Electronic Preprint for Journal of Information Processing Vol.24 No.5

Regular Paper

Reservation-Based Scheduling for Automotive DSMS under High Overload Condition

Jaeyong Rho[1,a)] Takuya Azumi[2,b)] Akihiro Yamaguchi[3,c)] Kenya Sato[4,d)]
Nobuhiko Nishio[1,e)]

Received: November 18, 2015, Accepted: May 17, 2016

Abstract: Recent automotive systems require various data, including data from on-board sensors and external sources to recognize environmental conditions. As the amount of sensor data used in automotive systems increases, processes that use such data become increasingly complicated. In addition, similar data processing can be duplicated over multiple applications. To address these issues, a data stream management system (DSMS) for automotive systems based on a data integration architecture has been developed. However, hard real-time deadlines cannot be guaranteed due to unpredictable load changes caused by data streams. For example, the arrival time and CPU utilization requested by data streams from vehicle-to-vehicle communications change rapidly depending on environmental conditions. We propose the reservation-based operator path earliest deadline first (ROP-EDF) scheduling algorithm for an automotive DSMS under overload conditions. The proposed algorithm reserves processor time preferentially for hard real-time tasks so that tasks can meet deadlines under overload conditions. ROP-EDF can be used for load testing on a single processor system. Experimental results show the effectiveness of the proposed algorithm compared with existing algorithms relative to the deadline miss ratio under overload conditions.

Keywords: real-time scheduling, automotive DSMS, overload, earliest deadline first, load testing

1. Introduction

Recently, many automotive applications, such as pre-crash safety, adaptive cruise control, and lane keeping assist systems, have emerged. In such applications, several types of sensor data, such as camera images, radar data, and acceleration data, are required. In addition, data from other vehicles and road sensors are required to improve prediction accuracy. Consequently, the total amount of required data is significant. In current automotive systems, data are processed and managed individually in electronic control units (ECU). Thus, processes that use the data can be duplicated over multiple applications in ECUs. In addition, the associated software development costs increase as the amount of data increases. Automotive data stream management systems (DSMS) [1], [2], [3] have been developed to address these issues.

A DSMS is suitable for real-time processing and management of rapidly generated continuous data streams (e.g., sensor data) with low latency. Most DSMS, including Borealis [4], STREAM [5], and TelegraphCQ [6], are designed to run on general-purpose computers. Research into operator scheduling [7], e.g., chain scheduling [8] and scheduling for a wide-area network [9], has been conducted. Such scheduling algorithms primarily target streaming applications, such as network monitoring, financial data analysis, and sensor networks, to minimize memory usage or network latency and maximize throughput.

In contrast to existing scheduling algorithms for generic DSMS, strict real-time constraints are required for the automotive DSMS. The automotive DSMS processes and combines data streams from on-board sensors and external sources, and must provide data to multiple applications within timing constraints called deadlines. For example, for adaptive cruise control and collision avoidance systems, computing the distance to another vehicle from several sensors' information and applying the brakes automatically when the distance is less than a predetermined threshold must be completed within strict timing constraints. The deadline in safety-related applications must be met to avoid serious car accidents. Such processing is called a hard real-time task, and it must execute within a deadline.

Existing real-time scheduling algorithms in DSMS cannot be applied directly to the automotive DSMS because hard real-time task deadlines will be missed under overload conditions. Several studies have adopted real-time scheduling in DSMS to meet timing constraints over data streams [10], [11], [12], [13]. Li et al. [10] proposed operator path earliest deadline first (OP-EDF) scheduling to handle huge amounts of sensor data and satisfy real-time requirements. They define an OP that comprises operators and stream queues and treat it as a task. The objective of OP-EDF is to minimize the deadline miss ratio (DMR). Schmidt et al. [12], Wei et al. [11], and Ma et al. [13] proposed algorithms that can handle hard real-time tasks when systems are not over-

1 Ritsumeikan University, Kusatsu, Shiga 525–8577, Japan
2 Osaka University, Toyonaka, Osaka 560–8531, Japan
3 Nagoya University, Nagoya, Aichi 464–8601, Japan
4 Doshisha University, Kyoto 602–8580, Japan
a) no@ubi.cs.ritsumei.ac.jp
b) takuya@sys.es.osaka-u.ac.jp
c) yamagut@ertl.jp
d) ksato@mail.doshisha.ac.jp
e) nishio@cs.ritsumei.ac.jp

loaded. However, the algorithm proposed by Schmidt et al. only handles hard real-time tasks with predictable arrival times and may miss deadlines in an unpredictable manner under overload conditions. The algorithms proposed by Wei et al. and Ma et al. only concentrate on minimizing the DMR, tuple value loss ratio, and load shedding overhead when a system is overloaded.

Developers of the automotive DSMS must perform load testing to confirm system performance because such systems must perform reliably under overload conditions. Vehicle-to-vehicle (V2V) communication provides accurate detection of potentially dangerous situations that on-board sensors alone cannot detect [14]. Thus, by exchanging data between vehicles, V2V communication can improve safety [15], [16]. The volume and arrival time of data streams from V2V communication can change rapidly depending on environmental conditions, e.g., many V2V data streams will be generated at a congested urban intersection. The European Telecommunications Standards Institute specification for collision risk warning applications using V2V communications requires a vehicle to receive and process at least 1,000 data packets (e.g., position and speed) per second [17]. The large volumes of V2V data streams require significant processing time, which can interfere with hard real-time task requirements; thus, hard deadlines cannot always by guaranteed.

The previously proposed ER-EDF [18] guarantees that such deadlines are met under overload conditions. ER-EDF provides reserved processor time for soft and hard real-time tasks. However, it is only valid when the requested CPU utilization is less than or equal to 100%. Therefore, adopting ER-EDF in automotive systems in which requested CPU utilization can exceed 100% is difficult. Generally, automotive systems have low-speed and low-capacity memory, as well as low CPU performance due to strict cost limitations. Therefore, systems can become overloaded quite quickly when the amount of V2V data increases beyond capacity.

Contribution: This paper presents a reservation-based OP-EDF (ROP-EDF) for the automotive DSMS. ROP-EDF consists of two admissions and scheduling algorithms and can handle hard real-time tasks under high overload conditions. ROP-EDF can perform load testing on single processor systems to test real-time constraints, and satisfy hard deadlines under overload conditions. The main contribution of this study is as follows.

- We address the problem with OP-EDF, i.e., hard real-time tasks can miss deadlines due to transient overload. Simulation results show significant improvement in terms of deadline miss ratio in comparison with OP-EDF under overload conditions.
- We resolve ER-EDF limitations, i.e., hard real-time task deadlines cannot be guaranteed when requested CPU utilization exceeds 100%. Simulation results performed in comparison with ER-EDF demonstrate that ROP-EDF does not allow deadline misses when the total requested CPU utilization exceeds 100%.

Organization: The remainder of this paper is organized as follows. Section 2 presents the system and task models. Problem definitions are given in Section 3. Section 4 presents the proposed algorithm. Section 5 evaluates the effectiveness of the proposed algorithm. Related work is discussed in Section 6, and we conclude the paper in Section 7.

2. Preliminaries

Here, we describe stream processing, an automotive DSMS using an eDSMS [1], [3], and our task model.

2.1 Basic Concept of Stream Processing

Stream processing systems, such as a DSMS, are designed to handle high-volume and bursty data streams. DSMS typically target applications such as on-line financial analysis, network monitoring, and sensor networks that require continuous data stream processing. A **data stream** is defined as an ordered sequence of data that is generated continuously in large volumes with unpredictable data arrival rate.

A **query** that describes the data stream processing flow using multiple operators is registered with the system. The query is composed of one or more operators, and two consecutive operators are connected by a **stream queue**. The query processes operators continuously as new data stream arrives. An **operator** represents a particular algorithmic transformation of an input tuple to an output tuple. In **Fig. 1**, the circles and the edges between the circles in the query file indicate operators and stream queues, respectively. A **tuple** is a set of data values (e.g., vehicle ID, vehicle speed, and vehicle position). In stream processing, a precedent operator executes a tuple from an input stream queue and produces one or more result tuples. The result tuples are stored in an output stream queue. The data streams in the output stream queue are delivered to a subsequent operator or an application.

