7)$=2	$IS(\tau_{31,1}, s_7, 2, 9)$	$CICs(\tau_{31,1}, s_7)$={**4, 9, 13, 26**}
	u_1	$EL(\tau_{31,1}, u_1)$=10	s_8	$EN(\tau_{31,1}, s_8)$=1	$IS(\tau_{31,1}, s_8, 1, 10)$	$CICs(\tau_{31,1}, s_8)$={}
Round 3	u_1	$EL(\tau_{31,1}, u_1)$=10	s_1	$EN(\tau_{31,1}, s_1)$=3	$IS(\tau_{31,1}, s_1, 3, 10)$	$CICs(\tau_{31,1}, s_1)$={**4**}
	u_2	$EL(\tau_{31,1}, u_2)$=10	s_2	$EN(\tau_{31,1}, s_2)$=3	$IS(\tau_{31,1}, s_2, 3, 10)$	$CICs(\tau_{31,1}, s_2)$={**4**}
	u_1	$EL(\tau_{31,1}, u_1)$=11	s_3	$EN(\tau_{31,1}, s_3)$=3	$IS(\tau_{31,1}, s_3, 3, 11)$	$CICs(\tau_{31,1}, s_3)$={**4, 19, 22**}
	u_2	$EL(\tau_{31,1}, u_2)$=11	s_4	$EN(\tau_{31,1}, s_4)$=3	$IS(\tau_{31,1}, s_4, 3, 11)$	$CICs(\tau_{31,1}, s_4)$={**4, 8, 23**}
	u_1	$EL(\tau_{31,1}, u_1)$=12	s_5	$EN(\tau_{31,1}, s_5)$=3	$IS(\tau_{31,1}, s_5, 3, 12)$	$CICs(\tau_{31,1}, s_5)$={5, 10, 29}
	u_2	$EL(\tau_{31,1}, u_2)$=12	s_6	$EN(\tau_{31,1}, s_6)$=3	$IS(\tau_{31,1}, s_6, 3, 12)$	$CICs(\tau_{31,1}, s_6)$={**4, 7, 12, 16**}
	u_1	$EL(\tau_{31,1}, u_1)$=13	s_7	$EN(\tau_{31,1}, s_7)$=3	$IS(\tau_{31,1}, s_7, 3, 13)$	$CICs(\tau_{31,1}, s_7)$={**4, 9**}
	u_2	$EL(\tau_{31,1}, u_2)$=13	s_8	$EN(\tau_{31,1}, s_8)$=1	$IS(\tau_{31,1}, s_8, 1, 13)$	$CICs(\tau_{31,1}, s_8)$={}
Round 4	u_2	$EL(\tau_{31,1}, u_2)$=13	s_1	$EN(\tau_{31,1}, s_1)$=4	$IS(\tau_{31,1}, s_1, 4, 13)$	$CICs(\tau_{31,1}, s_1)$={}
	u_2	$EL(\tau_{31,1}, u_2)$=13	s_2	$EN(\tau_{31,1}, s_2)$=4	$IS(\tau_{31,1}, s_2, 4, 13)$	$CICs(\tau_{31,1}, s_2)$={}
	u_2	$EL(\tau_{31,1}, u_2)$=13	s_3	$EN(\tau_{31,1}, s_3)$=4	$IS(\tau_{31,1}, s_3, 4, 13)$	$CICs(\tau_{31,1}, s_3)$={}
	u_2	$EL(\tau_{31,1}, u_2)$=13	s_4	$EN(\tau_{31,1}, s_4)$=4	$IS(\tau_{31,1}, s_4, 4, 13)$	$CICs(\tau_{31,1}, s_4)$={**4**}
	u_1	$EL(\tau_{31,1}, u_1)$=14	s_5	$EN(\tau_{31,1}, s_5)$=4	$IS(\tau_{31,1}, s_5, 4, 14)$	$CICs(\tau_{31,1}, s_5)$={}
	u_1	$EL(\tau_{31,1}, u_1)$=14	s_6	$EN(\tau_{31,1}, s_6)$=4	$IS(\tau_{31,1}, s_6, 4, 14)$	$CICs(\tau_{31,1}, s_6)$={**4, 12**}
	u_2	$EL(\tau_{31,1}, u_2)$=14	s_7	$EN(\tau_{31,1}, s_7)$=4	$IS(\tau_{31,1}, s_7, 4, 14)$	$CICs(\tau_{31,1}, s_7)$={**4**}
	u_1	$EL(\tau_{31,1}, u_1)$=15	s_8	$EN(\tau_{31,1}, s_8)$=1	$IS(\tau_{31,1}, s_8, 1, 15)$	$CICs(\tau_{31,1}, s_8)$={}
Round 5	u_1	$EL(\tau_{31,1}, u_1)$=15	s_1	$EN(\tau_{31,1}, s_1)$=4	$IS(\tau_{31,1}, s_1, 4, 15)$	$CICs(\tau_{31,1}, s_1)$={}
	u_1	$EL(\tau_{31,1}, u_1)$=15	s_2	$EN(\tau_{31,1}, s_2)$=4	$IS(\tau_{31,1}, s_2, 4, 15)$	$CICs(\tau_{31,1}, s_2)$={}
	u_1	$EL(\tau_{31,1}, u_1)$=15	s_3	$EN(\tau_{31,1}, s_3)$=4	$IS(\tau_{31,1}, s_3, 4, 15)$	$CICs(\tau_{31,1}, s_3)$={**19, 22**}
	u_2	$EL(\tau_{31,1}, u_2)$=15	s_4	$EN(\tau_{31,1}, s_4)$=5	$IS(\tau_{31,1}, s_4, 5, 15)$	$CICs(\tau_{31,1}, s_4)$={}
	u_2	$EL(\tau_{31,1}, u_2)$=15	s_5	$EN(\tau_{31,1}, s_5)$=4	$IS(\tau_{31,1}, s_5, 4, 15)$	$CICs(\tau_{31,1}, s_5)$={**5**}
	u_1	$EL(\tau_{31,1}, u_1)$=16	s_6	$EN(\tau_{31,1}, s_6)$=5	$IS(\tau_{31,1}, s_6, 5, 16)$	$CICs(\tau_{31,1}, s_6)$={}
	u_1	$EL(\tau_{31,1}, u_1)$=16	s_7	$EN(\tau_{31,1}, s_7)$=5	$IS(\tau_{31,1}, s_7, 5, 16)$	$CICs(\tau_{31,1}, s_7)$={}
	u_1	$EL(\tau_{31,1}, u_1)$=16	s_8	$EN(\tau_{31,1}, s_8)$=1	$IS(\tau_{31,1}, s_8, 1, 16)$	$CICs(\tau_{31,1}, s_8)$={}
Round 6	u_1	$EL(\tau_{31,1}, u_1)$=16	s_1	$EN(\tau_{31,1}, s_1)$=4	$IS(\tau_{31,1}, s_1, 4, 16)$	$CICs(\tau_{31,1}, s_1)$={}
	u_1	$EL(\tau_{31,1}, u_1)$=16	s_2	$EN(\tau_{31,1}, s_2)$=4	$IS(\tau_{31,1}, s_2, 4, 16)$	$CICs(\tau_{31,1}, s_2)$={}
	u_1	$EL(\tau_{31,1}, u_1)$=16	s_3	$EN(\tau_{31,1}, s_3)$=5	$IS(\tau_{31,1}, s_3, 5, 16)$	$CICs(\tau_{31,1}, s_3)$={}
	u_1	$EL(\tau_{31,1}, u_1)$=16	s_4	$EN(\tau_{31,1}, s_4)$=5	$IS(\tau_{31,1}, s_4, 5, 16)$	$CICs(\tau_{31,1}, s_4)$={}
	u_1	$EL(\tau_{31,1}, u_1)$=16	s_5	$EN(\tau_{31,1}, s_5)$=5	$IS(\tau_{31,1}, s_5, 5, 16)$	$CICs(\tau_{31,1}, s_5)$={}
	u_1	$EL(\tau_{31,1}, u_1)$=16	s_6	$EN(\tau_{31,1}, s_6)$=5	$IS(\tau_{31,1}, s_6, 5, 16)$	$CICs(\tau_{31,1}, s_6)$={}
	u_1	$EL(\tau_{31,1}, u_1)$=16	s_7	$EN(\tau_{31,1}, s_7)$=5	$IS(\tau_{31,1}, s_7, 5, 16)$	$CICs(\tau_{31,1}, s_7)$={}
	u_1	$EL(\tau_{31,1}, u_1)$=16	s_8	$EN(\tau_{31,1}, s_8)$=1	$IS(\tau_{31,1}, s_8, 1, 16)$	$CICs(\tau_{31,1}, s_8)$={}

