Deadline Miss Early Detection Method for DAG Tasks Considering Variable Execution Time

Hayate Toba
Graduate School of Science and Engineering, Saitama University, Japan

Takuya Azumi
Graduate School of Science and Engineering, Saitama University, Japan

Abstract

Autonomous driving systems must guarantee safety, which requires strict real-time performance. A series of processes, from sensor data input to vehicle control command output, must be completed by the end-to-end deadline. If a deadline miss occurs, the system must quickly transition to a safe state. To improve safety, an early detection method for deadline misses was proposed. The proposed method represents the autonomous driving system as a directed acyclic graph (DAG) with a mixture of timer-driven and event-driven nodes. It assigns appropriate time constraints for each node based on the end-to-end deadline. However, the existing methods assume the worst-case execution time (WCET) for calculating the time constraints of each node and do not consider the execution time variation of nodes, making the detection of deadline misses pessimistic. This paper proposes a deadline miss early detection method to determine the possibility of deadline misses quantitatively at the beginning of each node execution in a DAG task. It calculates the time constraints of each node using probabilistic execution time, which treats execution time as a random variable. Experimental evaluation shows that the proposed method reduces pessimism, which is a problem of conventional methods using WCET, and then achieves more accurate early detection of deadline misses. The evaluation also indicates that the execution time of static analysis required for deadline miss early detection is within a practical level.

2012 ACM Subject Classification Computer systems organization → Real-time systems

Keywords and phrases Autonomous driving system, deadline miss early detection, DAG, event-driven task, timer-driven task, probabilistic execution time

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2024.8

Funding This work was supported by JST PRESTO Grant Number JPMJPR21P1.

1 Introduction

In recent years, autonomous vehicles have attracted a great deal of attention. The realization and widespread adoption of autonomous vehicles are expected to solve social problems such as traffic congestion, traffic accidents, transportation in rural areas, and labor shortages in logistics. Examples of autonomous driving systems include *Waymo Driver* [18], *Autoware* [7], and *Apollo* [1], and all of these systems aim for fully autonomous driving. According to the standards set by the society of automotive engineers, there are six levels of autonomous driving ranging from zero to five [15]. As the level increases, the responsibility of the system becomes greater. Fully autonomous driving corresponds to level five, and the system of level three and above must guarantee safety even in emergency situations.

To ensure the safe operation, autonomous driving systems must be strictly real-time. This is because even a slight delay in processing can lead to a serious accident. For instance, a delay in emergency braking or processing of collision avoidance action can cause an accident that results in the loss of life. Therefore, it must be guaranteed that all processing – from sensor data acquisition at the input of the system to vehicle control at the output – will be completed within a specified time frame. To this end, an autonomous driving system can be

represented as a directed acyclic graph (DAG) with an end-to-end deadline [11, 4, 8]. With the DAG representation, the system can be analyzed to satisfy the real-time constraints, considering complex dependencies among many tasks.

Efficient scheduling algorithms that take a deadline into account have contributed to the reduction of deadline misses in DAGs. However, completely eliminating the probability of deadline misses is still impossible due to various complex factors. This challenge underscores the necessity for a functionality that shifts immediately to a safe state when a deadline miss occurs. Implemented in autonomous driving systems as a solution is a minimum risk maneuver (MRM) mode. By immediately shifting to MRM mode when a deadline miss is detected, the occurrence of serious accidents can be mitigated. However, the current system can only detect a deadline miss at the end of the processing flow. To address this problem, a method for deadline miss early detection was proposed [19]. This method sets appropriate time constraints for each task in the middle based on the deadline given at the end of the processing flow. Detecting deadline misses and shifting to MRM mode earlier can further improve the safety of autonomous driving systems.

The existing method [19] assumes that the execution time of each task in the processing flow is the worst-case execution time (WCET). However, the actual execution time of tasks is variable, and the probability of them reaching WCET is extremely low. Therefore, assuming that task execution time is always WCET leads to pessimistic analysis results. In other words, there are cases where a deadline miss is predicted in the middle of the processing flow, and the system shifts to MRM, but the deadline may have been met even if the processing had continued. When the probability of the execution time reaching WCET is significantly low, the majority of MRM transitions are inherently unnecessary and interfere with the smooth operation of the system.

To reduce this pessimism, this paper proposes a deadline miss early detection method that takes into account the variation in execution time of each task in the system. To account for the variation, the execution time of each task is taken as a random variable. In this case, the problem is more complex than the situation when only WCET is considered. This is because, in order to set time constraints at the middle of the processing flow based on end-to-end deadlines, it is necessary to calculate the sum of execution times of multiple consecutive tasks in the flow. If the execution time of each task is assumed to be a fixed value, this calculation is a simple addition. However, when the execution time of each task is considered as a random variable, the sum cannot be calculated in the same way as for fixed values. Therefore, we introduce a method that convolves the execution time distributions of consecutive tasks to obtain their total execution time distribution. This gives the deadline miss probability from the middle of the processing flow, and the time constraints are set so that this probability is below a user-defined threshold.

The primary contributions of this paper are summarized as follows:

- We propose the deadline miss early detection method with reduced pessimism by introducing probabilistic execution time to consider task execution time variability.
- We enable flexible deadline miss early detection by convolving probabilistic execution times and defining a probabilistic time constraint on the start time of each job.
- We show that the proposed method outperforms the conventional method and completes the calculation of time constraints in practical time through experimental evaluation on random DAGs modeled after an actual autonomous driving system.

The remainder of this paper is organized as follows. Section 2 describes the system model of this paper. Section 3 defines the problem model addressed in this paper. Section 4 introduces the proposed method. Section 5 evaluates and discusses the proposal method. Section 6 introduces related research. Section 7 presents the conclusions and future work.

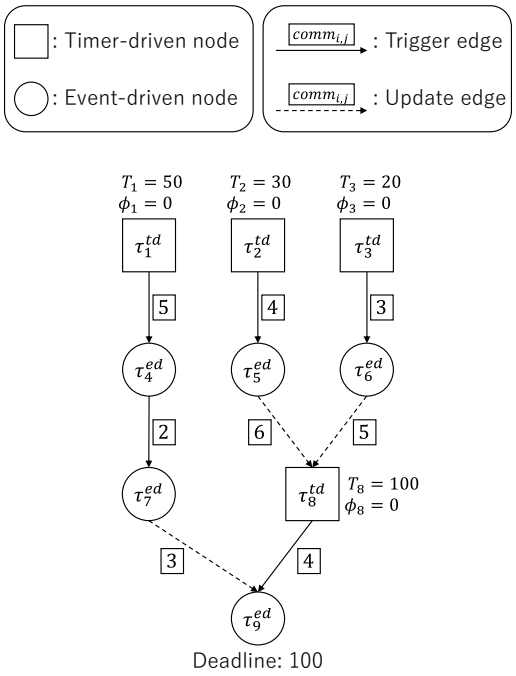


Figure 1 Simple example of DAG.

Table 1 DAG notations.

Symbol	Description
n	Total number of nodes in a DAG
τ_i	Node
τ_i^{td}	*Timer-driven node*
τ_i^{ed}	*Event-driven node*
V^{td}	Set of all *timer-driven nodes* in a DAG
T_i	Period of τ_i^{td}
ϕ_i	Offset of τ_i^{td}
$\langle \tau_i, k \rangle$	k^{th} job of τ_i
$\langle \tau_i^{td}, k \rangle$	k^{th} job of τ_i^{td}
$\langle \tau_i^{ed}, k \rangle$	k^{th} job of τ_i^{ed}
X_i	Probabilistic execution time of τ_i
$\mathbb{X}_i$	Probability distribution of X_i
$e_{i,j}^{tr}$	*Trigger edge* from τ_i to τ_j
$e_{i,j}^{up}$	*Update edge* from τ_i to τ_j
$comm_{i,j}$	Worst-case communication time of $e_{i,j}$

2 System Model

This section outlines the system model of this paper. We first introduce the DAG model and then define probabilistic execution time.