2.2 Data Integration Architecture Based on DSMS

Figure 2 shows the current architecture for an automotive system. In the current architecture, data retrieved from on-board sensors and external sources are processed individually by each application program embedded in ECUs because automotive suppliers provide a product as a set, i.e., the software and the related sensor. Therefore, similar data processing can be duplicated over multiple applications in different ECUs. Furthermore, if system

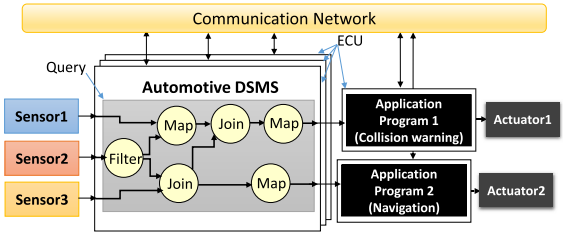

Fig. 1 Data integration architecture for an automotive system.

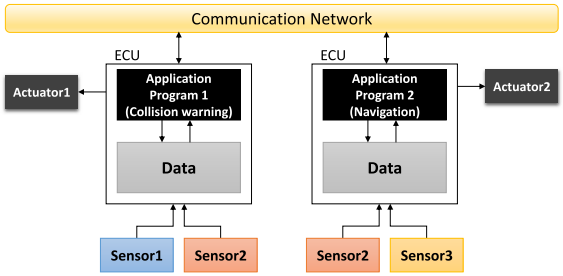

Fig. 2 Current architecture for an automotive system.

properties (e.g., sensor type) are changed, large parts of the application programs must be modified.

To tackle these issues, an automotive DSMS based on a data integration architecture has been developed [1], [3] (Fig. 1). Applications can be separated from sensor devices because data processing using various sensors is defined in the automotive DSMS. Data processed by the automotive DSMS can be accessible in a location-transparent manner from multiple applications. In addition, changing system properties becomes easier than the current architecture since application programs are independent from specific sensors; thus, automotive software development costs can be reduced.

Operators, such as Filter, Map, Aggregate, and Join, used in a query for an automotive DSMS are based on the general-purpose Borealis DSMS [4]. **Filter** filters an input data stream according to a static condition and outputs more than one filtered data stream. The **Map** operator applies a function to an input data stream to obtain the corresponding output data stream, and the **Aggregate** operator outputs a group of data streams that have the same attribute among multiple data streams. The **Join** operator combines input data streams from multiple sources into one data stream according to given conditions.

2.3 Task Model

In an automotive DSMS, two types of tasks can coexist simultaneously, i.e., a set of m hard real-time tasks $\{OP_{H1}, OP_{H2}, \ldots, OP_{Hm}\}$ and n soft real-time tasks $\{OP_{S1}, OP_{S2}, \ldots, OP_{Sn}\}$ [*1]. For example, computing the distance to another vehicle for adaptive cruise control and collision avoidance systems can be a hard real-time task. This type of task must finish its execution within a deadline even under overload conditions. Missing deadlines can result in large negative values and may cause a collision.

For soft real-time tasks, minimizing the delay is still valuable for the system to complete the execution of tasks even if they miss their deadlines. Results produced after deadlines can still valuable for the system, though the values may decrease with time. For example, in a vehicle-infrastructure cooperative right-turn collision caution application, audio and visual alerts are triggered if a driver takes their foot off the brake when they are waiting to make a right turn and another vehicle approaches from the opposite direction [20]. A task used in this system has a soft deadline because it may not negatively affect the system if the soft deadline is missed. Even if a soft deadline is missed, we can still provide safety to drivers and prevent service quality from degrading severely by completing the execution of this type of tasks as quickly as possible rather than aborting the task execution.

In this paper, we consider an OP (dotted line, **Fig. 3**) as a single task. An OP comprises one or more operators (circles, Fig. 3) and queues (edges, Fig. 3) between two consecutive operators. When a tuple enters a unique OP, it is processed by operators in sequence. The i-th task OP_{ti} is expressed as follows.

$$< C_{ti}, a_{ti}, D_{ti}, d_{ti} > \quad (1)$$

[*1] In this paper, we use definitions of hard and soft real-time that are described in Ref. [19].

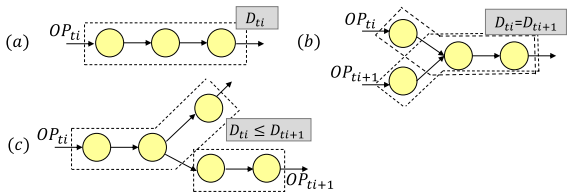

Fig. 3 Methods for allocating a task using OP (operator path).

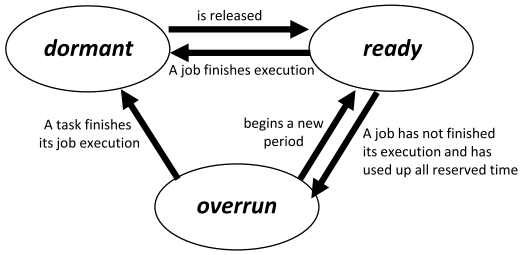

Fig. 4 Real-time task states.

$$C_{ti} = \sum_{z=1}^{n} c_{tiz} \quad (2)$$

Here $i \geq 1$, $\boldsymbol{t}$ represents the task type where $\boldsymbol{H}$ denotes hard real-time tasks and $\boldsymbol{S}$ denotes soft real-time tasks, and $\boldsymbol{C_{ti}}$ denotes the execution time of task OP_{ti}. Assume OP_{ti} consists of n operators, and the execution time of the j-th operator in OP_{ti} with a tuple is $\boldsymbol{c_{tij}}$. $\boldsymbol{a_{ti}}$ denotes the time at which an input tuple arrives at OP_{ti}, and $\boldsymbol{d_{ti}}$ is the absolute deadline of OP_{ti}. We assume that the arrival time of the input tuple is the same as the release time of the task OP.

$\boldsymbol{J_{ti}^{l}}$ represents a job that is the l-th instance of task OP_{ti}. Each job of task OP_{ti} has the same relative deadline as OP_{ti}'s relative deadline $\boldsymbol{D_{ti}}$. The absolute deadline of task OP_{ti} can be calculated as $a_{ti} + D_{ti}$. All tasks OP_{ti} have relative deadlines that are less than or equal to their periods. Furthermore, we assume that the first job of task OP_{ti}, except for aperiodic tasks, is released at time 0 when we refer to a sequence of jobs.

Methods to allocate a task OP with more than one operator and stream queue have three patterns (Fig. 3). A set of sequence operators (each operator has a single input and output stream queue) is defined as one task OP (Fig. 3 (a)). Figure 3 (b) shows the method for allocating a task for an operator that combines and fuses more than two input data streams. An operator connected to more than two input stream queues processes when an input data stream arrives and places its result into an output-stream queue. Each task OP has the same relative deadline. Figure 3 (c) shows the method for allocating a task for an operator with more than two output stream queues. Here, OP_{ti} with a minimum relative deadline (i.e., a key path) is executed first, and other branch paths (e.g., OP_{ti+1}, Fig. 3 (c)) are executed in the order of their deadlines.

In the task model, real-time tasks OP_{ti} can be in one of three states: *dormant*, *ready*, or *overrun* (**Fig. 4**). When a task is created, its default state is *dormant*. A *dormant* task goes into the *ready* state at the task release time. When the execution of a *ready* task is not finished and exceeds the reserved time, it enters the *overrun* state. When a *ready* task completes execution, it goes to the *dormant* state and waits to begin a new task period.