In Round 6, all subsystems cannot search for more high-priority message-processing jobs to fill $EL(\tau_{31,1}, u_1)$=16. Hence, the LBRT is $R^-(\tau_{31,1}) = 16$, and the final CNs for each subsystems are $CN(\tau_{31,1}, s_1)$=3, $CN(\tau_{31,1}, s_2)$=3, $CN(\tau_{31,1}, s_3)$=4, $CN(\tau_{31,1}, s_4)$=4, $CN(\tau_{31,1}, s_5)$=4, $CN(\tau_{31,1}, s_6)$=4, $CN(\tau_{31,1}, s_7)$=4, $CN(\tau_{31,1}, s_8)$=0.

subsystem s_1; the result is true, and the obtained CICs are CICs$(\tau_{31,1}, s_1) = \{4, 8, 11, 20\}$, and CL$(\tau_{31,1}, u_1)$ is updated to 3.

2) The selected core is changed to u_2 because it exhibits lower CL than u_1 (2<3). In this case, $EL(\tau_{31,1}, u_2) = 3$ is verified in the subsystem s_2 because s_1 has been verified by the previous operation.

The result is true, the obtained CICs are CICs$(\tau_{31,1}, s_2) = \{4, 8, 11, 20\}$, and CL$(\tau_{31,1}, u_2)$ is updated to 3. The same pattern is searched from s_3 to s_8. Finally, the above search process is referred to as "Round 1" because all subsystems are searched once. Notably, the result of $IS(\tau_{31,1}, s_8, 1, 6)$ is false, and the obtained CICs are null.

The search for each subsystem in Round 1 is repeated in Rounds 2–6. Remarkably, two important rules exist in the analysis.

1) For any s_x, "the newly obtained CICs$(\tau_{i,k}, s_y)$ of the current round are selected from those of the latest previous round with nonempty CICs$(\tau_{i,k}, s_y)$." For example, the newly obtained CICs by s_1 in Round 2 are CICs$(\tau_{31,1}, s_1) = \{4, 8\}$, which are selected from the obtained CICs$(\tau_{31,1}, s_1) = \{4, 8, 11, 20\}$ in

Algorithm 1 MRSLR Algorithm

Input: $U = \{u_1, u_2, ..., u_{|U|}\}$, $S = \{s_1, s_2, ..., s_{|S|}\}$, $\tau_{i,k}$;
Output: LBRT $R^-(\tau_{i,k})$ and its related values;
1: **for** $(x \leftarrow 1; x \leqslant |U|; x++)$ **do**
2: Initialize $CL(\tau_{i,k}, u_x) \leftarrow B_{\tau_i}$;
3: **end for**
4: **for** $(y \leftarrow 1; y \leqslant |S|; y++)$ **do**
5: Initialize $CICs(\tau_{i,k}, s_y)$ based on Definition 1;
6: Initialize $CN(\tau_{i,k}, s_y) \leftarrow B_{\tau_i}$;
7: **end for**
8: $round \leftarrow 1$;
9: **while** (*true*) **do**
10: Select the core u_x with the minimum $CL(\tau_{i,k}, u_x)$ among all cores;
11: **for** $(y \leftarrow 1; y \leqslant |S|; y++)$ **do**
12: $EL(\tau_{i,k}, u_x) = CL(\tau_{i,k}, u_x) + C_{\tau_i}$;
13: $EN(\tau_{i,k}, s_y) = CN(\tau_{i,k}, s_y) + C_{\tau_i}$;
14: $result \leftarrow IS(\tau_{i,k}, s_y, EN(\tau_{i,k}, s_y), EL(\tau_{i,k}, u_x))$; // using the inherited search to search CICs;
15: **if** ($result$ == *true*) **then**
16: Update $CICs(\tau_{i,k}, s_y)$ as the newly found CICs;
17: $CN(\tau_{i,k}, s_y) \leftarrow CN(\tau_{i,k}, s_y) + C_{\tau_i}$;
18: $CL(\tau_{i,k}, u_x) \leftarrow EL(\tau_{i,k}, u_x)$;
19: **if** $(y < |S|)$ **then**
20: Select the core u_x with the minimum $CL(\tau_{i,k}, u_x)$ among all cores;
21: **end if**
22: **end if**
23: **end for**
24: $round$++;
25: **if** (the search results are false for all subsystems) **then**
26: $R^-(\tau_{i,k}) \leftarrow EL(\tau_{i,k}, u_x)$;
27: return;
28: **end if**
29: **end while**

Algorithm 2 MRSUR Algorithm

Input: $U = \{u_1, u_2, ..., u_{|U|}\}$, $S = \{s_1, s_2, ..., s_{|S|}\}$, $\tau_{i,k}$, $R^-(\tau_{i,k})$ and its related values using MRSLR;
Output: UBRT $R^+(\tau_{i,k})$ and its related values;
1: **for** $(x \leftarrow 1; x \leqslant |U|; x++)$ **do**
2: Initialize $CL(\tau_{i,k}, u_x)$ as the final $CL(\tau_{i,k}, u_x)$ using MRSLR;
3: **end for**
4: **for** $(y \leftarrow 1; y \leqslant |S|; y++)$ **do**
5: Initialize $CICs(\tau_{i,k}, s_y)$ based on Definition 1;
6: $CN(\tau_{i,k}, s_y)$ is the final $CN(\tau_{i,k}, s_y)$ using MRSLR;
7: **end for**
8: var $round \leftarrow 1$;
9: **while** (*true*) **do**
10: Select the core u_x with the minimum $CL(\tau_{i,k}, u_x)$ among all cores;
11: **for** $(y \leftarrow 1; y \leqslant |S|; y++)$ **do**
12: $EL(\tau_{i,k}, u_x) = CL(\tau_{i,k}, u_x) + C_{\tau_i}$;
13: $EN(\tau_{i,k}, s_y) = CN(\tau_{i,k}, s_y) + C_{\tau_i}$;
14: $result \leftarrow SS(\tau_{i,k}, s_y, EN(\tau_{i,k}, s_y), EL(\tau_{i,k}, u_x))$; // using the shared search to search CICs;
15: **if** ($result$ == *true*) **then**
16: Update $CICs(\tau_{i,k}, s_y)$ as the newly found CICs;
17: $CN(\tau_{i,k}, s_y) \leftarrow CN(\tau_{i,k}, s_y) + C_{\tau_i}$;
18: $CL(\tau_{i,k}, u_x) \leftarrow EL(\tau_{i,k}, u_x)$;
19: **if** $(y < |S|)$ **then**
20: Select the core u_x with the minimum $CL(\tau_{i,k}, u_x)$ among all cores;
21: **end if**
22: **end if**
23: **end for**
24: $round$++;
25: **if** (the search results are false for all subsystems) **then**
26: $R^+(\tau_{i,k}) \leftarrow EL(\tau_{i,k}, u_x)$;
27: return;
28: **end if**
29: **end while**

Round 1; the newly found CICs by s_3 in Round 5 are $\text{CICs}(\tau_{31,1}, s_3) = \{9, 22\}$, which are selected from the obtained $\text{CICs}(\tau_{31,1}, s_3) = \{4, 19, 22\}$ in Round 3. This search process is referred to as the inherited search (IS) because the current round inherits the obtained CICs of the previous round.