2.1 DAG

An autonomous driving system can be represented as a DAG, where tasks are nodes, and the data flows between tasks are directed edges. The inputs of the system are data from various sensors such as cameras, LiDAR, and the global navigation satellite system (GNSS), which are acquired in different periods. Using these sensor data, the system performs self-localization, object perception, and route planning, and finally outputs a single vehicle control command. Thus, the DAG considered in this paper has multiple entry nodes with different periods and a single exit node. In addition, since vehicle control must not be delayed in an autonomous driving system, a deadline is defined at the node that outputs the control command (i.e., exit node). A simple example of the DAG considered in this paper is shown in Figure 1.

DAG notations in this paper are listed in Table 1. A DAG is denoted as $G = (V, E)$, where $V = \{\tau_1, \tau_2, \ldots, \tau_i, \ldots, \tau_n\}$ is the set of all nodes, each node τ_i represents the i^{th} task in the DAG, and $n = |V|$ is the total number of tasks in the DAG. There are two types of node:

timer-driven nodes – nodes that are triggered at a given period, denoted by τ_i^{td}

event-driven nodes – nodes that are triggered when data from their predecessor nodes arrive, denoted by τ_i^{ed}.

Each *timer-driven node* τ_i^{td} has the period T_i and the offset ϕ_i, where the offset means the start time of the first execution of *timer-driven nodes*. A node is executed an infinite number of times, and each execution is called a job. The k^{th} job of τ_i is denoted by $\langle \tau_i, k \rangle$. Each node has a probabilistic execution time denoted as X_i, and the probability distribution of the execution time of the job is the same as that of the node.

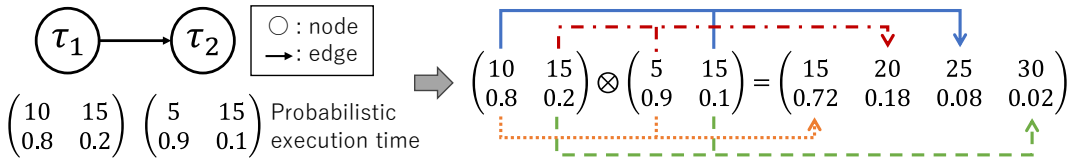


Figure 2 Probabilistic total execution time of two nodes.

E is the set of all edges, and each edge $e_{i,j} \in E$ represents communication between nodes, where data generated by τ_i is used by τ_j. Note that the buffer for inter-node communication is overwritten each time new data arrives so that only the latest data is preserved. There are two types of edges:

trigger edges – edges indicating that the successor *event-driven nodes* are triggered when data from the predecessor node arrives, denoted by $e^{tr}_{i,j}$

update edges – edges indicating that the data received from predecessor nodes are stored in memory at that time and read out when successor nodes are triggered, denoted by $e^{up}_{i,j}$.

Among a pair of nodes connected by *update edges*, the successor node is either a *timer-driven node* or an *event-driven node* that is activated by the *trigger edge* from another predecessor node. $comm_{i,j}$ represents the worst-case communication time of each edge $e_{i,j}$. While considering variations in both node execution time and inter-node communication time would be preferable, this paper assumes worst-case communication times for the simplicity of the problem.

As an example of these notations, the nodes and edges in Figure 1 are classified as follows. The nodes $\{\tau^{td}_1, \tau^{td}_2, \tau^{td}_3, \tau^{td}_8\}$ are *timer-driven nodes*, the nodes $\{\tau^{ed}_4, \tau^{ed}_5, \tau^{ed}_6, \tau^{ed}_7, \tau^{ed}_9\}$ are *event-driven nodes*, the edges $\{e^{tr}_{1,4}, e^{tr}_{2,5}, e^{tr}_{3,6}, e^{tr}_{4,7}, e^{tr}_{8,9}\}$ are *trigger edges*, and the edges $\{e^{up}_{5,8}, e^{up}_{6,8}, e^{up}_{7,9}\}$ are *update edges*.

2.2 Probabilistic Execution Time

This paper treats the execution time of each node τ_i as a random variable X_i in order to reduce pessimism in the early detection of deadline misses. The execution time X_i takes various values probabilistically, and the probability distribution of X_i is represented as $\mathbb{X}_i = (x_i, p_i)$. Here, $x_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,l}\}$ is an ascending list, dividing the range of possible execution times for τ_i into $l-1$ intervals. $p_i = \{p_{i,1}, p_{i,2}, \ldots, p_{i,l}\}$ is a list of probabilities for τ_i to take corresponding execution time (i.e., $p_{i,m} = P(x_{i,m-1} < X_i \leq x_{i,m})$). Note that $x_{i,0} = 0$, $x_{i,l}$ is WCET of τ_i, and $\sum_{p_i} p_{i,m} = 1$. For τ_1 in Figure 2, $P((0 <)\ X_i \leq 10) = 0.8, P(10 < X_i \leq 15) = 0.2$. In this study, probability distributions for nodes are generated based on statistics obtained by running an actual autonomous driving system, so these distributions are similar to that of the real system, and these distributions are assumed to be independent of each other.

To verify whether the system satisfies the time constraints, it is essential to calculate the total execution time of a sequence of multiple nodes. However, probabilistically varying values can not simply be added together, unlike the fixed value. Therefore, convolution is introduced to consider the sum of probabilistic execution times as a random variable. For the probabilistic execution times X_i, X_j, the convolution is expressed as $X_i \otimes X_j$ and calculated as follows:

$$P(Y = y) = \sum_{m=1}^{l} P(X_i = x_{i,m}) \cdot P(X_j = y - x_{i,m}), \tag{1}$$

where Y is the probabilistic total execution time, and X_i and X_j are independent of each other. A simplified example of this operation is illustrated in Figure 2.

Table 2 Problem model notations.

Symbol	Description
sg_i	Subgraph
$head(sg_i)$	Index of the head *timer-driven node* of sg_i
T_{sg_i}	Period of sg_i
$T_{sg_{exit}}$	Period of the subgraph that includes exit node
$j\tau_i$	*Join node*
$t\tau_i$	*Tail node*
$sub(t\tau_i)$	Index of the subgraph that contains $t\tau_i$
$pred^{tr}(\tau_i)$	Set of nodes with a *trigger edge* to τ_i
$DFC(t\tau_i)$	*Data freshness* constraint of the data generated by $t\tau_i$
$data(t\tau_i, k)$	Data generated by the *tail node* job $\langle t\tau_i, k\rangle$
$stamp[data(t\tau_i, k)]$	Timestamp of $data(t\tau_i, k)$
D	End-to-end deadline
d_k	Deadline of k^{th} job of exit node
HP	Hyper-period of a DAG
JLD	Set of all the job-level dependencies in HP
$(\langle\tau_i, k\rangle \rightarrow \langle\tau_j, s\rangle)$	Job-level dependency from $\langle\tau_i, k\rangle$ to $\langle\tau_j, s\rangle$
N_{sg_i}	Number of times sg_i is executed during HP
$RST(\tau_i, k)$	Reference start time of $\langle\tau_i, k\rangle$
$RFT(\tau_i, k)$	Reference finish time of $\langle\tau_i, k\rangle$

3 Problem Model

In an autonomous driving system, time constraints must be satisfied. To meet these constraints, we need to analyze the dependencies among jobs, and finally to find the relationship between the start time of each job and the probability of a deadline miss. However, the timing analysis of a DAG with a mixture of nodes that are activated at different periods is a complex problem [2]. Therefore, in Section 3.1, the DAG is divided into subgraphs that are driven by a single period (i.e., only the first node is timer-driven). We then define the constraints to be satisfied in Section 3.2, and determine the job-level dependencies in Section 3.3, following the approach of Ref. [19]. The notations for the problem model are listed in Table 2.

3.1 Dividing DAG into Subgraphs

A DAG that includes *timer-driven nodes* with different periods is divided into subgraphs with single-period, denoted by sg_i. As mentioned above, the subgraph consists of a single *timer-driven node* and zero or more *event-driven nodes* that are directly/indirectly triggered by the *timer-driven* one (i.e., $sg_i = \{\tau_a^{td}, \tau_b^{ed}, \ldots, \tau_c^{ed}\}$). The first nodes in the sequences are triggered according to their own period, and subsequent nodes are sequentially triggered in a chain. Therefore, the period of a subgraph is the same as the head *timer-driven node*. The period of the subgraph sg_i is denoted by T_{sg_i} and given as follows:

$$T_{sg_i} = T_{head(sg_i)}, \tag{2}$$

where $head(sg_i)$ is a function that returns the index of the head *timer-driven node* of sg_i.