3. Problem Statement

In an automotive DSMS, the number of received data streams from on-board sensors (e.g., a radar sensor) and external communications changes depending on environmental conditions. Thus, the CPU utilization requested by these data streams can increase rapidly and may exceed processor capacity. However, due to high requested CPU utilization caused by a large number of data streams, deadlines for hard real-time tasks cannot always be satisfied. To guarantee the deadlines for hard real-time tasks (e.g., a late response to stop a car may cause an accident), we must address the problems of OP-EDF [10] and ER-EDF [18]. The proposed algorithm primarily targets the overload situation caused by a high volume of data streams and is intended to perform load testing.

The EDF algorithm has been proven to be optimal and can schedule tasks with CPU utilization up to 100%. When the system is overloaded (i.e., total CPU utilization > 100%), tasks scheduled using EDF may miss deadlines unpredictably. OP-EDF can cause hard real-time tasks to miss deadlines if overload occurs; thus, we must consider the range of overload conditions for OP-EDF to be applicable in an automotive DSMS.

The ER-EDF algorithm [18] was proposed to guarantee that deadlines are satisfied for hard real-time tasks under overload conditions. However, it is only valid when the total requested CPU utilization is less than or equal to 100%. In ER-EDF, the requested CPU utilization of a task represents the peak CPU utilization for hard real-time tasks and the average CPU utilization for soft real-time tasks. Thus, the total requested CPU utilization is computed by $\sum_{i=1}^{m} \psi(OP_{Hi}) + \sum_{j=1}^{n} \theta(OP_{Sj})$, where the number of hard and soft real-time tasks is m and n, respectively. Here, $\boldsymbol{\psi}$ represents the peak CPU utilization of OP_{ti}, i.e., the maximum CPU utilization among all OP_{ti} jobs, and $\boldsymbol{\theta}$ denotes the average CPU utilization of OP_{ti}, which is computed by $\theta(OP_{ti}) = \left(\sum_{k=1}^{l} U(J_{ti}^{k})\right)/l$ with l jobs. CPU utilization $\boldsymbol{U}$ of the k-th job of OP_{ti} is $U(J_{ti}^{k}) = \mathit{Processing\ time}(C)/\mathit{Relative\ deadline}(D)$. Under ER-EDF admission control, soft and hard real-time tasks share a common capacity to reserve processor time. Therefore, hard real-time tasks can fail to reserve processor time due to insufficient capacity.

We show an example in which a hard real-time task fails to reserve processor time because soft real-time tasks reserve the processor time (in **Fig. 5**). In this example, there is a task set with one hard real-time task OP_{H1} and three soft real-time tasks OP_{S1}, OP_{S2}, and $OP_{S3}(A)$, as shown in **Table 1**. Here $OP_{S3}(A)$ is an aperiodic task activated by external sources such as V2V communication. The CPU utilization U of OP_{ti} jobs can change in a range; e.g., the utilizations of OP_{S3} jobs range from 10% to 50%. In Table 1, the system is under overload condition, i.e., $\psi(OP_{H1}) + \sum_{j=1}^{n} \psi(OP_{Sj}) = 145\%$, and the total requested CPU utilization is 105%.

If there is no *ready* task and only an *overrun* task exists, then reserved processor time $\mathfrak{R}$ of the *overrun* task will be replenished, and the task will be processed again. As can be seen in Fig. 5, when the CPU utilization of J_{S1}^{1} and J_{S2}^{1} are 40% and 30%, respectively, the reserved processor time $\mathfrak{R}$ of OP_{S1} and

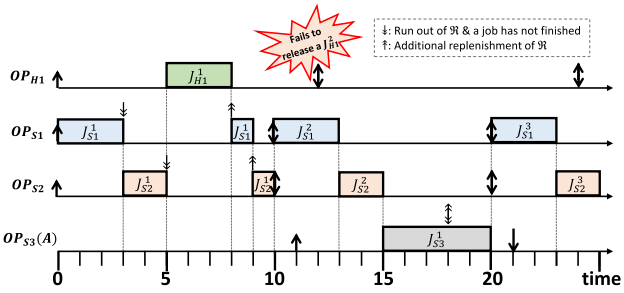

Fig. 5 Restrictive reservation in ER-EDF.

Table 1 Task set parameters.

TASK (OP_{ti})	OP_{H1}	OP_{S1}	OP_{S2}	$OP_{S3}(A)$
Period (T)	12 ms	10 ms	10 ms	—
Relative deadline (D)	12 ms	10 ms	10 ms	10 ms
CPU util. of jobs (U)	15%–25%	20%–40%	10%–30%	10%–50%
Requested CPU util.	25%	30%	20%	30%
Average CPU util. (θ)	20%	30%	20%	30%
Peak CPU util. (ψ)	25%	40%	30%	50%

OP_{S2} will be replenished at time 8 and 9, respectively. At time 10, OP_{S1} reserves processor time $\mathfrak{R}(OP_{S1}) = 10{\times}0.3 = 3$ ($D{\times}\theta$) because it has the shortest relative deadline among all OPs. We assume that the processing time of J_{S1}^{2} is $10 \times 0.3 = 3$ ($D \times U$). Then, $\mathfrak{R}(OP_{S2})$ is $10 \times 0.2 = 2$, and we assume that execution time of J_{S2}^{2} is $10 \times 0.2 = 2$. At time 11, OP_{S3} reserves processor time $10{\times}0.3 = 3$ ($D{\times}\theta$), and execution time of J_{S3}^{1} is $10{\times}0.5 = 5$. The remaining capacity is $1 - 0.3 - 0.2 - 0.3 = 0.2$. At time 12, OP_{H1} attempts to reserve processor time $12 \times 0.25 = 3$ ($D \times \psi$); however, there is insufficient capacity for OP_{H1} ($0.25 > 0.2$). Thus, J_{H1}^{2} fails to reserve processor time and is rejected.

4. ROP-EDF Algorithm

The proposed ROP-EDF algorithm, which is an improved form of the OP-EDF and ER-EDF algorithms, is suitable for an automotive DSMS. ROP-EDF consists of admission control and scheduling algorithms. To guarantee that hard real-time task deadlines are satisfied under overload conditions, ROP-EDF employs a strategy to reserve processor time based on task type (i.e., hard or soft real-time tasks). The proposed ROP-EDF is based on ER-EDF [18], which is an enhanced version of R-EDF [21].

We propose two following admission policies: i) policy 1 allocates reserved processor time $\mathfrak{R}$ according to the CPU utilization of a job for soft real-time tasks; ii) policy 2 allocates a proportional distribution of reserved processor time $\mathfrak{R}$ to soft real-time tasks.

Here, the total processor capacity is defined as 1. The average CPU utilization of a task is expressed as θ, and the peak CPU utilization of the task is expressed as ψ. The capacity of hard real-time tasks $\boldsymbol{C_{SUMH}}$ denotes summation of the peak CPU utilizations of hard tasks, and $\boldsymbol{C_{SUMS}}$ denotes summation of the average CPU utilizations of soft tasks. $\boldsymbol{C_S}$ represents the amount of usable CPU utilization for soft tasks. The peak real-time capacity $\boldsymbol{PC_{SUMRT}}$ indicates the total peak CPU utilization of all real-time tasks.