2) When all subsystems cannot search for more high-priority message-processing jobs to fill the EL in the same round, then the final LBRT is the EL. For example, in Round 6, all subsystems cannot search for more high-priority message-processing jobs to fill the $EL(\tau_{31,1}, u_1) = 16$ using the IS. Hence, the delay 16 is unreachable, and the LBRT is $R^-(\tau_{31,1}) = 15 + 1 = 16$, namely, the maximum reachable delay 15 pluses 1 time unit execution time. The IS can guarantee that the final obtained LBRT is a successive and real response time. This premise is proven in Theorem 1.

Subsequently, we propose the multicore round search for LBRT (MRSLR) algorithm described in Algorithm 1.

Theorem 1: The response time $R^-(\tau_{i,k})$ computed by the MRSLR algorithm is an actual existing response time that is less than or equal to the WCRT.

Proof: When the $EL(\tau_{i,k}, u_x)$ on u_x is explored from $B_{\tau_i} + C_{\tau_i}$ to $B_{\tau_i} + 2C_{\tau_i}$, $B_{\tau_i} + 3C_{\tau_i}$,... with increasing C_{τ_i} until it cannot be filled by high-priority message-processing jobs. If we can prove that $EL(\tau_{i,k}, u_x)$ (starts at $B_{\tau_i} + C_{\tau_i}$) increases successively, then the final nonincreased $EL(\tau_{i,k}, u_x)$ should be the successive response time for the following reasons. First, MRSLR constantly selects the core u_x with the minimum $\text{CL}(\tau_{i,k}, u_x)$ in all cores, and the difference of CLs among these cores does not exceed C_{τ_i}; therefore, $EL(\tau_{i,k}, u_x)$ increases successively for each core (line 20). Second, the condition of the IS is that the newly obtained $\text{CICs}(\tau_{i,k}, s_y)$ of the current round are selected from those of the latest previous round with nonempty $\text{CICs}(\tau_{i,k}, s_y)$. Consequently, the $EL(\tau_{i,k}, u_x)$ increases successively in the same subsystem on different rounds (line 14). Third, one round comprises the $|S|$ ISs such that the $EL(\tau_{i,k}, u_x)$ increases successively on the same round (lines 11–23). Fourth, the round search comprises multiple rounds such that the $EL(\tau_{i,k}, u_x)$ increases successively in the whole round search (lines 9–29). Hence, the response time $R^-(\tau_{i,k}) = EL(\tau_{i,k})$ is an actual existing response time that is less than or equal to the WCRT. ■

D. Multicore Round Search for UBRT

The MRSLR algorithm can obtain the actual existing LBRT because the IS uses a strict condition, that is, the newly obtained $\text{CICs}(\tau_{i,k}, s_y)$ of the current round are selected from those of the latest previous round with nonempty $\text{CICs}(\tau_{i,k}, s_y)$. When the condition is loosened slightly, we can still continuously perform the search based on the result of the MRSLR algorithm. The relaxed condition is described as follows: "the newly obtained $\text{CICs}(\tau_{i,k}, s_y)$ of the current round are always selected from those of the initial $\text{CICs}(\tau_{i,k}, s_y)$ in Definition 1." The search with this condition is called the shared search (SS) because the current round always uses the same shared results of the initial $\text{CICs}(\tau_{i,k}, s_y)$. When SS is used in the round search, the obtained response time is an UBRT because it ignores the succession of the interference. This proposition is proven in Theorem 2. Correspondingly, we propose the global scheduled round search (GSRS) for UBRT (MRSUR) algorithm explained in Algorithm 2.

Theorem 2: The response time $R^+(\tau_{i,k})$ computed by the MRSUR algorithm is a safe UBRT, which is larger than or equal to the WCRT.

Proof: The above condition can be explained by the ending condition of the MRSUR algorithm in which all the

subsystems cannot search for a further EN of high-priority message-processing jobs in an EL to fill it.

No more than $\mathrm{CN}(\tau_{i,k}, s_1)$ high-priority message-processing jobs exist in $R^+(\tau_{i,k})$ triggered from s_1.

No more than $\mathrm{CN}(\tau_{i,k}, s_2)$ high-priority message-processing jobs exist in $R^+(\tau_{i,k})$ triggered from $s_2 \ldots$

No more than $\mathrm{CN}(\tau_{i,k}, s_{|S|})$ high-priority message-processing jobs exist in $R^+(\tau_{i,k})$ triggered from $s_{|S|}$.

Moreover, we have $\sum_{u_x \in U} B_{\tau_{i,k}} + \sum_{s_y \in S} (\mathrm{CN}(\tau_{i,k}, s_y) \times C_{\tau_{i,k}}) \leqslant \sum_{u_x \in U} (R^+(\tau_{i,k}))$ according to the search process. Therefore, the $R^+(\tau_{i,k})$ cannot be filled by the high-priority message-processing jobs from all the subsystems even if the condition of successive interference is not considered. Therefore, the obtained $R^+(\tau_{i,k})$ using MRSUR is a safe UBRT that is larger than or equal to the WCRT. ■

Table III shows the whole process of obtaining the UBRT for $\tau_{31,1}$.

The initial CICs for all subsystems during initialization are still shown in (1). The initial CL for each core and CN for each subsystem are obtained from those of the MRSLR algorithms (see initialization of Table III).

In Round 1, we first use the SS $SS(\tau 31, 1, s_1, 4, 16)$ to search for the CICs. We obtain $\mathrm{CICs}(\tau 31, 1, s_1) = \{20\}$. These CICs can be observed because they are constantly selected from those of the initialization in the SS. This search improves the possibility of finding the CICs in the current round at the cost of ignoring the succession of the interfering delay. The same search is used for s_2–s_8 of the current and the following rounds.

Finally, all subsystems cannot search for further high-priority message-processing jobs to fill $EL(\tau_{31,1}, u_2) = 18$ in Round 2. Hence, the UBRT is $R^+(\tau_{31,1}) = 18$.

V. WCRT Analysis Based on Partitioned Scheduling

This section presents WCRT analysis based on partitioned scheduling by using a bin packing algorithm.

A. Bin Packing Problem

We first group all the message-processing jobs from the same subsystem and define them as follows.

Definition 2 (Message-Processing Job Sequence): The message-processing job sequence of the subsystem s_y contains the message-processing jobs triggered in s_y in a hyper period $H(s_y)$ and are denoted as $\mathrm{seq}(s_y)$.