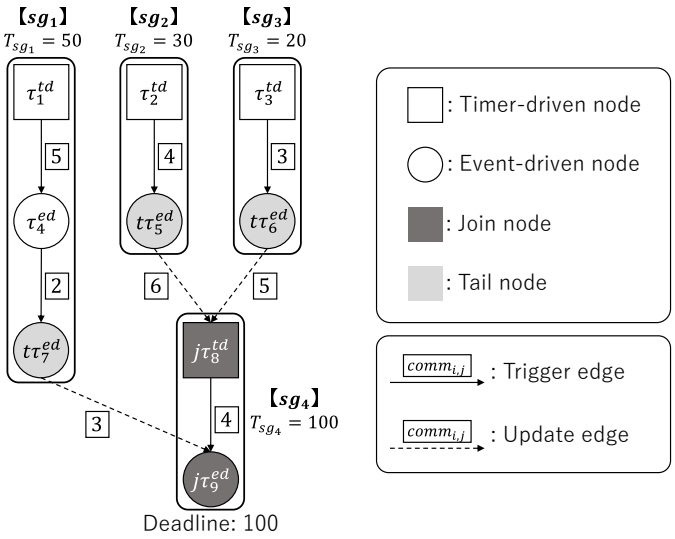


Figure 3 Dividing the example DAG into a set of subgraphs.

The result of dividing the example DAG in Figure 1 into a set of subgraphs is shown in Figure 3. The subgraph sg_1 consists of one *timer-driven node* τ_1^{td} and two *event-driven nodes* τ_4^{ed}, τ_7^{ed}. Then, the period of sg_1 is $T_{sg_1} = T_{head(sg_1)} = T_1 = 50$ according to Equation (2).

The transmissions of data across subgraphs are particularly important for analyzing job-level dependencies, as described in more detail later. Therefore, the nodes involved in the transfer of such data are defined:

Join nodes – nodes that receive data from nodes in multiple different subgraphs, denoted by $jt\tau_i$

Tail nodes – nodes that have at least one edge to a *join node* in other subgraphs, denoted by $t\tau_i$.

$j\tau_8^{td}$ and $j\tau_9^{ed}$ are *join nodes*, and $t\tau_5^{ed}$, $t\tau_6^{ed}$ and $t\tau_7^{ed}$ are *tail nodes*, in Figure 3. Note that nodes can be both *join nodes* and *tail nodes*. For instance, in Figure 3, if $j\tau_9^{ed}$ has an edge to a node in another subgraph, $j\tau_9^{ed}$ is both a *join node* and a *tail node*.

For the model addressed in this paper, all nodes in a DAG are included in one of the subgraphs without duplication. That is, even if an *event-driven node* has multiple predecessor nodes in different subgraphs, the node is triggered by predecessor nodes in the same subgraph to which the node belongs. Therefore, any edge connecting a *tail node* to another node in another subgraph is an *update edge*.

3.2 Definitions of Constraints

An autonomous driving system must satisfy the following two constraints: (i) *data freshness* constraint and (ii) end-to-end deadline. These two constraints are defined in this section.

Since the surroundings of an autonomous vehicle change constantly, using too old data can lead to less accurate self-localization and object perception [8, 17]. This loss of accuracy is so undesirable for the safety that each node in an autonomous driving system must use sufficiently fresh data. To ensure this, the *data freshness* constraint is imposed on the system as an acceptable staleness of data used by each *join node*. When a *join node* is triggered, if the data from all predecessor nodes that satisfy the *data freshness* constraint are available, that *join node* can be executed. Here, *data freshness* is the time duration since the timestamp

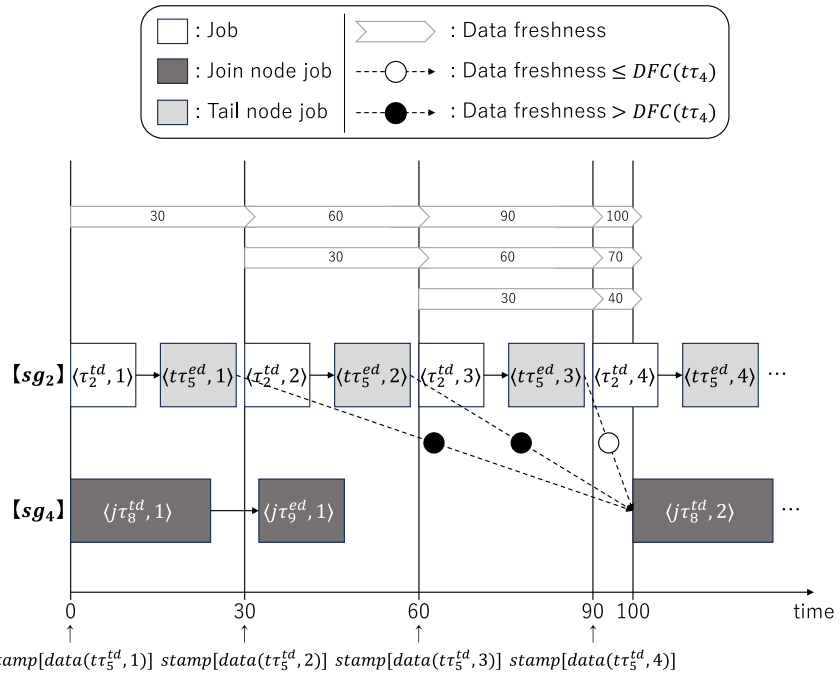


Figure 4 The *data freshness* constraint for $t\tau_5^{ed}$ and $j\tau_8^{td}$.

labeled on the data until the data is used. The timestamp of data generated by sensors of the system, which are the entry nodes of the DAG, is the execution start time of that entry node. The timestamp is not updated by regular successor nodes, but is carried over until the data is used and integrated with other data by a *join node*. *Join nodes* update the timestamp of the data they generate to their own execution start times, and the timestamp is carried over until the data is used by another *join node*. The *data freshness* constraint of the data output from a *tail node* $t\tau_i$ is denoted by $DFC(t\tau_i)$ and is defined as follows:

$$DFC(t\tau_i) = \alpha \times T_{sg_{sub(t\tau_i)}}, \tag{3}$$

where $\alpha \in \mathbb{R}^+$ and $T_{sg_{sub(t\tau_i)}}$ is the period of the subgraph that contains $t\tau_i$. As evident from Equation (3), $DFC(t\tau_i)$ is a positive real multiple of $T_{sg_{sub(t\tau_i)}}$. Here, the value of α can be set flexibly according to the system requirements. The smaller value of α results in a shorter valid data duration, making the *data freshness* constraint stricter, while the larger value of α makes the *data freshness* constraint more lenient.

An example of *data freshness* constraint for $t\tau_5^{ed}$ and $j\tau_8^{td}$ in Figure 3 when $\alpha = 2$, as shown in Figure 4. Here, the data output from the *tail node* job $\langle t\tau_i, k\rangle$ is denoted as $data(t\tau_i, k)$, and the timestamp of $data(t\tau_i, k)$ is denoted as $stamp[data(t\tau_i, k)]$. $DFC(t\tau_5)$ is 60 ms according to Equation (3). Since $stamp[data(t\tau_5, 1)]$ is carried over from $stamp[data(\tau_2, 1)]$, which is the execution start time of $\langle\tau_2, 1\rangle$, $stamp[data(t\tau_5, 1)]$ is 0 ms. Therefore, jobs of $j\tau_8$ that start execution by $stamp[data(t\tau_5, 1)] + DFC(t\tau_5) = 60$ ms can use $data(\tau_5, 1)$. Since $\langle j\tau_8, 2\rangle$ starts execution at 100 ms, $\langle j\tau_8, 2\rangle$ cannot use $data(t\tau_5, 1)$. Similarly, since $stamp[data(t\tau_5, 2)] + DFC(t\tau_5) = 90$ ms, $\langle j\tau_8, 2\rangle$ cannot use $data(t\tau_5, 2)$ too. Meanwhile, since $stamp[data(t\tau_5, 3)] + DFC(t\tau_5) = 120$ ms, $\langle j\tau_8, 2\rangle$ can use $data(t\tau_5, 3)$.