The number of hard and soft real-time tasks is denoted m and n, respectively. Here, these capacities can be computed by $C_{SUMH} = \sum_{i=1}^{m} \psi(OP_{Hi})$, $C_{SUMS} = \sum_{i=1}^{n} \theta(OP_{Si})$, $C_S = 1 - C_{SUMH}$, and

Algorithm 1 Admission Control (policy 1)

STEP 1: Initialize $C_{SUMH} \leftarrow \sum_{i=1}^{m} \psi(OP_{Hi})$, $C_S \leftarrow 1 - C_{SUMH}$, $PC_{SUMRT} \leftarrow 0$
STEP 2: Sort tasks by ascending absolute deadline and the deadline miss ratio
if a task is a hard real-time task **then**
  if $C_{SUMH} - \psi \geq 0$ **then**
    $C_{SUMH} \leftarrow C_{SUMH} - \psi$
    $PC_{SUMRT} \leftarrow PC_{SUMRT} + \psi$
  end if
else
  if $C_S - U \geq \alpha$ **then**
    $C_S \leftarrow C_S - U$
    $PC_{SUMRT} \leftarrow PC_{SUMRT} + \psi$
  end if
end if
STEP 3: Return the reserved processor time of a task
if a task is a hard real-time task **then**
  $C_{SUMH} \leftarrow C_{SUMH} + \psi$
  $PC_{SUMRT} \leftarrow PC_{SUMRT} - \psi$
else
  $C_S \leftarrow C_S + U$
  $PC_{SUMRT} \leftarrow PC_{SUMRT} - \psi$
end if

Algorithm 2 Admission Control (policy 2)

STEP 1: Initialize $C_{SUMH} \leftarrow \sum_{i=1}^{m} \psi(OP_{Hi})$, $C_{SUMS} \leftarrow \sum_{i=1}^{n} \theta(OP_{Si})$, $C_S \leftarrow 1 - C_{SUMH}$, $PC_{SUMRT} \leftarrow 0$, and $RM \leftarrow C_S$.
STEP 2: Sort tasks by ascending absolute deadline
if a task is a hard real-time task **then**
  if $C_{SUMH} - \psi \geq 0$ **then**
    $C_{SUMH} \leftarrow C_{SUMH} - \psi$
    $PC_{SUMRT} \leftarrow PC_{SUMRT} + \psi$
  end if
else
  $\omega \leftarrow RM \times \frac{\theta}{C_{SUMS}}$
  if a job of the task has finished **then**
    if $C_S - \omega \geq \alpha$ **then**
      $C_S \leftarrow C_S - \omega$
      $PC_{SUMRT} \leftarrow PC_{SUMRT} + \psi$
    end if
  end if
end if
STEP 3: Return the reserved processor time of a task
if a task is a hard real-time task **then**
  $C_{SUMH} \leftarrow C_{SUMH} + \psi$
  $PC_{SUMRT} \leftarrow PC_{SUMRT} - \psi$
else
  $C_S \leftarrow C_S + \omega$
  $PC_{SUMRT} \leftarrow PC_{SUMRT} - \psi$
end if

$PC_{SUMRT} = \sum_{i=1}^{m} \psi(OP_{Hi}) + \sum_{i=1}^{n} \psi(OP_{Si})$, where $OP_{ti} (i \geq 1)$ is admitted to a processor. When $PC_{SUMRT} + \alpha > 1$, the system is considered to be in an overload condition, where $\boldsymbol{\alpha}$ [*2] denotes the overall overhead costs associated with context-switching and scheduling for all tasks. Otherwise, the system is in a non-overload condition (i.e., $PC_{SUMRT} + \alpha \leq 1$).

4.1 Admission Control

In ROP-EDF admission control, the capacities of hard and soft real-time tasks are initially computed, i.e., initialize PC_{SUMRT}, C_{SUMH}, C_S, RM, and C_{SUMS} (Algorithm 2 only), as shown in Algorithms 1 and 2.

At release time, all tasks must undergo a feasibility test prior to admission to ROP-EDF scheduling. Release time OP_{ti} is defined as the time at which data arrives into OP_{ti}. If more than one OP_{ti} is released in a given time unit, then the task with the shortest absolute deadline is tested first. Initially, the test determines whether the task is a hard or soft real-time task. Then, the scheduler checks resource availability. For example, OP_{Hi} can be allocated to a processor if $\psi(OP_{Hi}) \leq C_{SUMH}$. Otherwise, OP_{Hi} will fail to release a new job.

Two policies apply to a soft real-time task OP_{Si}. In policy 1, if $C_S \geq$ CPU utilization of job U, then OP_{Si} will be allocated to a processor and can release a job, as shown in Algorithm 1. Thus, only a job with CPU utilization less than the amount of usable CPU utilization for soft real-time tasks can reserve the processor time and be released. If resource contention occurs between two OP_{Si}s with the same release time, period, and absolute deadline, task starvation can be avoided by rejecting the task with the lower DMR. A scheduler must record deadline miss counts and satisfied deadline counts for each task to calculate the DMR. Therefore, processor time is reserved for a task with higher DMR.

In policy 2, the remaining processor time for soft real-time tasks C_S will be distributed proportionally to the average CPU utilization of the soft real-time tasks based on ω, as shown in Algorithm 2. To prevent the number of unfinished jobs from increasing, a new job cannot be released if unfinished jobs exist.

Upon task completion, the reserved processor time must be released. For a hard real-time task, C_{SUMH} increases by its peak CPU utilization in both policies. For a soft real-time task, C_S increases by the CPU utilization of job U in policy 1 and increases the amount of ω in policy 2. PC_{SUMRT} decreases by the peak CPU utilization in both policies and releases the reserved processor time.

4.2 Scheduling

The proposed scheduling algorithm is based on ER-EDF. The scheduler assumes that each real-time task releases jobs according to the period parameter and assigns $\mathfrak{R}$ to task OP_{ti} released through admission control. For a hard real-time task OP_{Hi}, processor time $\mathfrak{R}(OP_{Hi}) = D_{Hi} \times \psi$ is assigned to OP_{Hi}. The $\mathfrak{R}$ for a soft real-time task differs between policies 1 and 2. For policy 1, when processor time is reserved, processor time $\mathfrak{R}(OP_{Si}) = D_{ti} \times U$ is assigned to each soft real-time task OP_{Si}. Here, U is the CPU utilization of J_{Si}^{k} released by OP_{Si}. For policy 2, the scheduler assigns processor time $\mathfrak{R}(OP_{Si}) = D_{ti} \times \omega$ to a soft real-time task OP_{Si}.

The proposed ROP-EDF algorithm protects against overload conditions; otherwise, tasks are scheduled according to the EDF strategy. If a task requires additional execution time after using

[*2] Scheduling and context-switching overheads used in this paper are explained in Ref. [22]. In this paper, we assume that the worst-case execution time of each task includes cache-related preemption delay and assume the case of all cache assesses miss since we must consider the worst-case scenario.

Algorithm 3 Scheduling

```
1: STEP 1: Select a task for execution
2: if any real-time task is in the ready state then
3:    Select a task whose latest released job has the earliest deadline and
      execute its job
4: else if there is a task in the overrun state then
5:    Select a task in the overrun state whose latest released job has the ear-
      liest deadline, put it in the ready state and execute its job
6: end if
7: STEP 2: The scheduler waits until the next time unit
8: if a running task finishes all its jobs then
9:    It enters the dormant state
10: else if the system is overloaded (PC_SUMRT + α > 1) and the current task
    does not finished its execution and the task used all its ℜ then
11:   if any real-time task is in the ready state then
12:      Current task enters the overrun state
13:   else if CPU utilization of the current task is greater than or equal to
      (1 − α) then
14:      Its state is switched into overrun state
15:   else
16:      It continues to execute
17:   end if
18: end if
19: Check all tasks for reached release times and set them to the ready state
20: STEP 3: Go to step 1
```

all reserved processor time $\mathfrak{R}$, the state switches from *ready* to *overrun* and the task is preempted by other *ready* tasks. Replenishment of $\mathfrak{R}$ is then assigned to task OP_{ti} when a new task OP_{ti} begins. However, to ensure efficient use of processor time, if an *overrun* task exists and there is no *ready* task, $\mathfrak{R}$ will be assigned to the *overrun* task and the task is processed until a *ready* task appears. Algorithm 3 shows the proposed ROP-EDF scheduling algorithm.

Theorem 1. *The upper CPU utilization bound of total hard real-time tasks U_{ub} for ROP-EDF is $U_{ub} = 1 - \alpha$.*

Proof. According to a schedulability test by Liu et al. [23], a task set of s independent, preemptable, periodic tasks with relative deadlines D_i equal to their periods T_i can be scheduled on a single processor feasibly if Eq. (3) is valid.

$$\sum_{i=1}^{s} \frac{C_i}{D_i} \leq 1 \tag{3}$$

Here, C_i denotes execution time.