The sequences of the motivating example can be easily obtained as follows:

$$\begin{cases} \mathrm{seq}(s_1) = \{\tau_{1,1}, \tau_{2,1}, \tau_{3,1}, \tau_{2,2}\} \\ \mathrm{seq}(s_2) = \{\tau_{5,1}, \tau_{6,1}, \tau_{7,1}, \tau_{5,2}\} \\ \mathrm{seq}(s_3) = \{\tau_{9,1}, \tau_{10,1}, \tau_{11,1}, \tau_{10,2}, \tau_{11,2}\} \\ \mathrm{seq}(s_4) = \{\tau_{13,1}, \tau_{14,1}, \tau_{13,2}, \tau_{15,1}, \tau_{13,3}\} \\ \mathrm{seq}(s_5) = \{\tau_{17,1}, \tau_{18,1}, \tau_{17,2}, \tau_{19,1}\} \\ \mathrm{seq}(s_6) = \{\tau_{21,1}, \tau_{22,1}, \tau_{23,1}, \tau_{21,2}, \tau_{23,2}, \tau_{21,3}\} \\ \mathrm{seq}(s_7) = \{\tau_{25,1}, \tau_{26,1}, \tau_{27,1}, \tau_{25,2}, \tau_{25,3}\} \\ \mathrm{seq}(s_8) = \{\tau_{29,1}, \tau_{30,1}, \tau_{31,1}, \tau_{31,2}\}. \end{cases}$$

Algorithm 3 SFFD Algorithm

Input: $U = \{u_1, u_2, \ldots, u_{|U|}\}$, $S = \{s_1, s_2, \ldots, s_{|S|}\}$, $\tau_{i,k}$;
Output: Partitioned groups $\{g_1, g_2, \ldots\}$;
1: **for** $(y \leftarrow 1; y \leqslant |S|; y++)$ **do**
2: Obtain $seq(s_y)$ and $H(s_y)$;
3: **end for**
4: Calculate $size(s_y) = \frac{num(seq(s_y))}{H(s_y)}$ using Eq.(2);
5: Calculate $size(u_x) = \frac{\sum_{s_y \in S} (size(s_y))}{|U|}$ using Eq.(3);
6: Order sequences according to a descending order of $size(s_y)$ in a seq_des_list;
7: **while** (there are sequences to be assigned in seq_des_list) **do**
8: Select the sequence $seq(s_y)$ from seq_des_list;
9: **for** $(x \leftarrow 1; x \leqslant |U|; x++)$ **do**
10: Obtain the remaining size $size(u_x)$ of the core u_x;
11: **if** ($size(s_y)$ can fit into $size(u_x)$) **then**
12: Assign all the jobs of the $seq(s_y)$ into the u_x;
13: Put s_y into the group g_x;
14: Update the size of core u_x as $size(u_x) \leftarrow (size(u_x) - size(s_y))$;
15: break;
16: **end if**
17: **end for**
18: **if** ($size(s_y)$ cannot fit to all the cores) **then**
19: Assign all the jobs of the $seq(s_y)$ into the core $u_{\max}$ with maximum remaining size $size(u_{\max}) = \max_{u_x \in U}\{size(u_x)\}$;
20: Put s_y into the group $g_{\max}$;
21: Update the size of core $size(u_{\max})$ as $size(u_{\max}) \leftarrow (size(u_{\max}) - size(s_y))$;
22: **end if**
23: **end while**

In general, tasks are statically assigned to cores in partitioned scheduling. However, the message-processing task jobs triggered from the same subsystem would form a message-processing sequence. Hence, we consider that all the message-processing jobs in the same sequence are assigned to the same core. Unlike general real-time tasks, our bin packing approach for message-processing jobs in the gateway is based on message-processing job sequence rather than tasks.

According to the basic idea of bin packing approaches, the "size" of all sequences and the specific size of each core should be obtained. First, we assume that the size $\mathrm{size}(s_y)$ of the sequence $\mathrm{seq}(s_y)$ is the trigged frequency in the hyper period $H(s_y)$ and calculated as follows:

$$\mathrm{size}(s_y) = \frac{\mathrm{num}\big(\mathrm{seq}(s_y)\big)}{H(s_y)} \tag{2}$$

where $\mathrm{num}(\mathrm{seq}(s_y))$ represents the number of message-processing jobs in the sequence $\mathrm{seq}(s_y)$. The above size of a sequence indicates the accumulated utilization on one core triggered by a same subsystem because the execution time of any message-processing job is 1 time unit. After that, we assume that the $\mathrm{size}(u_x)$ is the sum of $\mathrm{size}(s_y)$ of all sequences dividing the number of cores and is calculated as follows:

$$\mathrm{size}(u_x) = \frac{\sum\limits_{s_y \in S} \big(\mathrm{size}(s_y)\big)}{|U|}. \tag{3}$$

Therefore, each core has the same size of $\mathrm{size}(u_x)$. Note that the above size of each core is not the true capacity for task running, but an expected average size to achieve balanced workload on all cores.

TABLE III
PROCESS AND RESULTS OF THE UBRT COMPUTATION FOR $\tau_{31,1}$ (THE VALID RESULTS ARE IN BOLD TEXT) USING MRSUR

Initialization	$\begin{cases} CL(\tau_{31,1},u_1)=15 \\ CL(\tau_{31,1},u_2)=15 \end{cases}$		$\begin{cases} CN(\tau_{31,1},s_1)=3 \\ CN(\tau_{31,1},s_2)=3 \\ CN(\tau_{31,1},s_3)=4 \\ CN(\tau_{31,1},s_4)=4 \\ CN(\tau_{31,1},s_5)=4 \\ CN(\tau_{31,1},s_6)=4 \\ CN(\tau_{31,1},s_7)=4 \\ CN(\tau_{31,1},s_8)=0 \end{cases}$		$\begin{cases} CICs(\tau_{31,1},s_1)=\{4,8,11,20\} \\ CICs(\tau_{31,1},s_2)=\{4,8,11,20\} \\ CICs(\tau_{31,1},s_3)=\{4,8,11,19,22\} \\ CICs(\tau_{31,1},s_4)=\{4,8,12,15,23\} \\ CICs(\tau_{31,1},s_5)=\{5,10,15,19,29\} \\ CICs(\tau_{31,1},s_6)=\{4,7,12,16,21,25\} \\ CICs(\tau_{31,1},s_7)=\{4,9,13,17,26\} \\ CICs(\tau_{31,1},s_8)=\{12\}. \end{cases}$
Round	u_x	Expected delay	s_y	Current number	Global Search	CICs
Round 1	u_1	$EL(\tau_{31,1},u_1)$=16	s_1	$EN(\tau_{31,1},s_1)$=4	$SS(\tau_{31,1},s_1,4,16)$	$CICs(\tau_{31,1},s_1)$={**20**}
	u_2	$EL(\tau_{31,1},u_2)$=16	s_2	$EN(\tau_{31,1},s_2)$=4	$SS(\tau_{31,1},s_2,4,16)$	$CICs(\tau_{31,1},s_2)$={**20**}
	u_1	$EL(\tau_{31,1},u_1)$=17	s_3	$EN(\tau_{31,1},s_3)$=5	$SS(\tau_{31,1},s_3,5,17)$	$CICs(\tau_{31,1},s_3)$={**19**}
	u_2	$EL(\tau_{31,1},u_2)$=17	s_4	$EN(\tau_{31,1},s_4)$=5	$SS(\tau_{31,1},s_4,5,17)$	$CICs(\tau_{31,1},s_4)$={**23**}
	u_1	$EL(\tau_{31,1},u_1)$=18	s_5	$EN(\tau_{31,1},s_5)$=5	$SS(\tau_{31,1},s_5,5,18)$	$CICs(\tau_{31,1},s_5)$={}
	u_1	$EL(\tau_{31,1},u_1)$=18	s_6	$EN(\tau_{31,1},s_6)$=5	$SS(\tau_{31,1},s_6,5,18)$	$CICs(\tau_{31,1},s_6)$={**4**}
	u_2	$EL(\tau_{31,1},u_2)$=18	s_7	$EN(\tau_{31,1},s_7)$=5	$SS(\tau_{31,1},s_7,5,18)$	$CICs(\tau_{31,1},s_7)$={}
	u_2	$EL(\tau_{31,1},u_2)$=18	s_8	$EN(\tau_{31,1},s_8)$=1	$SS(\tau_{31,1},s_8,1,18)$	$CICs(\tau_{31,1},s_8)$={}
Round 2	u_2	$EL(\tau_{31,1},u_2)$=18	s_1	$EN(\tau_{31,1},s_1)$=5	$SS(\tau_{31,1},s_1,5,18)$	$CICs(\tau_{31,1},s_1)$={}
	u_2	$EL(\tau_{31,1},u_2)$=18	s_2	$EN(\tau_{31,1},s_2)$=5	$SS(\tau_{31,1},s_2,5,18)$	$CICs(\tau_{31,1},s_2)$={}
	u_2	$EL(\tau_{31,1},u_2)$=18	s_3	$EN(\tau_{31,1},s_3)$=6	$SS(\tau_{31,1},s_3,6,18)$	$CICs(\tau_{31,1},s_3)$={}
	u_2	$EL(\tau_{31,1},u_2)$=18	s_4	$EN(\tau_{31,1},s_4)$=6	$SS(\tau_{31,1},s_4,6,18)$	$CICs(\tau_{31,1},s_4)$={}
	u_2	$EL(\tau_{31,1},u_2)$=18	s_5	$EN(\tau_{31,1},s_5)$=5	$SS(\tau_{31,1},s_5,5,18)$	$CICs(\tau_{31,1},s_5)$={}
	u_2	$EL(\tau_{31,1},u_2)$=18	s_6	$EN(\tau_{31,1},s_6)$=6	$SS(\tau_{31,1},s_6,6,18)$	$CICs(\tau_{31,1},s_6)$={}
	u_2	$EL(\tau_{31,1},u_2)$=18	s_7	$EN(\tau_{31,1},s_7)$=5	$SS(\tau_{31,1},s_7,5,18)$	$CICs(\tau_{31,1},s_7)$={}
	u_2	$EL(\tau_{31,1},u_2)$=18	s_8	$EN(\tau_{31,1},s_8)$=1	$SS(\tau_{31,1},s_8,1,18)$	$CICs(\tau_{31,1},s_8)$={}