To guarantee the real-time performance of the system, an end-to-end deadline constraint is also imposed. The end-to-end deadline constrains the total time from input to output of the system. Let D denote the end-to-end deadline given to the exit node, and d_k denote the

deadline given to the k^{th} job of the exit node. In the example in Figure 3, an end-to-end deadline of 100 ms is assigned to the exit node τ_9^{ed}. The deadline for the first job of the exit node d_1 is D, and the deadlines for the second and subsequent jobs are shifted by their activation times. Here, since the activation interval of the exit node follows the period of the subgraph containing the exit node, d_k can be calculated as follows:

$$d_k = D + (k-1) \times T_{sg_{exit}}, \tag{4}$$

where $T_{sg_{exit}}$ is the period of the subgraph containing the exit node (i.e., the period of the head *timer-driven node* in the subgraph).

3.3 Determining Job-level Dependencies

An autonomous driving system must complete the execution of the jobs of the exit node by the end-to-end deadline while satisfying the *data freshness* constraints for each job. To this end, it is essential to analyze the relationship named job-level dependency, which jobs have "valid" data exchanges, meaning data that meets the *data freshness* constraint output by one job is used by another job.

In order to cover the job-level dependencies in all cases, an interval that is the least common multiple (LCM) of the periods of all subgraphs in the DAG (i.e., the periods of all *timer-driven nodes*) must be considered. This is because the periods of the subgraphs resulting from the division of the DAG are different from each other. Such interval is the hyper-period of the DAG, denoted as *HP*, and calculated as follows:

$$HP = \text{LCM}_{\forall \tau_i^{td} \in V^{td}}(T_i). \tag{5}$$

To analyze job-level dependencies, the first step is to calculate reference values of the start and finish times of each job. This operation calculates the times at which data is read and written, and investigates which data satisfies the *data freshness* constraint. Therefore, by calculating the reference start times and reference finish times for all jobs within *HP*, all job-level dependencies can be derived. The reference start times and reference finish times for all jobs (of *timer-driven nodes* and *event-driven nodes*) can be calculated according to Equations (5)-(8) in Ref. [19]. Here, the reference start time and reference finish time of a job $\langle \tau_i, k \rangle$ are denoted by $RST(\tau_i, k)$ and $RFT(\tau_i, k)$, respectively.

Using the reference start times and reference finish times of each job, determine all job-level dependencies in *HP*. This allows us to understand the flow of data to each job of the exit node and to identify the jobs involved in that flow. Finally, the start times of these jobs can be used to estimate the probabilities of deadline misses at runtime.

The set of all job-level dependencies within *HP* is denoted by JLD, and the job-level dependency from $\langle \tau_i, k \rangle$ to $\langle \tau_j, s \rangle$ by $(\langle \tau_i, k \rangle \rightarrow \langle \tau_j, s \rangle) \in JLD$. The job-level dependencies in each subgraph are simply obtained, as the node-level dependencies are directly job-level dependencies. That is, if an edge from τ_i to τ_j within the subgraph exists, then $(\langle \tau_i, k \rangle \rightarrow \langle \tau_j, k \rangle)$ for any k. An example is provided with $sg_2 = \{\tau_2^{td}, \tau_5^{ed}\}$ in Figure 3. *HP* of this DAG is 300 ms, as derived from Equation (5). Here, the number of times sg_i is executed during *HP* is denoted as N_{sg_i}, and calculated as follows:

$$N_{sg_i} = \frac{HP}{T_{sg_i}}. \tag{6}$$

Then, $N_{sg_2} = \frac{300}{30} = 10$, and $\{(\langle \tau_2, 1 \rangle \rightarrow \langle \tau_5, 1 \rangle), (\langle \tau_2, 2 \rangle \rightarrow \langle \tau_5, 2 \rangle), \ldots, (\langle \tau_2, 10 \rangle \rightarrow \langle \tau_5, 10 \rangle)\}$ is added to JLD.

Table 3 Notations for the proposed method.

Symbol	Description
$L_{i,k}$	*plaxity* of $\langle \tau_i, k \rangle$
$\mathbb{L}_{i,k}$	Probability distribution of $L_{i,k}$
$L^{\mathrm{c}}_{i,k}$	*plaxity-cdf* of $\langle \tau_i, k \rangle$
$\mathbb{L}^{\mathrm{c}}_{i,k}$	Cumulative distribution of $L_{i,k}$
$AST(\tau_i, k)$	Actual start time of $\langle \tau_i, k \rangle$
$succ(\tau_i, k)$	Set of jobs with a job-level dependency from $\langle \tau_i, k \rangle$

In contrast, job-level dependencies between different subgraphs, i.e., from jobs of *tail nodes* to jobs of *join nodes*, are not as simple. Job-level dependencies from jobs of *tail node* to jobs of *join node* are defined based on the reference start times, reference finish times, and *data freshness* constraints, as follows:

▶ **Definition 1** ([19]). *Suppose that $e_{\tau_i,\tau_j} \in E$ and $t\tau_i$ is a tail node, and $j\tau_j$ is a join node. There exists a job-level dependency from $\langle t\tau_i, k \rangle$ to $\langle j\tau_j, s \rangle$, that is, if*

1. $RFT(t\tau_i, k) + comm_{i,j} \leq RST(j\tau_j, s)$ *and*
2. $RST(j\tau_j, s) - stamp[data(t\tau_i, k)] \leq DFC(t\tau_i)$.

Here, let $\langle t\tau_i, k \rangle \in sg_i$, then $stamp[data(t\tau_i, k)]$ is given by:

$$stamp[data(t\tau_i, k)] = RST(\tau_{head(sg_i)}, k), \tag{7}$$

where $\tau_{head(sg_i)}$ is the head *timer-driven node* of the subgraph containing the job $\langle t\tau_i, k \rangle$, and $RST(\tau_{head(sg_i)}, k)$ is the reference start time for the job of that head node. Condition 1 of Definition 1 requires that the data from the job of the *tail node* has arrived at the reference start time for the job of the *join node*, and condition 2 of Definition 1 requires that the data satisfy the *data freshness* constraint.

4 Proposed Method

In the existing method for deadline miss early detection, a time constraint is given to each job based on the end-to-end deadline and job-level dependencies determined in Section 3 and on WCET of each node. The time constraint is a threshold called *laxity* whose value represents the slack time for each job to the end-to-end deadline. That is, if the execution start time of a job exceeds the *laxity*, the data that satisfies the *data freshness* constraint will not arrive soon enough at the successor job, and a deadline miss is predicted as a result. The *laxity* of each job can be calculated by recursively subtracting the execution time of the predecessor job and the communication time between jobs from the deadline given to the job of the exit node. When the execution time of each job is a fixed value (WCET), the calculation of *laxity* is straightforward. However, when considering that the execution time of each job varies, simple subtraction is not applicable. This paper proposes a method for calculating the *laxity* of each job for a DAG in which each node has a probabilistic execution time to consider the variations in execution time. The notations for this section are presented in Table 3.

The *laxity* calculated using the execution times represented as random variables is also a random variable, and this "probabilistic" *laxity* is referred to as *plaxity*. The *plaxity* of each job $\langle \tau_i, k \rangle$ is denoted as $L_{i,k}$, and the probability distribution of $L_{i,k}$ is denoted as

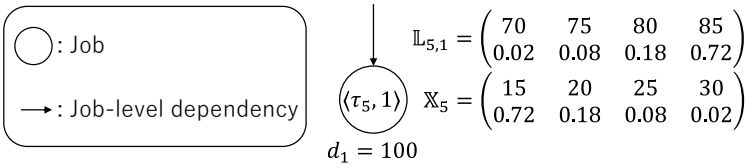


Figure 5 *plaxity* of the exit node job.