In ROP-EDF, the real-time task capacity for soft and hard tasks is managed separately to prevent hard real-time tasks from being deprived of processor time by soft real-time tasks that have been released prior to hard real-time tasks. If 100% processor time is allocated to hard real-time tasks and the total amount of overhead incurred during scheduling and context-switching α is 0, then ROP-EDF performs in the same manner as EDF. In this paper, we assume that the worst-case execution time of each task already includes cache-related preemption overhead. Thus, hard real-time tasks can reserve processor time and meet deadlines if Eq. (3) is valid regardless of the requested CPU utilization for soft real-time tasks. Thus, according to Eq. (3), the upper CPU utilization bound for the total hard tasks is $1 - \alpha$. □

5. Performance Evaluation

Simulations of ROP-EDF were conducted to evaluate and compare performance with OP-EDF and ER-EDF [*3].

5.1 Experimental Setup

Simulation experiments were conducted using a PC (Intel Core i7-4500U, 1.80 GHz; 8 GB RAM). We assumed that all hard real-time tasks were periodic tasks that must complete by their deadlines. The relative deadlines of the periodic tasks were assumed to be equal to their periods. The peak CPU utilization of *OP* indicates the highest utilization among all *OP* jobs, and the average CPU utilization of *OP* equals the average CPU utilization among all *OP* jobs. The requested CPU utilization of *OP* is the peak CPU utilization for hard real-time tasks and the average CPU utilization for soft real-time tasks. The requested CPU utilization is required to determine the reserved processor time for policy 2. Note that context-switching overhead ($\alpha = 0$) was ignored in these simulations. The DMR is the primary comparison parameter, which is expressed as follows.

$$DMR = \frac{1}{b} \sum_{i=1}^{b} miss(f_i) \tag{4}$$

Here, b is the number of jobs, and $miss(f_i)$ is defined as follows.

$$miss(f_i) = \begin{cases} 0 & (f_i \leq d_i) \\ 1 & (\text{otherwise}) \end{cases}$$

Here, f_i and d_i are the completion time and absolute deadline of the i-th job, respectively.

5.2 OP-EDF Comparison

Here, the task set comprised two hard and three soft real-time tasks (**Table 2**). The relative task deadlines were equal to their periods. The execution time was 20,000 ms. This experiment was performed within a particular requested CPU utilization range [100%, 130%]; i.e., the system was full and overload conditions.

The simulation parameters used for this experiment when the total requested CPU utilization was 100% are shown in Table 2. All jobs for each *OP* received constant CPU utilization (denoted cnst) when the total requested CPU utilization was 100%. For the parameters in the range [110%, 130%], CPU utilizations of OP_{S2}

Table 2 Total Requested CPU Utilization = 100%.

TASK (OP_{ti})	OP_{H1}	OP_{H2}	OP_{S1}	OP_{S2}	OP_{S3}
The number of jobs	222	200	100	133	200
Period (T)	90 ms	100 ms	200 ms	150 ms	100 ms
Relative deadline (D)	90 ms	100 ms	200 ms	150 ms	100 ms
Distribution	cnst	cnst	cnst	cnst	cnst
CPU util. of jobs (U)	25%	16%	26%	18%	15%
Requested CPU util.	25%	16%	26%	18%	15%
Average CPU util. (θ)	25%	16%	26%	18%	15%
Peak CPU util. (ψ)	25%	16%	26%	18%	15%
Total peak CPU util.			100%		
Total requested CPU util.			100%		

[*3] A constant bandwidth server (CBS) based on resource reservation [27] is not included in the performance evaluation because it does not discuss a task with multiple deadlines; thus, it cannot directly schedule tasks having multiple deadlines.

Table 3 Total Requested CPU Utilization = 110%.

TASK (OP_{ti})	OP_{H1}	OP_{H2}	OP_{S1}	OP_{S2}	OP_{S3}
The number of jobs	222	200	100	133	200
Period (T)	90 ms	100 ms	200 ms	150 ms	100 ms
Relative deadline (D)	90 ms	100 ms	200 ms	150 ms	100 ms
Distribution	cnst	cnst	cnst	cnst	gaus
CPU util. of jobs (U)	25%	16%	26%	18%	15%–35%
Requested CPU util.	25%	16%	26%	18%	25%
Average CPU util. (θ)	25%	16%	26%	18%	25%
Peak CPU util. (ψ)	25%	16%	26%	18%	35%
Total peak CPU util.			120%		
Total requested CPU util.			110%		

Table 4 Total Requested CPU Utilization = 120%.

TASK (OP_{ti})	OP_{H1}	OP_{H2}	OP_{S1}	OP_{S2}	OP_{S3}
The number of jobs	222	200	100	133	200
Period (T)	90 ms	100 ms	200 ms	150 ms	100 ms
Relative deadline (D)	90 ms	100 ms	200 ms	150 ms	100 ms
Distribution	cnst	cnst	cnst	cnst	gaus
CPU util. of jobs (U)	25%	16%	26%	18%	15%–55%
Requested CPU util.	25%	16%	26%	18%	35%
Average CPU util. (θ)	25%	16%	26%	18%	35%
Peak CPU util. (ψ)	25%	16%	26%	18%	55%
Total peak CPU util.			140%		
Total requested CPU util.			120%		

Table 5 Total Requested CPU Utilization = 130%.

TASK (OP_{ti})	OP_{H1}	OP_{H2}	OP_{S1}	OP_{S2}	OP_{S3}
The number of jobs	222	200	100	133	200
Period (T)	90 ms	100 ms	200 ms	150 ms	100 ms
Relative deadline (D)	90 ms	100 ms	200 ms	150 ms	100 ms
Distribution	cnst	cnst	cnst	gaus	gaus
CPU util. of jobs (U)	25%	16%	26%	18%–38%	15%–55%
Requested CPU util.	25%	16%	26%	28%	35%
Average CPU util. (θ)	25%	16%	26%	28%	35%
Peak CPU util. (ψ)	25%	16%	26%	38%	55%
Total peak CPU util.			160%		
Total requested CPU util.			130%		

and OP_{S3} were changed to increase the total requested CPU utilization to 110%, 120%, and 130%. The requested CPU utilization of OP_{S3} was set to 25%, and the CPU utilization of the OP_{S3} jobs ranged from 15% to 35% (**Table 3**). For the total requested CPU utilization of 120%, the requested CPU utilization of OP_{S3} was changed to 35%, and the CPU utilization of OP_{S3} jobs ranged from 15% to 55% (**Table 4**). At 130%, the CPU utilization requested by OP_{S2} was set to 28%, and the CPU utilization of its OP_{S2} jobs ranged from 18% to 38% (**Table 5**).

For OP_{S2} and OP_{S3} jobs with variable CPU utilization, we assumed that the processing time of each job would vary because the amount of received data changes according to the number of vehicles within V2V communication range. The variable OP_{S2} and OP_{S3} CPU utilizations were assigned by random Gaussian distribution. We assumed that all hard real-time OP_{H1} and OP_{H2} tasks were periodic. The task order was as follows: tasks one and two were hard real-time tasks, and tasks three to five were soft real-time tasks. Thus, hard real-time tasks were allocated to a processor first when their absolute deadlines and release times were the same as the soft real-time tasks. We generated 10 task sets for each requested CPU utilization level and compared the average DMR obtained by each algorithm.

We evaluated the DMR of hard and soft real-time tasks scheduled by the proposed ROP-EDF and OP-EDF for the range [100%, 130%] of total requested CPU utilization. When the total

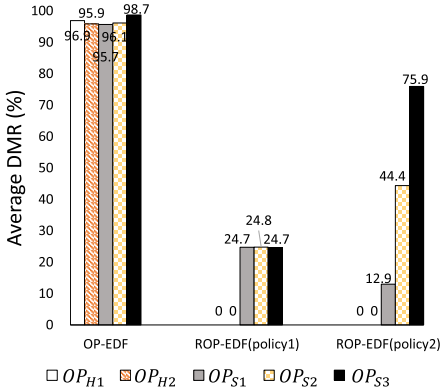

Fig. 6 Total requested CPU utilization = 110%.