In Round 2, all subsystems cannot search for more high-priority message-processing jobs to fill $EL(\tau_{31,1},u_2)$=18. Hence, the UBRT is $R^+(\tau_{31,1})=18$, and the final CNs for each subsystems are $CN(\tau_{31,1},s_1)$=4, $CN(\tau_{31,1},s_2)$=4, $CN(\tau_{31,1},s_3)$=5, $CN(\tau_{31,1},s_4)$=5, $CN(\tau_{31,1},s_5)$=4, $CN(\tau_{31,1},s_6)$=5, $CN(\tau_{31,1},s_7)$=4, $CN(\tau_{31,1},s_8)$=0.

TABLE IV
PROPERTY VALUES OF EACH SUBSYSTEM

s_y	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
$num(seq(s_k))$	4	4	5	5	4	6	5	4
$H(s_y)$	24	24	24	24	30	30	30	30
$size(s_y)$	0.17	0.17	0.21	0.21	0.17	0.2	0.17	0.13
$size(u_x)$	0.71	0.71	0.71	0.71	0.71	0.71	0.71	0.71

TABLE V
PARTITIONED GROUPS OF THE SUBSYSTEMS (THE VALID RESULTS ARE IN BOLD TEXT)

s_y	$size(s_y)$	$size(u_1)$	$size(u_2)$
s_3	0.21	**0.71**	
s_4	0.21	**0.5**	
s_6	0.2	**0.29**	
s_1	0.17	0.09	**0.71**
s_2	0.17	0.09	**0.54**
s_5	0.17	0.09	**0.37**
s_7	0.17	0.09	**0.2**
s_8	0.13	**0.09**	0.03

B. Subsystem-Based FFD

After determining the required size(u_x) of the bin packing approach, we present the sequence-based FFD (SFFD) algorithm (Algorithm 3).

The main steps of the SFFD algorithm are as follows.

1) Obtain the sequence (lines 1–3) and size of each subsystem (line 4) as well as the size of each core (line 5).
2) Order sequences by decreasing order of size(s_y) (line 6).
3) Iteratively assign sequences to the cores upon which they fit according to the FF condition (lines 7–23).

C. Example of WCRT Analysis Based on Partitioned Scheduling

In the following, we use the motivating example to explain the method of obtaining the WCRT by using the partitioned scheduling.

1) We obtain the size of each subsystem using (2), and the size of each core is obtained using (3) (Table IV).
2) We obtain the partitioned groups $g_1=\{s_3,s_4,s_6,s_8\}$ and $g_2=\{s_1,s_2,s_5,s_7\}$ using the SFFD algorithm. The details are shown in Table V. For example, the size of s_3 is 0.21, and it is assigned to group g_1 because 0.21 can fit into 0.71, which is the size of the core u_1. The same pattern is used for s_4 and s_6 until the size of s_1 cannot fit into the size of u_1. Subsequently, s_1, s_2, s_5, and s_7 are included in group g_2 based on SFFD. Finally, the size of s_8 cannot fit into the sizes of u_1 and u_2, and it is included in group g_1, which has the maximum size.
3) We can reuse the MRSLR and MRSUR algorithms to calculate the LBRT and UBRT of $\tau_{i,k}$ after the partition, as long as we assume that only one core in the inputs of algorithms (i.e., $U=\{u_1\}$). For example, to further calculate the $R^+(\tau_{31,1})$ using the partitioned scheduling, the group $g_1=\{s_3,s_4,s_6,s_8\}$ will be scheduled in the core u_1, and the CICs for $\tau_{31,1}$ are

$$\begin{cases} \text{CICs}(\tau_{31,1},s_3)=\{4,8,11,19,22\} \\ \text{CICs}(\tau_{31,1},s_4)=\{4,8,12,15,23\} \\ \text{CICs}(\tau_{31,1},s_6)=\{4,7,12,16,21,25\} \\ \text{CICs}(\tau_{31,1},s_8)=\{12\}. \end{cases}$$

4) Finally, we obtain that $R^-(\tau_{31,1})=14$ and $R^+(\tau_{31,1})=15$ using the MRSLR and MRSUR algorithms, respectively.

TABLE VI
MESSAGE SETS OF SUBSYSTEMS

Subsystem	Number of messages	Task name	Number of Released instants
s_1	65	$\tau_1 - \tau_{65}$	509
s_2	65	$\tau_{101} - \tau_{165}$	509
s_3	69	$\tau_{201} - \tau_{269}$	468
s_4	69	$\tau_{301} - \tau_{369}$	468
s_5	85	$\tau_{401} - \tau_{485}$	614
s_6	85	$\tau_{501} - \tau_{585}$	614
s_7	89	$\tau_{601} - \tau_{689}$	571
s_8	89	$\tau_{701} - \tau_{789}$	571

TABLE VII
RESPONSE TIME OF NINE LOW-PRIORITY MESSAGE-PROCESSING JOBS IN THE MOTIVATING EXAMPLE OF TABLE I

$\tau_{i,k}$	GSRS		GSEE	PSRS		PSEE
	R^-	R^+	R^e	R^-	R^+	R^e
$\tau_{31,1}$	16	18	17	14	15	15
$\tau_{31,2}$	16	19	19	14	16	16
$\tau_{30,2}$	16	18	17	14	15	15
$\tau_{29,2}$	16	18	17	14	15	15
$\tau_{27,1}$	13	13	13	13	13	13
$\tau_{26,1}$	13	13	13	13	13	13
$\tau_{25,1}$	12	12	12	12	12	12
$\tau_{25,2}$	12	12	12	12	12	12
$\tau_{25,3}$	12	12	12	12	12	12

VI. WCRT EVALUATION WITH EXPERIMENTS

A. Experimental Message Sets

To evaluate the applicability of the proposed methods, we used four real CAN message sets provided by an automaker. The message sets contain 65, 69, 85, and 89 messages and are assigned into 14 ECUs of a CAN subsystem in 500 Kb/s. The concrete values are shown in the supplementary material. All messages are gateway messages, and they are sent strictly periodically and share a common released time without queuing jitters. We verified that four message sets are scheduled by using the SV algorithm (see supplementary material); the hyper period is 1 000 000 μs for each message set, and the numbers of released instants of message-processing jobs in this hyper period are 509, 468, 614, and 571. The system should be extended to a large-scale CAN cluster with eight subsystems; thus, we used the setting in Table VI for each subsystem. We implemented a tool using Java on a standard desktop computer (2.6 GHz Intel CPU and 4 GB memory) to calculate the WCRT of each message-processing task. We also implemented a CAN cluster gateway based on the presented architecture with a Freescale MPC560xB microcontroller, and measurement on the real gateway showed that the execution time of each message-processing task is 40 μs (i.e., 1 time unit). Therefore, the time values in the experiment appears are based on 40 μs of the time unit. The blocking caused by nonmessage-processing tasks is 120 μs in the platform.