$\mathbb{L}_{i,k} = (\lambda_{i,k}, \ell p_{i,k})$. Here, $\{\lambda_{i,k,1}, \lambda_{i,k,2}, \ldots, \lambda_{i,k,u}\}$ is an ascending list of possible values for the *plaxity* of the job $\langle \tau_i, k \rangle$, and $\{\ell p_{i,k,1}, \ell p_{i,k,2}, \ldots, \ell p_{i,k,u}\}$ is probabilities for the corresponding *plaxity* values. For example, consider the following $\mathbb{L}_{i,k}$:

$$\mathbb{L}_{i,k} = \begin{pmatrix} \lambda_{i,k} \\ \ell p_{i,k} \end{pmatrix} = \begin{pmatrix} 70 & 75 & 80 & 85 \\ 0.02 & 0.08 & 0.18 & 0.72 \end{pmatrix}.$$

In this case, $L_{i,k}$ is 70 with a probability of 0.02, 75 with 0.08, 80 with 0.18, and 85 with 0.72. Let $AST(\tau_i, k)$ denote the actual start time of the job $\langle \tau_i, k \rangle$, when $75 < AST(\tau_i, k) \leq 80$, $P(L_{i,k} < AST(\tau_i, k)) = 0.02 + 0.08 = 0.1$, and $P(AST(\tau_i, k) \leq L_{i,k}) = 0.18 + 0.72 = 0.9$. In other words, based on the definition of *laxity*, the probability of missing the deadline is 0.1, and that of meeting the deadline is 0.9. The threshold for the probability of meeting the deadline is determined according to user and system requirements, and if the probability of meeting the deadline falls below the threshold, a deadline miss is detected. However, as calculated above, the probability of meeting the deadline cannot be obtained directly by comparing $AST(\tau_i, k)$ and $L_{i,k}$. Therefore, *plaxity-cdf* is defined as the cumulative probability form of *plaxity*, denoted as $L^{\mathrm{c}}_{i,k}$, and the distribution of $L^{\mathrm{c}}_{i,k}$ is denoted as $\mathbb{L}^{\mathrm{c}}_{i,k} = (\lambda_{i,k}, \ell p^{\mathrm{c}}_{i,k})$. Here, for all integers v with $1 \leq v \leq u$, $\ell p^{\mathrm{c}}_{i,k,v} \in \ell p^{\mathrm{c}}_{i,k}$ is given as follows:

$$\ell p^{\mathrm{c}}_{i,k,v} = \sum_{h=v}^{u} \ell p_{i,k,h}. \tag{8}$$

By Equation (8), the cumulative probability form of $\mathbb{L}_{i,k}$, the previous example, is calculated as follows:

$$\mathbb{L}^{\mathrm{c}}_{i,k} = \begin{pmatrix} \lambda_{i,k} \\ \ell p^{\mathrm{c}}_{i,k} \end{pmatrix} = \begin{pmatrix} 70 & 75 & 80 & 85 \\ 1 & 0.98 & 0.9 & 0.72 \end{pmatrix}.$$

By comparing $AST(\tau_i, k)$ and $\mathbb{L}^{\mathrm{c}}_{i,k}$, the probability of meeting the deadline is directly obtained from the start time of the job. Formally, for $AST(\tau_i, k)$ such that $\lambda_{i,k,v-1} < AST(\tau_i, k) \leq \lambda_{i,k,v}$, the probability of meeting the deadline is $\ell p^{\mathrm{c}}_{i,k,v}$. Note that $\lambda_{i,k,0} = 0, \lambda_{i,k,u+1} = \infty$, and $\ell p^{\mathrm{c}}_{i,k,u+1} = 0$. The *plaxity-cdf* is used to compare with the start time of the job, and is not directly involved in the calculation of the *plaxity* of each job. The *plaxity* alone is primarily used for the calculation of the *plaxity* of each job.

In the following, we define the method for calculating the *plaxity* for each job. As with *laxity*, the procedure is to calculate the *plaxity* recursively, starting from the job of the exit node. Therefore, the calculation method for *plaxity* of the job of the exit node is defined first. The possible values for the *plaxity* of the exit node job are obtained by subtracting the possible execution times of the job from the deadline. The probability of each *plaxity* value corresponds to the probability of the job taking respective execution time. Then, the *plaxity* $L_{i,k}$ of the exit node job $\langle \tau_i, k \rangle$ is defined as follows:

$$P(L_{i,k} = \lambda) = P(X_i = d_k - \lambda). \tag{9}$$

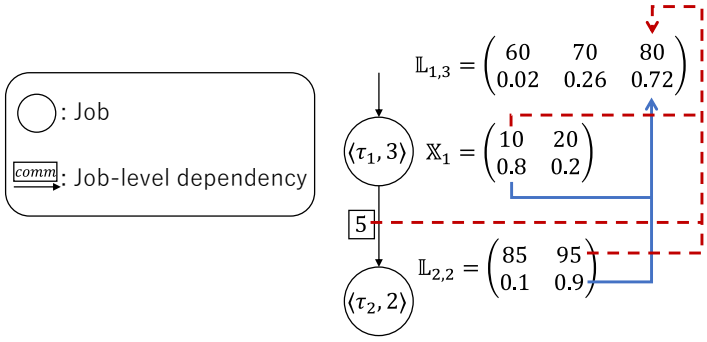


Figure 6 *plaxity* of the job that has one successor job.

An example of calculating the *plaxity* $L_{5,1}$ of the exit node job $\langle\tau_5, 1\rangle$ in Figure 5 is shown below. Using $\mathbb{X}_5, d_k$, and Equation (9),

$$\begin{aligned} & P(L_{5,1} = 85) = P(X_5 = 15) = 0.72, \\ & P(L_{5,1} = 80) = P(X_5 = 20) = 0.18, \\ & P(L_{5,1} = 75) = P(X_5 = 25) = 0.08, \\ \text{and} \quad & P(L_{5,1} = 70) = P(X_5 = 30) = 0.02 \end{aligned}$$

are obtained. Therefore, $\mathbb{L}_{5,1}$ is

$$\mathbb{L}_{5,1} = \begin{pmatrix} 70 & 75 & 80 & 85 \\ 0.02 & 0.08 & 0.18 & 0.72 \end{pmatrix}.$$

Next, the *plaxity* of jobs that have successor jobs is defined. For the sake of clarity, jobs with only one successor job are considered. The *plaxity* of jobs with a successor job can be defined by applying the convolution operation in Equation (1). The possible values of the *plaxity* of the job with a successor job are obtained by subtracting the communication time between the jobs and the possible execution time of the predecessor job from the *plaxity* values of the successor job. The probability of each *plaxity* value corresponds to the joint probability of the predecessor job taking the respective execution time and the successor job having the corresponding *plaxity* value. Since these random variables are independent of each other, the joint probability is calculated as the product of their probabilities. Then, we define the calculation called *convolutional subtraction*, applying Equation (1). Here, the operation of deriving the *plaxity* of the job $\langle\tau_i, k\rangle$ with only one successor job $\langle\tau_j, s\rangle$ by convolutional subtraction is expressed as $L_{i,k} = L_{j,s} \otimes^{-} X_i$, and defined as follows:

$$P(L_{i,k} = \lambda) = \sum_{m=1}^{l} P(X_i = x_{i,m}) \cdot P(L_{j,s} = \lambda + comm_{i,j} + x_{i,m}). \tag{10}$$

Note that this operation takes into account the communication time between jobs. An example of calculating the *plaxity* $L_{1,3}$ of the job $\langle\tau_1, 3\rangle$ is illustrated in Figure 6. Using $\mathbb{X}_1, comm_{1,2}, \mathbb{L}_{2,2}$, and Equation (10),

$$\begin{aligned} & P(L_{1,3} = 60) = P(X_1 = 20) \cdot P(L_{2,2} = 85) = 0.02, \\ & P(L_{1,3} = 70) = P(X_1 = 10) \cdot P(L_{2,2} = 85) \\ & \qquad\qquad + P(X_1 = 20) \cdot P(L_{2,2} = 95) = 0.26, \\ \text{and} \quad & P(L_{1,3} = 80) = P(X_1 = 10) \cdot P(L_{2,2} = 95) = 0.72 \end{aligned}$$

are obtained. Therefore, $\mathbb{L}_{1,3}$ is

$$\mathbb{L}_{1,3} = \begin{pmatrix} 60 & 70 & 80 \\ 0.02 & 0.26 & 0.72 \end{pmatrix}.$$