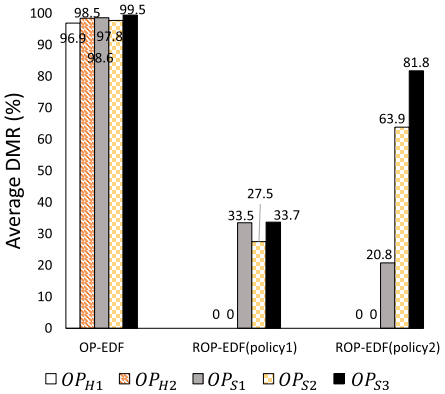

Fig. 7 Total requested CPU utilization = 120%.

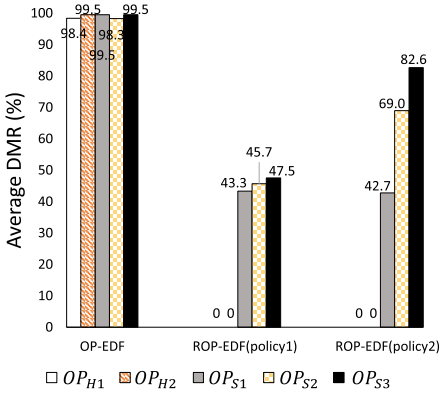

Fig. 8 Total requested CPU utilization = 130%.

requested CPU utilization was 100%, no deadlines were missed for five *OP*s because both algorithms are based on the EDF algorithm, which guarantees that all deadlines will be met if the total CPU utilization does not exceed 100%. At 110%, OP_{H1} and OP_{H2} missed deadlines under OP-EDF; however, no deadlines were missed under the proposed ROP-EDF (**Fig. 6**). As shown in **Figs. 7** and **8**, when the total CPU utilization was 120% to 130%, all *OP*s scheduled by OP-EDF missed deadlines, and the miss rates were greater than 95%. Under ROP-EDF, no deadlines were missed for hard real-time *OP*s because ROP-EDF reserves sufficient processor time.

Under policy 1, soft real-time *OP*s began to miss deadlines because the allocated processor time was insufficient in the range [110%, 130%]. More than one of the three soft real-time *OP*s could not release jobs at the least common multiple period be-

cause competition to reserve execution time occurred. Furthermore, soft real-time *OP*s missed deadlines because the task with the lowest DMR was rejected to prevent task starvation. All missed deadlines for soft real-time tasks were caused by job rejection; however, soft real-time tasks that were released could complete jobs on time.

Under policy 2, the processor time distributed proportionally to average CPU utilizations was allocated to three soft real-time *OP*s. To prevent accumulation of unfinished jobs, a task cannot release a new job while unfinished jobs remain. A soft real-time *OP* with a shorter relative deadline has a higher DMR, as shown in Fig. 6 to Fig. 8. In contrast, a soft real-time *OP* with a longer relative deadline has more chances to be executed before its absolute deadline with additional replenishment of processor time. This occurs when all hard real-time tasks are completed and no *ready* state tasks exist.

5.3 ER-EDF Comparison

We compared the average DMR for hard real-time tasks scheduled by the proposed ROP-EDF and ER-EDF. This experiment was performed in the range [100%, 130%] of total requested CPU utilization. We generated 50 task sets randomly for 100%, 110%, 120%, and 130% requested CPU utilization. Each task set had 10 tasks comprising four hard real-time tasks and six soft real-time tasks. The first to ninth tasks were periodic tasks, and the tenth task was aperiodic. The task order in each task set was as follows: tasks one to four were hard real-time tasks, and tasks five to ten were soft real-time tasks. Thus, hard real-time tasks were allocated to a processor first when their absolute deadlines and release times were the same as the soft real-time tasks. The relative deadlines of the first to ninth tasks were generated randomly in the range [10 ms, 100 ms]. In addition, the requested CPU utilizations of all tasks, except for the aperiodic task, were generated randomly, and the summation of their requested CPU utilizations was 100%. It was assumed that the minimum arrival time between any two consecutive jobs of the aperiodic task and the relative deadline was 50 ms for the aperiodic task, and its requested CPU utilization was set to the total requested CPU utilization of each task set to become 110%, 120%, and 130%. The execution time of the first experiment was 20,000 ms, and we compared the average DMR of the hard real-time tasks for each total requested CPU utilization rate.

We evaluated the average DMR of the hard real-time tasks scheduled by the proposed ROP-EDF and ER-EDF in the total requested CPU utilization range [100%, 130%]. **Figure 9** shows the average DMR measured for each ER-EDF and OP-EDF simulation performed at 100%, 110%, 120%, and 130% utilization rates. As can be seen in Fig. 9, the proposed ROP-EDF did not miss any deadlines for hard real-time tasks for any total requested CPU utilization. In contrast, the average DMR for the hard real-time tasks increased as the total requested CPU utilization increased with ER-EDF. The processor time was insufficient when the total requested CPU utilization exceeded 100% due to the aperiodic task. Thus, tasks that could not reserve sufficient processor time could be hard real-time tasks because the aperiodic task reserved processor time before the hard real-time tasks could reserve processor time. On the other hand, the hard real-time tasks with ROP-EDF could reserve sufficient processor time even if the total requested CPU utilization exceeded 100%, because capacities for hard and soft real-time tasks were managed individually.

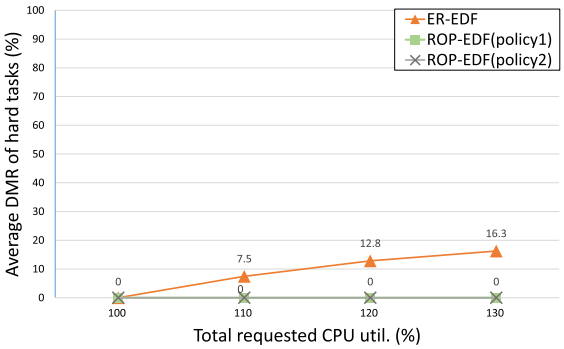

Fig. 9 Comparison of DMR under ER-EDF and ROP-EDF.

5.4 Discussion

In the first experiment, there were no missed deadlines for the hard real-time tasks with ROP-EDF; however, the average DMRs for all tasks under OP-EDF were greater than 95% when the total requested CPU utilization exceeded 100%. The DMR of the hard real-time tasks is a constant zero, which means that the schedulability of the hard real-time tasks is not influenced by the soft real-time tasks.

In the second experiment, the average DMR of the hard real-time tasks increased with total requested CPU utilization under ER-EDF. In contrast, the deadlines for all hard real-time tasks were met under the proposed ROP-EDF. Therefore, we consider that policy 1 should be applied to systems that prioritize the completion of soft real-time tasks on time, and policy 2 should be applied to systems that must execute as many soft real-time tasks as fairly as possible.

6. Related Work

Babcock et al. [8] introduced the chain scheduling algorithm that targets applications such as networking systems, financial services, and sensor networks. This algorithm attempts to minimize run-time system memory usage by reducing the internal buffer size. Furthermore, it uses the average execution time and selectivity of operators as parameters to adapt to changing system loads. However, this algorithm does not consider hard real-time constraints, which is important for real-time processing. Thus, it is difficult to adopt the chain scheduling algorithm in embedded automotive systems.

Schmidt et al. [11] introduced the RM scheduling algorithm, which is based on the rate monotonic algorithm and a hard real-time scheduling strategy. It targets electrical train services and presents two different strategies, i.e., minimum output delay and maximum DSMS throughput. RM scheduling attempts to satisfy timing constraints by assigning higher priority to operators with shorter periods. This algorithm assumes that RM scheduling can be applied to applications with only periodic tasks and is thus difficult to employ in embedded automotive systems because tasks with unpredictable arrival times should be schedulable.

Table 6 ROP-EDF vs Prior Work.