Four methods, namely, GSRS, global scheduled exhaustive exploration (GSEE), partitioned scheduled round search (PSRS), and partitioned scheduled exhaustive exploration (PSEE), are used for comparison.

1) GSRS is implemented by combining the MRSLR (Algorithm 1) and MRSUR (Algorithm 2) algorithms to determine LBRT and UBRT.
2) GSEE is the exhausting exploration based on global scheduling.
3) PSRS is implemented by partition using the SFFD algorithm (Algorithm 3); then the MRSLR (Algorithm 1) and MRSUR algorithms (Algorithm 2) are combined to determine LBRT and UBRT on each core.
4) PSEE is the exhausting exploration based on partitioned scheduling.

Considering that the execution time of each message-processing task is 1 time unit in the gateway, the results obtained using GSEE and PSEE are the exact WCRTs.

B. Experimental Results

Experiment 1: This experiment aims to verify the correctness of the proposed methods. Thus, we conducted this experiment with the small-scale simulated message set of Table I (the motivating example) on eight subsystems and a dual-core CPU. Table VII shows the results of nine low-priority message-processing jobs. The following observations are drawn.

1) The computed WCRTs using GSEE are always between LBRTs and UBRTs using GSRS, and the computed WCRTs using PSEE are always between LBRTs and UBRTs using PSRS. Although we only list nine jobs, the other jobs demonstrate the same regular pattern as these jobs. The results verify that both GSRS and PSRS can obtain safe UBRT using individual scheduling types.
2) The LBRTs and UBRTs using the PSRS are less than or equal to, respectively, those using GSRS. Specifically, PSRS generates tighter LBRTs and UBRTs than GSRS for four low-priority message-processing jobs ($\tau_{31,1}$, $\tau_{31,2}$, $\tau_{30,2}$, and $\tau_{29,2}$), whereas other jobs show the same results. The preliminary results show that partitioned scheduling can obtain tighter WCRTs than global scheduling for low-priority message-processing jobs.

Experiment 2: This experiment compares GSRS with PSRS on the large-scale CAN cluster with a dual-core CPU and eight subsystems (Table VI). Eight subsystems exist in the CAN cluster, and each subsystem contains 509, 509, 468, 468, 614, 614, 571, and 571 release instants. The total number of combinations for the lowest priority task m_{789} should be $509^2 \times 468^2 \times 614^2 \times 571^2 = 6\,974\,871\,800\,400\,943\,493\,184$ using GSRS. Moreover, each combination should consume about 1.25 ms with eight subsystems. Therefore, the total computation time should reach 16 587 880 043 years, and the exact WCRT cannot be obtained using GSEE. All subsystems are divided into two groups, namely, $g_1 = \{s_1, s_2, s_3, s_8\}$ and $g_2 = \{s_4, s_5, s_6, s_7\}$, using the SFFD algorithm. The message-processing tasks are assigned to u_1 and u_2, respectively. The total number of combinations for m_{789} is $509^2 \times 468 \times 571 = 69\,233\,697\,468$ and each combination should consume approximately 0.53 ms with four subsystems. The total computation time should be about 424 days using PSEE. Considering the aforementioned situations, this experiment only shows the results using GSRS and the PSRS, as shown in Table VIII. The following observations are drawn.

1) The PSRS generates tighter WCRTs than GSRS for the message-processing tasks triggered from s_4, s_5, s_6, and s_8 (account for 67.7% in the total number of 616), larger WCRTs for the message-processing tasks triggered from s_2 and s_3 (account for 21.75%), and equal WCRTs for the message-processing tasks triggered from s_1 and s_7 (account for 10.55%). Specifically, PSRS can

TABLE VIII
RESPONSE TIMES WITH A DUAL-CORE CPU

Subsystem	Task	GSRS		PSRS	
		R^-	R^+	R^-	R^+
s_8	$\tau_{701} - \tau_{789}$	880	1080	640	640
s_7	$\tau_{601} - \tau_{689}$	640	640	640	640
s_6	$\tau_{501} - \tau_{585}$	440	440	400	400
s_5	$\tau_{401} - \tau_{485}$	400	400	240	240
s_4	$\tau_{301} - \tau_{369}$	280	280	160	160
s_3	$\tau_{201} - \tau_{269}$	240	240	400	400
s_2	$\tau_{101} - \tau_{165}$	200	200	240	240
s_1	$\tau_1 - \tau_{65}$	160	160	160	160

TABLE IX
RESPONSE TIMES WITH A FOUR-CORE CPU

Subsystem	$\tau_{i,k}$	GSRS		PSRS	
		R^-	R^+	R^-	R^+
s_8	$\tau_{701} - \tau_{789}$	280	280	240	240
s_7	$\tau_{601} - \tau_{689}$	280	280	240	240
s_6	$\tau_{501} - \tau_{585}$	240	240	240	240
s_5	$\tau_{401} - \tau_{485}$	240	240	240	240
s_4	$\tau_{301} - \tau_{369}$	200	200	160	160
s_3	$\tau_{201} - \tau_{269}$	200	200	160	160
s_2	$\tau_{101} - \tau_{165}$	160	160	160	160
s_1	$\tau_1 - \tau_{65}$	160	160	160	160

reduce the WCRT by 440 μs (the reduced percentage reaches 40.74%) compared with GSRS for the message-processing tasks triggered from s_8. Hence, partitioned scheduling is better than global scheduling in reducing WCRTs in general.

2) The obtained LBRTs and UBRTs using PSRS are always equal. Considering that the exact WCRT belongs to the interval of LBRT and UBRT, the exact WCRTs can be directly obtained and are equal to corresponding LBRTs and UBRTs without exhaustive exploration when using PSRS. For GSRS, the LBRTs and UBRTs are also equal for the message-processing tasks triggered from s_1–s_7; however, for the low-priority message-processing tasks triggered from s_8, the LBRTs and UBRTs are no longer equal, and the differences are 200 μs. Considering that an exact WCRT cannot be obtained in limited time using GSEE, the exact WCRTs of low-priority messages cannot be obtained when global scheduling is employed. Furthermore, the approximate and pessimistic WCRT cannot capture the complexity of practical automotive systems, and it may lead to an unnecessarily conservative design [3]. Therefore, the partitioned scheduling is useful in reducing complexity and the unnecessarily conservative design of automotive systems because it can obtain a tighter WCRT.
3) The obtained WCRTs increase with the decreasing priority of messages using GSRS, whereas PSRS shows a different pattern. The reason for the difference is that the messages are partitioned to different cores using SFFD, which cannot guarantee integrity of priorities because partition is a bin packing problem (NP-hard). For example, the message-processing tasks triggered from s_4 are processed in the u_2, and tighter WCRTs are obtained than the message-processing tasks triggered from s_2 and s_3, which are processed in the u_1. Therefore, the use of PSRS includes both positive and negative effects. The positive effect is that the WCRTs of several low-priority message-processing tasks (triggered from s_4 and s_5) are considerably small, and the negative effect is that the WCRTs of several high-priority message-processing tasks (triggered from s_2 and s_3) are large. The timing constraints of these high-priority messages may be no longer valid in the gateway.
4) Another situation is that although s_1 and s_2 are assigned into the same group using SFFD, the WCRTs of message-processing tasks triggered from s_2 using GSRS are less than those using PSRS. The results indicate that GSRS is useful for several high-priority messages because GSRS using multiple cores can reduce the WCRTs more efficiently than PSRS using a single-core in approximately equal scales.
5) According to the experimental results, the difference between LBRTs and UBRTs in global scheduling is always larger than that in partitioned scheduling. The potential reason may be that partitioned scheduling is useful in capturing the complexity.