Finally, the *plaxity* of jobs that have multiple successor jobs is defined. In the case of *laxity*, to satisfy the deadline of all subsequent paths, the minimum value of the *laxity* among successor jobs is used. However, in the case of *plaxity*, since the direct comparison is not possible, determining the minimum value requires further consideration. Suppose that two jobs, each with a different *plaxity*. Define a new *plaxity* with the probability that both of these jobs meet their deadlines when they start at the same time t. Such probability is the joint probability of each job independently meeting the deadline from t. Since the *plaxity* of each job are independent, this joint probability can be calculated as the product of their probabilities. Therefore, the minimum value L of two different *plaxity*, $L_{i,k}$ and $L_{j,s}$ is given by:

$$\begin{aligned} \min(L_{i,k}, L_{j,s}) &= L; \\ P(L = \lambda) = P(L_{i,k} = \lambda) \cdot P(\lambda \leq L_{j,s}) &+ P(\lambda < L_{i,k}) \cdot P(L_{j,s} = \lambda). \end{aligned} \tag{11}$$

The minimum value of *plaxity* can be obtained for three or more jobs, by repeatedly applying this calculation. Thus, the *plaxity* $L_{i,k}$ of the job $\langle \tau_i, k \rangle$ with multiple successor jobs can be calculated by Equations (10) and (11), as follows:

$$L_{i,k} = \min_{\langle \tau_j, s \rangle \in succ(\tau_i, k)} \left(L_{j,s} \otimes^{-} X_i \right). \tag{12}$$

The *plaxity* of all jobs can be calculated by Equations (9) and (12). Then applying Equation (8) to the determined *plaxity* immediately yields the *plaxity-cdf*. During runtime, the actual start time of the job and the *plaxity-cdf* are compared, and if the probability of meeting the deadline is below the threshold, a deadline miss is detected.

5 Evaluation

The performance of the proposed method for early detection of deadline miss is evaluated through experiments which are performed by simulating task scheduling for multiple periodic DAGs.

5.1 Experiment Setup

The experimental targets for performing evaluation are DAGs randomly generated using RD-Gen [20], with modifications to fit the model considered in this paper. The period of *timer-driven nodes* and communication time are set to default values. First, *trigger edges* and *update edges* are defined. Since *timer-driven nodes* are activated at their own periods, all inputs to *timer-driven nodes* are *update edges*, and no *trigger edges* are input. For *event-driven nodes*, if they have only one input edge, that edge is a *trigger edge*. In the case of multiple input edges, the type of edge is determined based on the *chain* given by RD-Gen. Only the edges from nodes in the same chain are considered *trigger edges*, while inputs from other nodes are considered *update edges*. When determining the edge type in this way, note that an *event-driven node* can have multiple *trigger edges*. Such *event-driven nodes* cannot begin execution until all predecessor nodes connected by *trigger edges* have completed. Next, the execution time for each node is set based on the given

WCET. The average is set as $\mu = WCET/3.0$, and the standard deviation is $\sigma = \mu/2.0$. The execution time follows a normal distribution $\mathcal{N}(\mu, \sigma)$ with a probability of 0.98, and a normal distribution $\mathcal{N}(WCET, 0.2\sigma)$ with a probability of 0.02. This setting is empirically derived from the characteristics of *Autoware* runtimes measured by CARET [9].

The scheduling algorithm adopted is Earliest Deadline First (EDF) [16]. That is, jobs with shorter time to the deadline are given higher priority for execution. Here, implicit deadlines are assumed, and the relative deadline for each job is equal to the period of the corresponding node. The relative deadline of *event-driven nodes* that do not have their own period is assumed to be equal to the period of the subgraph they belong to. Additionally, each job is executed non-preemptively, and once assigned to a processor, the job will not be interrupted until completion.

The simulations are written in Python and executed on a system with 2.9 GHz 8-Core Intel Core i7 CPU, 32 GiB RAM, and Ubuntu 22.04 LTS (64-bit) OS.

5.2 Performance Metrics

This section describes the metrics for evaluation. First, the classification of simulation results needed to calculate performance metrics is listed:

- *True Positive (TP):* TP is the case where a deadline miss is detected early by the proposed method, and an actual deadline miss occurs when the processing has progressed. This result indicates that the early detection of deadline miss by the proposed method is accurate.
- *False Positive (FP):* FP is the case where a deadline miss is detected early by the proposed method, but no deadline miss actually occurs when processing has progressed. This result indicates that the deadline miss detection by the proposed method is pessimistic. In the context of autonomous driving systems, a higher occurrence of FP results implies that the system often transitions to the MRM mode even when no actual deadline miss occurs, which hinders smooth operation.
- *True Negative (TN):* TN is the case where a deadline miss is not detected early, and no deadline miss actually occurs when processing has progressed. This result means that the system can correctly predict that no deadline miss will occur. In safety-critical real-time systems, the occurrence of deadline misses is expected to be rare. Therefore, if the accuracy of early detection of deadline miss is high, most results will be TN when the system is in a safe state.
- *False Negative (FN):* FN is the case where a deadline miss is not detected early, but an actual deadline miss occurs when processing has progressed. This result indicates that the proposed method failed to detect the deadline miss early.

Using the counts of these results, the following performance metrics to evaluate the proposed method are calculated:

- **Accuracy:** The *Accuracy* represents the proportion of accurate predictions among the total number of results of deadline miss prediction and is defined as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{13}$$

As evident from the formula, the smaller FP and FN are, the closer *Accuracy* approaches 1. When FP and FN are both 0, i.e., a deadline miss always occurs when a deadline miss is predicted, and all actual deadline misses are detected early, *Accuracy* is exactly 1.

Table 4 Experimental parameters in Section 5.3.

Parameter	Value
Number of cores	8
Normalized utilization [%]	275, 280, 285, 290, 295, 300, 305[1]
α in Equation (3)	2.0, 2.1, 2.2, 2.3, 2.4, 2.5
Number of entry nodes	7, 8, 9
Number of exit nodes	1
Period of *timer-driven nodes* [ms]	10, 20, 30, 50, 60, 100
Probability threshold	1, 0.99, 0.95, 0.90

- **Recall:** The *Recall* represents the proportion of predicted deadline misses among all actual deadline misses and is defined as follows:

$$Recall = \frac{TP}{TP + FN} \tag{14}$$

The higher value of the *Recall*, the more reliably deadline misses are detected. From the standpoint of safety, the value close to 1 is desirable.

- **Precision:** The *Precision* is the ratio of accurately predicted deadline misses to the total number of deadline miss predictions and is defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{15}$$

A higher *Precision* value indicates that a higher proportion of early-detected deadline misses actually result in deadline misses, which is crucial for avoiding unnecessary disruptions.

- **F-measure:** The *F-measure* is the harmonic mean of the *Recall* and the *Precision*, and is derived as follows:

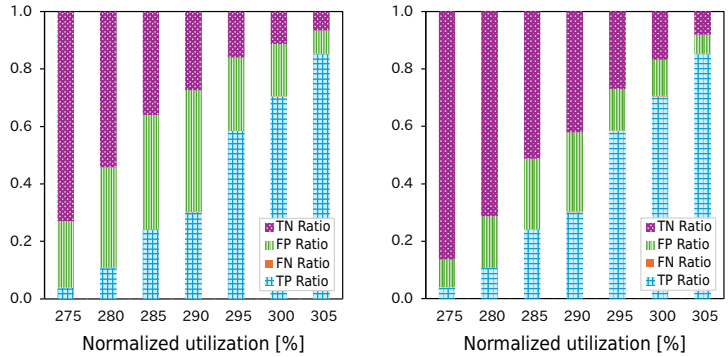
$$F\text{-}measure = 2 \cdot \frac{Recall \cdot Precision}{Recall + Precision} \tag{16}$$

As mentioned above, the higher *Recall* and *Precision*, the better the result. However, they are in a trade-off relationship. Therefore, the balance between the *Recall* and the *Precision* is evaluated using the *F-measure* index.