	Hard real-time guarantee in non-overload conditions	Hard real-time guarantee in overload conditions	Tasks with multiple deadlines	Requested CPU util. over 100%
Chain algorithm [8]				
RM algorithm [11]	✗			
RTSTREAM [12]	✗	✗		
Semantic load shedding [13]	✗			
OP-EDF [10]	✗		✗	
D_R_EDF etc. [24], [25], [26]	✗			
CBS etc. [27], [28], [29], [30]	✗	✗		✗
ER-EDF [18]	✗	✗		
ROP-EDF	✗	✗	✗	✗

Wei et al. [12] proposed RTSTREAM, which can handle huge amounts of sensor data streams and can process queries according to their deadlines. However, it is difficult to apply RTSTREAM to an automotive DSMS because it cannot handle a query with more than one deadline. RTSTREAM considers a query as a task and assumes that each task has only one deadline. In an automotive DSMS, data processing in a query should be shared with multiple applications with different deadlines.

Ma et al. [13] proposed the semantic load shedding algorithm, which targets Forest Fireproof Monitoring Control (FFMC). Note that FFMC requires real-time processing. They used an algorithm based on a priority table that considers the execution cost and value of a tuple. Tuples with higher execution cost and a lower value will be dropped first to shed load efficiently in overload conditions. Thus, they can determine which tuples should be discarded to decrease system load to obtain better performance. However, it is difficult to employ the semantic load shedding algorithm in embedded automotive systems because it only concentrates on minimizing the DMR, the tuple value loss ratio, and load shedding overhead.

Li et al. [10] introduced the OP-EDF scheduling algorithm for a DSMS in which a task with a shorter absolute deadline is assigned higher priority to minimize the DMR. They define an *OP* that consists of operators and queues and treat this *OP* as a real-time task instance. OP-EDF defines task allocation using multiple operators and queues. In addition, OP-EDF addresses task dependence and reduces system overhead by grouping operators and queues as a single task. The EDF algorithm works effectively under non-overload conditions; however, it can miss deadlines under overload conditions. Accordingly, OP-EDF does not meet the requirement of embedded automotive systems that hard real-time tasks must meet their deadlines even under overload conditions.

Several algorithms have been proposed to eliminate the weaknesses of EDF [24], [25], [26], because EDF misses deadlines unpredictably when the system is overloaded. The adaptive scheduling algorithm [24] uses EDF in underload conditions and uses Ant Colony Optimization to obtain outstanding results under overload conditions. The D_R_EDF algorithm [25] and a task scheduling algorithm [26] use EDF when a system is not overloaded, and these algorithms switch to the more efficient rate-monotonic algorithm in overload conditions. However, these algorithms aim to improve the success ratio of jobs and CPU utilization. Therefore, hard real-time tasks can miss deadlines in overload conditions with these algorithms.

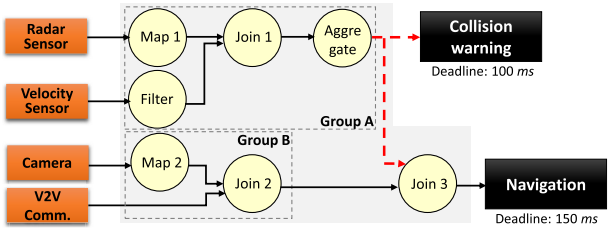

Fig. 10 Query for automotive data processing.

In traditional real-time systems, there are many mixed scheduling algorithms based on resource reservation strategies to schedule hard periodic tasks and soft aperiodic tasks [27], [28], [29], [30]. These algorithms are based on reserving a fraction of the processor time to each task using the EDF algorithm. We show an example of a query for automotive data processing and explain why such existing methods are not applicable to an automotive DSMS in **Fig. 10**.

Group A in Fig. 10 which consists of four operators outputs vehicle data from on-board sensors. The output data from Group A is combined with output data from Group B, which provides information about surrounding vehicles based on an on-board sensor and V2V communication using Join 3. This combined data is used for automotive navigation with a deadline of 150 ms, and the output data from Group A is used for a collision risk warning application with a deadline of 100 ms. As a result, the operators in Group A and the output result from Group A are shared between two applications with different deadlines. According to this example, the operators in the automotive DSMS are shared between multiple applications to reduce memory usage. In addition, there are operators that output multiple processed data from one input data, and these output data are used for different applications. Thus, an algorithm for an automotive DSMS should be able to handle operators with more than one deadline.

However, the existing algorithms based on resource reservation strategies [27], [28], [29], [30] assume that a single task has only one deadline or period. Consequently, these algorithms cannot handle tasks with multiple deadlines directly.

Table 6 shows a comparison of ROP-EDF and previous work. ROP-EDF can handle hard real-time tasks under high overload conditions and tasks with multiple deadlines. Furthermore, the proposed algorithm can guarantee the hard real-time constraints when the total requested CPU utilization is over 100%. The CBS resource reservation strategy [27] is similar to ROP-EDF; however, it cannot directly manage tasks that have multiple deadlines. This is because CBS assumes that a single task has only one dead-

line or period. To deal with a task with one or more deadlines, ROP-EDF uses methods required to construct operator paths described in Section 2 and applies resource reservation strategies to guarantee hard real-time constraints under overload conditions. Note that ROP-EDF has demonstrated all features shown in Table 6.

7. Conclusion

This paper has proposed the ROP-EDF algorithm, which is an efficient algorithm for load testing on a single processor for an automotive DSMS. The proposed algorithm is based on the OP-EDF and ER-EDF algorithms and resolves the limitations of existing algorithms based on resource reservation. We consider cases where hard real-time tasks can miss deadlines under OP-EDF when a system is overloaded. With ER-EDF, hard real-time tasks cannot always reserve processor time when the total requested CPU utilization of the task set exceeds 100%. We have addressed these problems, and the proposed ROP-EDF shows significant improvement compared to the OP-EDF and ER-EDF algorithms. According to our simulation results, ROP-EDF offers schedulability of hard real-time tasks even under overload conditions. While the DMRs increase for hard real-time tasks in OP-EDF and ER-EDF when the total requested CPU utilization exceeds 100%, the proposed ROP-EDF can guarantee deadlines for hard real-time tasks.

In future, we plan to extend the ROP-EDF algorithm so that it can be employed in distributed environments. In addition, if an automotive DSMS is implemented and run on multiple processors connected over communication networks, communication latency between processors should be considered.

References

[1] Yamada, M., Sato, K. and Takada, H.: Implementation and Evaluation of Data Management Methods for Vehicle Control Systems, *Proc. IEEE Vehicular Technology Conference* (*VTC Fall*), pp.5–8 (2011).

[2] Bolles, A., Appelrath, H., Geesen, D., Grawunder, M., Hannibal, M., Jacobi, J., Koster, F. and Nicklas, D.: StreamCars: A new flexible architecture for driver assistance systems, *Intelligent Vehicles Symposium* (*IV*), pp.252–257 (2012).

[3] Katsunuma, S., Honda, S., Sato, K., Watanabe, Y., Nakamoto, Y. and Takada, H.: Real-time-aware Embedded DSMS Applicable to Advanced Driver Assistance Systems, *Proc. IEEE 33rd International Symposium on Reliable Distributed Systems Workshops*, pp.106–111 (2014).

[4] Abadi, D.J., Ahmad, Y., Balazinska, M., Cherniack, M., Hwang, J., Lindner, W., Maskey, A.S., Rasin, E., Ryvkina, E., Tatbul, N., Xing, Y. and Zdonik, S.: The Design of the Borealis Stream processing Engine, *Proc. 2nd Biennial Conference on Innovative Data Systems Research* (*CIDR'05*), pp.277–289 (2005).

[5] Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Motwani, R., Nishizawa, I., Srivastava, U., Thomas, D., Varma, R. and Widom, J.: STREAM: The Stanford stream data manager, *Proc. IEEE Data Engineering Bulletin*, Vol.26, pp.19–26 (2003).

[6] Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M.J., Hellerstein, J.M., Hong, W., Krishnamurthy, S., Madden, S.R., Reiss, F. and Shah, M.A.: TelegraphCQ: Continuous dataflow processing, *Proc. 2003 ACM SIGMOD International Conference on Management of Data*, pp.668–668 (2003).

[7] Babcock, B., Babu, S., Datar, M., Motwani, R. and Thomas, D.: Operator Scheduling in Data Stream Systems, *Proc. VLDB Journal*, Vol.13, No.4, pp.333–353 (2004).