Experiment 3: Although PSRS can obtain exact WCRTs using the dual-core gateway (Table VIII of Experiment 2), the bottleneck in message-processing still exists in large-scale CAN clusters (e.g., the WCRTs for low-priority message-processing tasks triggered from s_8 are 1080 and 640 μs using GSRS and PSRS, respectively). To further eliminate the bottleneck, we increase the number of cores and consider the four-core gateway to observe the results. We still use the same message set as Experiment 2 in this experiment. All subsystems are divided into four groups, namely, $g_1 = \{s_1, s_7\}$, $g_2 = \{s_2, s_8\}$, $g_3 = \{s_3, s_5\}$, and $g_4 = \{s_4, s_6\}$, using the SFFD algorithm, and the message-processing tasks are assigned to u_1, u_2, u_3, and u_4, respectively. The results are shown in Table IX, and the following observations on the given real message sets and experimental results can be obtained.

1) In a four-core gateway, the LBRTs and UBRTs are always the same for all messages. The exact WCRT is equal to the LBRT and UBRT for each message. The four-core gateway can considerably reduce the WCRT. For example, the WCRT of low-priority message-processing tasks triggered from s_8 is reduced to 280 and 240 μs using GSRS and PSRS, respectively. Consequently, the reduced percentages on a four-core gateway reach 74.1% and 62.5%, respectively, compared with those on a dual-core gateway.
2) Considering that the minimum WCRT is 160 μs ($B_{\tau_{i,k}} + C_{\tau_{i,k}} = 120+40 = 160$) for the message-processing jobs triggered from s_1, these jobs cannot be interfered.
3) Considering that the WCRTs of message-processing jobs triggered from s_3 and s_4 using GSRS are 200 μs, these jobs could be interfered by at most one high-priority message-processing job before it is executed using global scheduling.
4) In partitioned scheduling, all message-processing jobs could be interfered by at most two high-priority message-processing jobs, and the interference is reduced to an acceptable range. Thus, the four-core gateway can ultimately eliminate the bottleneck in message transmission on large-scale CAN clusters.
5) Considering that PSRS can generate tighter WCRTs than GSRS for most message-processing tasks, partitioned

scheduling is still better than global scheduling when a four-core gateway is used.

C. Design Optimization Guide

Without loss of generality, many simulated message sets with the approximate equal parameter values and scales to the provided real message sets are tested and show the same regular pattern as Experiments 2 and 3. Considering space limitations, we do not provide detailed experimental results in this paper. Finally, we summarize the following multicore gateway design optimization guide on large-scale CAN clusters based on the experimental results.

1) Partitioned scheduling is better than global scheduling for dual- and four-core gateways in general. First, partitioned scheduling complies with the AUTOSAR specification. If all software developers and products comply with the AUTOSAR specification, then the gateway development efficiency can be improved by the AUTOSAR runtime environment (RTE) because AUTOSAR introduces the RTE to shield the details that are related to hardware such that the code portability in different developers is easy. Second, partitioned scheduling can significantly reduce more tightness in the WCRT of low-priority messages than global scheduling. Third, partitioned scheduling may obtain exact WCRT in the provided real message sets, thereby possibly avoiding the combinatorial explosion problem. Therefore, partitioned scheduling is useful in capturing the complexity and reducing an unnecessarily conservative design to reduce the design cost of practical automotive systems. Although partitioned scheduling may suffer from an unbalanced workload and unstable timing constraints of several high-priority messages, this shortcoming is less important than the above three advantages.
2) When a large-scale CAN cluster is configured with a dual-core gateway, the WCRTs obtained using partitioned scheduling is acceptable. The four-core gateway is a reasonable option in eliminating the bottleneck of the message-processing in the gateway. In this situation, all message-processing tasks are interfered by at most two high-priority message-processing tasks ($240\ \mu s - 160\ \mu s = 80\ \mu s$) using partitioned scheduling. Consequently, the message can go through the gateway immediately with a small message-processing delay.

VII. Conclusion

In this paper, we proposed two WCRT analysis methods for message-processing tasks based on global and partitioned scheduling paradigms, respectively, in the multicore automotive gateways and evaluate them by experiments. The correctness of the proposed analysis methods are verified by proofs. We conclude that partitioned scheduling is better than global scheduling for multicore automotive gateways with respect to the AUTOSAR specification, WCRT reduction, and timing precision. We also observed that a four-core gateway can eliminate the bottleneck of the message-processing with a small message-processing delay. The future work will be the WCRT analysis for general real-time tasks with multiple execution units by extending the proposed methods in this paper.

Acknowledgment

The authors would like to thank the associate editor and three anonymous reviewers whose constructive comments have helped to improve this paper.

References

[1] R. Obermaisser, C. El Salloum, B. Huber, and H. Kopetz, "From a federated to an integrated automotive architecture," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 7, pp. 956–965, Jul. 2009.

[2] M. D. Natale and A. Sangiovanni-Vincentelli, "Moving from federated to integrated architectures in automotive: The role of standards, methods and tools," *Proc. IEEE*, vol. 98, no. 4, pp. 603–620, Apr. 2010.

[3] N. Navet, S. Louvart, J. Villanueva, S. Campoy-Martinez, and J. Migge, "Timing verification of automotive communication architectures using quantile estimation," in *Proc. Eur. Congr. Embedded Real Time Softw. Syst.*, 2014, pp. 1–10.

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

[5] G. Xie *et al.*, "WCRT analysis of CAN messages in gateway-integrated in-vehicle networks," *IEEE Trans. Veh. Technol.*, vol. 66, no. 11, pp. 9623–9637, Nov. 2017.

[6] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (CAN) schedulability analysis: Refuted, revisited and revised," *Real Time Syst.*, vol. 35, no. 3, pp. 239–272, Apr. 2007.

[7] H. Zeng, M. Di Natale, P. Giusto, and A. Sangiovanni-Vincentelli, "Stochastic analysis of CAN-based real-time automotive systems," *IEEE Trans. Ind. Informat.*, vol. 5, no. 4, pp. 388–401, Nov. 2009.

[8] H. Zeng, M. D. Natale, P. Giusto, and A. Sangiovanni-Vincentelli, "Using statistical methods to compute the probability distribution of message response time in controller area network," *IEEE Trans. Ind. Informat.*, vol. 6, no. 4, pp. 678–691, Nov. 2010.

[9] J. H. Kim *et al.*, "Gateway framework for in-vehicle networks based on CAN, FlexRay, and Ethernet," *IEEE Trans. Veh. Technol.*, vol. 64, no. 10, pp. 4472–4486, Oct. 2015.

[10] E. Azketa *et al.*, "Schedulability analysis of multi-packet messages in segmented CAN," in *Proc. IEEE 17th Conf. Emerg. Technol. Factory Autom.*, 2012, pp. 1–8.