5.3 Comparison of Detection Results with Different Thresholds

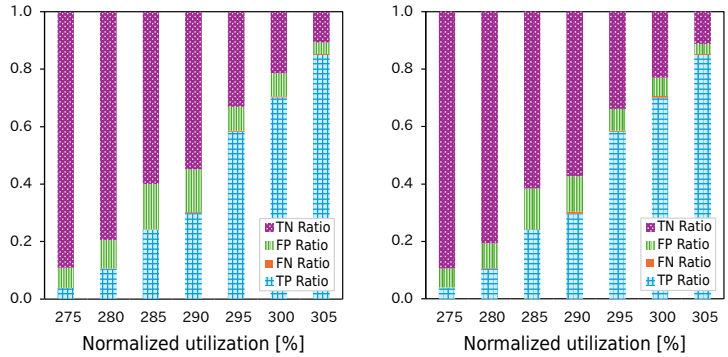
In this section, the performance of deadline miss early detection using different probability thresholds is examined. Here, how the probability threshold works is explained. The early detection of deadline miss is conducted at the beginning of job execution by referring to the *plaxity-cdf*. In practice, the largest *plaxity-cdf* value with a probability greater than or equal to the given threshold (i.e., quantile for the threshold) is compared with the execution start time. If the latter is greater, a deadline miss is detected. That is, if the probability of meeting the deadline at execution start time is greater than the specified threshold, a

[1] To compare the performance of deadline miss predictions, it is necessary to consider the situation in which a deadline miss occurs, i.e., when the system is overloaded. In this experimental setting, the normalized utilization is defined using WCET, although the job execution time is extremely smaller than WCET. Therefore, tasks with normalized utilization that are well above 100% are considered, for the occurrence of deadline misses.



(a) Threshold: 1.00.

(b) Threshold: 0.99.



(c) Threshold: 0.95.

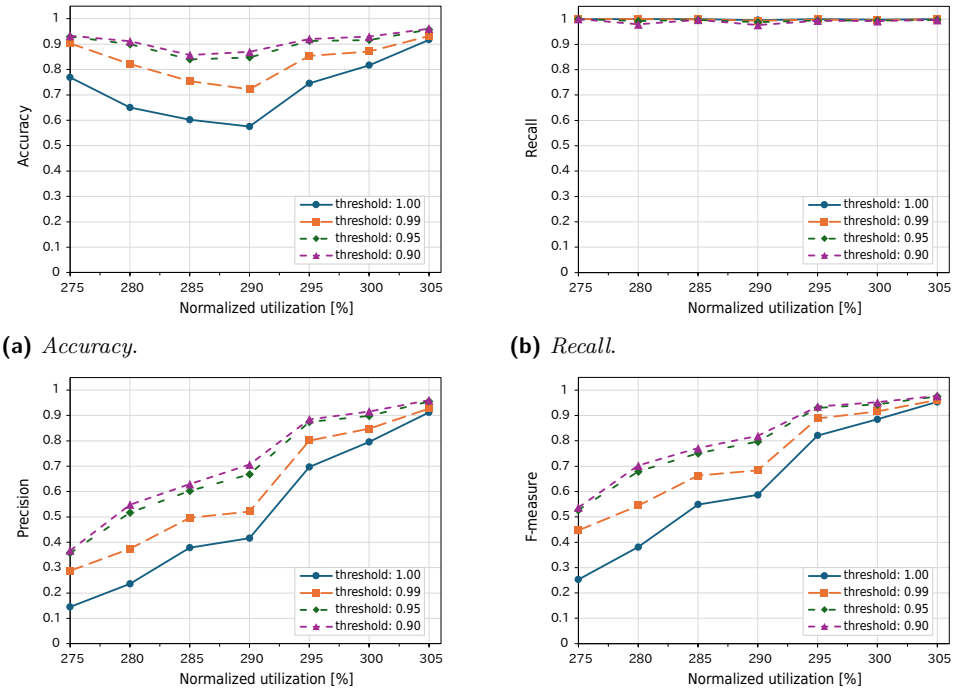
(d) Threshold: 0.90.

Figure 7 TP, FP, TN, and FN ratios with increasing utilization.

deadline miss is not detected. Therefore, setting a high probability threshold results in a more strict deadline miss prediction, while a low threshold leads to a more lenient prediction. When the threshold is set to 1, the results are the same as for the conventional method, which calculates *laxity* using WCET.

For each probability threshold, simulations are performed with increasing task utilization, and the results obtained are evaluated using defined performance metrics. The parameters used in this section are listed in Table 4. The normalized utilization is the total utilization of the task divided by the number of processor cores. The task utilization is based on the assumption of WCET. Note that the normalized utilization ratio above 100% does not mean that the system is immediately overloaded, since the actual execution time is extremely small compared to WCET in this paper. In this experiment, it is necessary to increase the utilization to a level where deadline misses occur for comparison, which is greater than 100%.

Changes in TP, FP, TN, and FN with increasing utilization are shown in Figure 7. For each utilization (i.e., each point on the x-axis of figures), 500 randomly-generated DAGs were run 10 times each. Note that the execution time of nodes differs from run to run for the same DAG. As the utilization rate increases, more deadline misses occur. Then, the fact that decreasing the threshold results in fewer occurrences of FP is clear. This implies a reduction in the number of instances where a deadline miss is predicted despite not actually missing a deadline, thereby mitigating pessimism. However, it is important to note that as the probability threshold decreases and FP decreases, FN increases. This can be easily understood from the fact that there is a trade-off between reducing the number of useless



(a) *Accuracy.* **(b)** *Recall.*

(c) *Precision.* **(d)** *F-measure.*

Figure 8 *Accuracy*, *Recall*, *Precision* and *F-measure* results with increasing utilization.

deadline miss predictions and detecting deadline errors without missing them. Then, the *Accuracy*, *Precision*, *Recall*, and *F-measure* are calculated from these values of TP, FP, TN, and FN.

The proposed method is clearly superior with respect to the *Accuracy* as shown in Figure 8(a). However, setting the probability threshold to 1 corresponds to the conventional method using WCET. Higher *Accuracy* is obtained by setting the threshold smaller than 1. For the thresholds shown in the figure, the decrease of FP is greater than the increase of FN with decreasing threshold, so that smaller thresholds result in higher *Accuracy*.

The *Recall* with each threshold as utilization increases is shown in Figure 8(b). The *Recall* is an indicator of the ability to detect deadline misses without omission, and is highly relevant to the safety. At least when the threshold is set to 0.99, the *Recall* is maintained at almost 1. When the threshold is set to 0.9, the *Recall* is reduced by up to five percent. Although lowering the threshold can reduce pessimism, the threshold must be set in consideration of the decrease in the *Recall*.

The comparison of the *Precision* results is shown in Figure 8(c). This result is the primary contribution of this paper. In the conventional method, the actual execution time of a task is almost never WCET, resulting in a low *Precision* in many cases where a deadline miss is detected but no deadline miss actually occurs. The purpose of this paper is to overcome this pessimism, and from the figure, this purpose has been fulfilled.

The *F-measure* results are shown in Figure 8(d). The *F-measure* is the harmonic mean of the *Recall* and the *Precision* and evaluates the balance between them. The results of *F-measure* show almost the same trend as those of *Precision*. This is because the *Recall* is always close to 1, meaning that the proposed method succeeds in reducing pessimism while minimizing the loss of the safety.

Table 5 *Earlier time* results.

Probability threshold	Average *earlier time* [ms]
1.00	301.8
0.99	229.0
0.95	198.0
0.90	183.4

Table 6 Experimental parameters in Section 5.5.

Parameter	Value
α in Equation (3)	2.0
Period of *timer-driven nodes* [ms]	10, 20, 30, 50, 60, 100

5.4 *Earlier Time* Result with Proposed Method

The *earlier time* of the proposed method is evaluated in this section. Here, the *earlier time* indicates how early a deadline miss is predicted. The systems, incorporating our proposed method, can detect a deadline miss earlier and shift to a safe state by *earlier time* than if deadline miss early detection is not performed. The task set and parameters evaluated in this section are identical to those in Section 5.3 and Table 4, respectively.