[8] Babcock, B., Babu, S., Datar, M. and Motwani, R.: Chain: Operator Scheduling for Memory Minimization in Data Stream Systems, *Proc. ACM SIGMOD International Conference on Management of Data*, pp.253–264 (2003).

[9] Pietzuch, P., Ledie, J., Shneidman, J., Roussopoulos, M., Welsh, M. and Seltzer, M.: Network-aware operator placement for stream-processing systems, *Proc. IEEE International Conference on Data Engineering* (*ICDE'06*), p.49 (2006).

[10] Li, X., Jia, Z., Ma, L., Zhang, R. and Wang, H.: Earliest Deadline Scheduling for Continuous Queries over Data Streams, *Proc. IEEE 6th International Conference on Embedded Software and Systems* (*ICESS'09*), pp.57–64 (2009).

[11] Schmidt, S., Legler, T., Schaller, D. and Lehner, W.: Real-Time Scheduling for Data Stream Management Systems, *Proc. 17th Euromicro Conference on Real-Time Systems, ECRTS'05*, pp.167–176 (2005).

[12] Wei, Y., Son, S. and Stankovic, J.: RTSTREAM: Real-Time Query Processing for Data Streams, *Proc. IEEE 9th International Symposium on Object and Component-Oriented Real-Time Distributed Computing* (*ISORC'06*), pp.141–150 (2006).

[13] Ma, L., Zhang, Q. and Shi, N.: A semantic load shedding algorithm based on priority table in Data Stream System, *Seventh International Conference on Fuzzy Systems and Knowledge Discovery* (*FSKD'10*), pp.1167–1172 (2010).

[14] Obst, M., Mattern, N., Schubert, R. and Wanielik, G.: Car-to-Car communication for accurate vehicle localization – The CoVeL approach, *Proc. 9th International Multi-Conference on Systems, Signals and Devices* (*SSD'12*), pp.1–6 (2012).

[15] Golestan, K., Seifzadeh, S., Kamel, M., Karray, F. and Sattar, F.: Vehicle Localization in VANETs Using Data Fusion and V2V Communication, *Proc. 2nd ACM International Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications* (*DIVANet*), pp.123–130 (2012).

[16] Jiao, Y., Xiaoguang, Y., Di, W. and Baiying, Shi.: Urban Traffic Control Tentative Exploration in Vehicle-Infrastructure Integration Environment, *Proc. International Conference on Intelligent Computation Technology and Automation* (*ICICTA*), pp.1110–1113 (2010).

[17] Intelligent Transport Systems (ITS); V2X Applications; Part 3: Longitudinal Collision Risk Warning (LCRW) Application Requirements Specification, ETSI TS 101 539-3 (2013).

[18] Matschulat, D., Marcon, C.M. and Hessel, F.: ER-EDF: A QoS Scheduler for Real-Time Embedded Systems, *Proc. IEEE/IFIP 18th International Workshop on Rapid System Prototyping*, pp.181–188 (2007).

[19] Buttazzo, G.: Hard real-time computing systems: Predictable scheduling algorithms and applications, 3rd ed., Springer Publishing Company (2011).

[20] Toyota to Bring Vehicle-Infrastructure Cooperative Systems to New Models in 2015, available from ⟨http://newsroom.toyota.co.jp/en/detail/4228471⟩ (accessed 2015-08-15).

[21] Yuan, W., Nahrstedt, K. and Kim, K.: R-EDF: A reservation-based edf scheduling algorithm for multiple multimedia task classes, *Proc. IEEE the Seventh Real-Time Technology and Applications Symposium* (*RTAS'01*), pp.149–154 (2001).

[22] Srinivasan, A., Holman, P., Anderson, J.H., Baruah, S. and Layland, J.: The case for fair multiprocessor scheduling, *Proc. 11th International Workshop on Parallel and Distributed Real-time Systems* (2003).

[23] Liu, C.L. and Layland, J.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *J. ACM* (*JACM*), pp.46–61 (1973).

[24] Kotecha, K. and Shah, A.: Adaptive scheduling algorithm for real-time operating system, *Proc. IEEE World Congress on Computational Intelligence*, pp.2109–2112 (2008).

[25] Kotecha, K. and Shah, A.: Efficient Scheduling Algorithms for Real-Time Distributed Systems, *Proc. 1st International Conference on Parallel, Distributed and Grid Computing* (*PDGC'10*), pp.44–48 (2010).

[26] Sharma, K. and Kumar, P.: A Novel Task Scheduling Algorithm for Real Time Systems, *Proc. International Conference on Communications and Signal Processing* (*ICCSP'13*), pp.995–998 (2013).

[27] Abeni, L. and Buttazzo, G.: Integrating Multimedia Applications in Hard Real-Time Systems, *Proc. IEEE Real-Time Systems Symposium*, pp.4–13 (1998).

[28] Caccamo, M., Buttazzo, G. and Thomas, D.: Efficient Reclaiming in Reservation-Based Real-Time Systems with Variable Execution Times, *IEEE Transactions on Computers*, Vol.54, pp.198–213 (2005).

[29] Lin, C. and Brandt, S.A.: Improving Soft Real-Time Performance Through Better Slack Reclaiming, *Proc. IEEE Real-Time Systems Symposium* (*RTSS'05*), pp.410–421 (2005).

[30] Nogueira, L. and Phnho, L.M.: A capacity sharing and stealing strategy for open real-time systems, *Proc. Journal of Systems Architecture*, Vol.56, pp.163–179 (2010).

Jaeyong Rho received his B.E. and M.S. degrees from Ritsumeikan University in Information Science and Engineering, Japan, in 2014 and 2016 respectively. His research interests include real-time scheduling and automotive embedded systems.

Takuya Azumi is an Assistant Professor at the Graduate School of Engineering Science, Osaka University. He received his Ph.D. degree from the Graduate School of Information Science, Nagoya University. From 2008 to 2010, he was under the research fellowship for young scientists for Japan Society for the Promotion of Science. From 2010 to 2014, he was an Assistant Professor at the College of Information Science and Engineering, Ritsumeikan University. His research interests include real-time operating systems and component based development. He is a member of IEEE, IEICE, and JSSST.

Akihiro Yamaguchi is a Researcher at Center for Embedded Computing Systems, Nagoya University. He graduated a Master in mathematics from Kobe University in 2006. From 2006 to 2010, he worked at TOSHIBA CORPORATION. His research interests include data stream management systems.

Kenya Sato received his B.E. and M.E. degrees from Osaka University, Japan, in 1984 and 1986 respectively. He received his Ph.D. degree from Nara Institute of Science and Technology, Japan, in 2000. In 1986–1991, he was a research engineer at Information and Electronics Research Laboratory, Sumitomo Electric Industries, Japan, and in 1991–1993, he was a visiting researcher in Computer Science Department, Stanford University, California, USA. Dr. Sato served as a chief technologist at Automotive Multimedia Interface Collaboration, Inc., Michigan, USA, in 2001–2003. Since 2004, he is an associate professor in Department of Information Systems Design, Doshisha University, Kyoto, Japan. His research interests include network architecture, distributed systems, embedded systems, and ITS. He is a member of IEEE computer society and ACM.

Nobuhiko Nishio was born in 1962. Graduated Mathematical Engineering and Information Physics, School of Engineering, The University of Tokyo in 1986. M.S. from Graduate School of The University of Tokyo in 1988. After quitting Ph.D. course, worked at Keio University Shonan Fujisawa Campus from 1993 to 2003. Ph.D. in Media and Governance in 2000. Associate professor at College of Science and Engineering, Ritsumeikan University in 2003. Since 2005, Professor at College of Information Science and Engineering, Ritsumeikan University (Current profession). Visiting Scientist in Google Inc. from 2007 to 2008. Majored in autonomous, distributed and collaborative systems, ubiquitous computing and networking and sensing systems. Awarded as the Yamashita Memorial Research Prize from IPSJ. Regular member of ACM and IEEE Computer Society.