[11] J. Sommer and R. Blind, "Optimized resource dimensioning in an embedded CAN-CAN gateway," in *Proc. IEEE Int. Symp. Ind. Embedded Syst.*, 2007, pp. 55–62.

[12] S. Lee, D. H. Lee, M. H. Kim, and K. C. Lee, "Traffic-balancing algorithm for can systems with dual communication channels to enhance the network capacity," *Int. J. Automotive Technol.*, vol. 11, no. 4, pp. 525–531, Aug. 2010.

[13] M. Sojka, P. Píša, O. Špinka, and Z. Hanzálek, "Measurement automation and result processing in timing analysis of a Linux-based CAN-to-CAN gateway," in *Proc. 6th IEEE Int. Conf. Intell. Data Acquisition Adv. Comput. Syst.*, vol. 2. Prague, Czech Republic, 2011, pp. 963–968.

[14] G. Han, H. Zeng, M. Di Natale, X. Liu, and W. Dou, "Experimental evaluation and selection of data consistency mechanisms for hard real-time applications on multicore platforms," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 903–918, May 2014.

[15] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *Proc. IEEE Int. Real Time Syst. Symp. (RTSS)*, 2007, pp. 149–160.

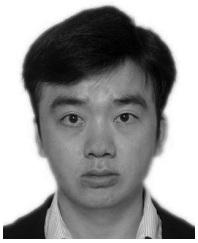
[16] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 4, pp. 553–566, Apr. 2009.

[17] N. Guan, M. Stigge, W. Yi, and G. Yu, "New response time bounds for fixed priority multiprocessor scheduling," in *Proc. 30th IEEE Real Time Syst. Symp.*, Washington, DC, USA, 2009, pp. 387–397.

[18] Q. Zhou, G. Li, and J. Li, "Improved carry-in workload estimation for global multiprocessor scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2527–2538, Sep. 2017.

[19] E. G. Schmidt, M. Alkan, K. Schmidt, E. Yürüklü, and U. Karakaya, "Performance evaluation of FlexRay/CAN networks interconnected by a gateway," in *Proc. IEEE Int. Symp. Ind. Embedded Syst.*, 2010, pp. 209–212.

[20] T. Ishigooka and F. Narisawa, "Message packing algorithm for CAN-based legacy control systems mixed with CAN and FlexRay," *SAE Int. J. Passenger Cars Electron. Elect. Syst.*, vol. 3, no. 1, pp. 88–97, Apr. 2010.
[21] A. Kern, D. Reinhard, T. Streichert, and J. Teich, "Gateway strategies for embedding of automotive CAN-frames into Ethernet-packets and vice versa," in *Proc. Archit. Comput. Syst.*, 2011, pp. 259–270.
[22] A. A. Nacer, K. Jaffres-Runser, J.-L. Scharbarg, and C. Fraboul, "Strategies for the interconnection of CAN buses through an Ethernet switch," in *Proc. 8th IEEE Int. Symp. Ind. Embedded Syst.*, 2013, pp. 77–80.
[23] F. C. Brăescu, L. Ferariu, and A. Nacu, "OSEK-based gateway algorithm for multi-domain CAN systems," in *Proc. IEEE Int. Conf. Intell. Comput. Commun. Process.*, 2011, pp. 423–428.
[24] S.-H. Seo, J.-H. Kim, S.-H. Hwang, K. H. Kwon, and J. W. Jeon, "A reliable gateway for in-vehicle networks based on LIN, CAN, and FlexRay," *ACM Trans. Embedded Comput. Syst.*, vol. 11, no. 1, p. 7, Mar. 2012.
[25] R. I. Davis, S. Kollmann, V. Pollex, and F. Slomka, "Schedulability analysis for controller area network (CAN) with FIFO queues priority queues and gateways," *Real Time Syst.*, vol. 49, no. 1, pp. 73–116, Jan. 2013.
[26] C. Bays, "A comparison of next-fit, first-fit, and best-fit," *Commun. ACM*, vol. 20, no. 3, pp. 191–192, Mar. 1977.
[27] J. Taube, F. Hartwich, and H. Beikirch, "Comparison of CAN gateway modules for automotive and industrial control applications," in *Proc. 10th Int. CAN Conf.*, 2005, pp. 1–8.
[28] T. Herpel, "Performance evaluation of time-critical data transmission in automotive communication systems," Ph.D. dissertation, Faculty Eng., Univ. Erlangen-Nuremberg, Erlangen, Germany, 2009.
[29] *Classic Platform*. Accessed: Mar. 2018. [Online]. Available: https://www.autosar.org/standards/classic-platform/
[30] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.



Guoqi Xie (M'15) received the Ph.D. degree in computer science and engineering from Hunan University, Changsha, China, in 2014.

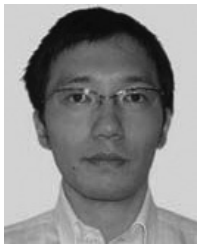
He is an Associate Professor of computer science and engineering with Hunan University. He was a Post-Doctoral Researcher with Nagoya University, Nagoya, Japan, from 2014 to 2015, and with Hunan University, from 2015 to 2017. His current research interests include embedded and cyber-physical systems, parallel and distributed systems, industrial and systems engineering.

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

Gang Zeng (M'03) received the Ph.D. degree in information science from Chiba University, Chiba, Japan, in 2006.

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

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



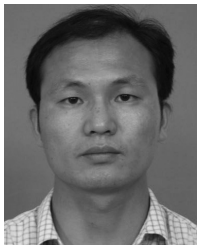
Ryo Kurachi received the Ph.D. degree in information science from Nagoya University, Nagoya, Japan, in 2012.

He is an Associate Professor with the Graduate School of Information Science, Nagoya University. From 2000 to 2006, he was with AISIN AW Company, Ltd., Kariya, Japan. Since 2007, he has been a Researcher with the Center for Embedded Computing Systems, Nagoya University. His current research interests include in-vehicle networks and real-time scheduling theory.

Hiroaki Takada (M'97) received the Ph.D. degree in information science from the University of Tokyo, Tokyo, Japan, in 1996.

He is a Professor with the Graduate School of Information Science, Nagoya University, Nagoya, Japan, where he is also the Executive Director of the Center for Embedded Computing Systems. He was a Research Associate with the University of Tokyo from 1989 to 1997, and was a Lecturer and then an Associate Professor with the Toyohashi University of Technology, Toyohashi, Japan, from 1997 to 2003. His current research interests include real-time operating systems, real-time scheduling theory, and embedded system design.

Dr. Takada is a member of the Information Processing Society of Japan, Information Processing Society of Japan, Institute of Electronics, Information and Communication Engineers, Japan Society for Software Science and Technology, and Society of Automotive Engineers of Japan.



Zhetao Li (M'17) received the Ph.D. degree in computer science from Hunan University, Changsha, China, in 2010.

He is a Professor at Xiangtan University, China. His current research interests include in-vehicle networks and Internet of Things (IoT).

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

He is a Professor of computer science and electronic engineering with Hunan University, Changsha, China. He is the Director of the Key Laboratory for Embedded and Network Computing of Hunan Province, China. His current research interests include computer architectures, embedded computing systems, cyber-physical systems, and Internet of Things.

Dr. Li is a member of the council of China Computer Federation and a Senior Member of Information Processing Society of Japan.

Keqin Li (M'90–SM'96–F'15) received the Ph.D. degree in computer science from the University of Houston, Houston, TX, USA, in 1990.

He is a SUNY Distinguished Professor of computer science with the State University of New York, New Paltz, NY, USA. He has published over 520 journal articles, book chapters, and refereed conference papers. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of Things, and cyber-physical systems.

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