The *earlier time* results at each probability threshold are shown in Table 5. As mentioned above, the deadline miss prediction is more severe for higher threshold values and less severe for lower values. Generally, a higher threshold results in earlier deadline miss predictions, as shown in Table 5. However, the fact that a large value of the *earlier time* does not immediately mean that the method is superior should be noted. The reason is that the earlier the deadline miss prediction is made, regardless of the deadline miss probability at the time, the larger the result of the *earlier time* will be. However, in that case, most of the predicted deadline misses will not actually occur, resulting in an overly pessimistic system. Therefore, comparing the *earlier time* at different threshold values is not inherently meaningful. Rather, assessing the *earlier time* while considering the *Accuracy* of deadline miss detection at each threshold makes more sense. With a probability threshold of 0.95, the *Accuracy* for deadline miss detection is significantly improved over the conventional method, exceeding 0.8 for all utilization (refer to Figure 8(a)). At this threshold, the average *earlier time* is 198.0 ms. The practical impact of this *earlier time* is evident in operational scenarios. For instance, when an autonomous vehicle is traveling at 60 km/h, this vehicle advances approximately 3.3 meters in 198.0 ms, and this distance has a significant impact on the success of crisis avoidance. Therefore, the proposed method can detect a deadline miss early enough while maintaining high prediction accuracy when an appropriate probability threshold is specified.

5.5 Calculation Time of *plaxity* of Each Job

The time cost of computing *plaxity* with respect to the following two factors are evaluated: (i) the number of nodes in a system, and (ii) the unit of execution time. Note that a smaller execution time unit means that the number of entries in the execution time distribution of each task increases. The common parameter settings to the experiments for the two factors are shown in Table 6.

First, the variation of the calculation time with the number of nodes in a system is investigated. In this experiment, 500 DAGs with the number of nodes randomly set in the range $[20, 200]$ were generated to measure *plaxity* calculation time. Note that the execution

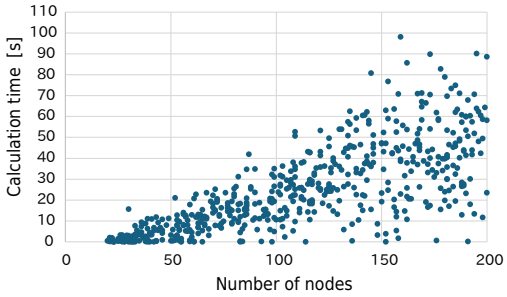


Figure 9 Calculation time of *plaxity* with increasing nodes.

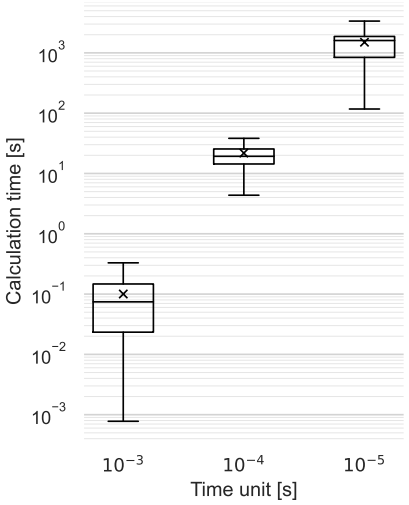


Figure 10 Calculation time of *plaxity* for different time units.

time unit for this experiment is 10^{-4} seconds. The results of this experiment are shown in Figure 9. The overall trend is that the calculation time lengthens as the number of nodes increases. However, the calculation time for a large number of nodes varies widely. This is because the number of nodes itself does not determine the number of *plaxity* calculated. The only jobs for which *plaxity* is calculated are those for which exit jobs have a direct or indirect dependency. The number of such jobs depends on the structure of the DAG and the difference in periods between subgraphs exchanging data. Nevertheless, the more nodes a system contains, the larger the maximum number of *plaxity* computed tends to be. As a result, the number of nodes and the calculation time of *plaxity* are correlated at a certain extent. Then, focusing on the case with 100 nodes, due to the fact that the number of nodes in *Autoware* is around 100, the calculation is completed in less than 40 seconds at the maximum. Considering that the calculation of *plaxity* is done statically during system development, this calculation time is a sufficiently practical level. Even when the number of nodes is doubled to 200, the calculation is completed in approximately 100 seconds, which is still a practical calculation cost.

Next, the relationship between the unit of execution time and the calculation time is examined. In this experiment, 100 DAGs each with execution time unit of 10^{-3} s, 10^{-4} s, and 10^{-5} s, the time cost of computing their *plaxity* is measured. Here, for example, the execution time distribution of a task with WCET of 5 ms has 5, 50, and 500 entries in each case. If the unit is 10^{-3} s, then all probabilities of execution times greater than 4 ms and less than 5 ms are integrated into the probability of 5 ms, which leads to pessimism. Therefore, while having as fine as possible is desirable, the increase in time cost resulting from this is not negligible and needs to be investigated. Note that the number of nodes in DAG is fixed at 100 for the aforementioned reason. The results of each experiment are shown in Figure 10. The calculation time in the case of 10^{-3} s is extremely short and not a problem at all. Besides, in the case of 10^{-4} s, as mentioned earlier, the calculation time is sufficiently practical level. Then, in the case of 10^{-5} s, the maximum calculation time is less than one hour. Although this is a longer time than the previous units, but still at a practical level, considering that the *plaxity* is calculated statically. As a whole, when the time unit is

reduced to 1/10, the calculation time tends to increase by a factor of about 100. This result is consistent with the insight obtained from the fact that the time complexity of applying linear convolution to distributions with m and n elements is $O(mn)$, and that the number of entries in the execution time distribution increases tenfold when the time unit is 1/10. Therefore, if the time unit is 10^{-6} s, the time required to calculate *plaxity* for a system with 100 nodes is expected to less than 100 hours, or about four days. From the standpoint of practicality, this will be the limit for mincing the execution time.

6 Related Work

Methods for determining job-level dependencies in HP have been proposed for timing analysis of DAGs consisting of tasks that are driven by different periods. One heuristic solution was proposed by Becker et al. [2]. In their method, the earliest and latest cases of data read and write times for each periodic task are considered, and all possible job-level dependencies are obtained. Then, as an equivalent problem to determining the job-level dependencies in HP, a method for converting a multi-period DAG into multiple single-period DAGs was proposed by Verucchi et al. [17]. These methods adequately determine job-level dependencies in multi-period DAGs. However, they consider DAGs consisting only of periodic tasks and are not directly applicable to DAGs with a mixture of *timer-driven* and *event-driven nodes*.

Models with a mixture of *timer-driven* and *event-driven nodes* have been studied in the field of the robot operating systems (ROS). The response time analysis of the ROS 2 processing chain, first proposed by Casini et al. [4] is an example of such a study. They also proposed a real-time scheduling method for ROS 2. However, their research has not determined the job-level dependencies.

As a result that straddles the above two contexts, a method for deadline miss early detection for mixed *timer-driven* and *event-driven* DAG tasks was proposed by Yano et al. [19]. Their method first determines the job-level dependencies in a mixed *timer-driven* and *event-driven* DAG task, and then places reasonable time constraints on each job based on defined end-to-end deadlines and *data freshness* constraints. At runtime, the system checks to see if the job start time meets the established time constraint to achieve deadline miss early detection. However, this method has the disadvantage of being pessimistic because each job is assumed to take WCET in calculating time constraints.

Pessimism caused by the assumption of WCET is a constant problem in the timing analysis of real-time systems. To overcome this problem, studies have been conducted to capture the runtime behavior of the system in probabilistic terms and to analyze the actual timing behavior in a more rigorous manner. One example is the method using convolution of probability mass functions, which was first proposed by Bernat et al. [3]. Their method expresses the execution time of program components as a probability mass function and defines a convolutional operation to add them. Then, by repeatedly applying this operation, the execution time of the entire program is finally expressed as a probability mass function. This framework was applied to the context of autonomous driving systems by Lee et al. [10]. They proposed a probabilistic analysis of the end-to-end latency of a multi-period DAG modeling an autonomous driving system. Similarly, a method to reduce end-to-end latency was proposed by Han et al. [6]. In their proposal, they introduced a technique to mitigate pessimistic predictions by using probabilistic execution time.

Table 7 Comparison of the proposed method with related work.

	DAG	MPD[a)]	MTE[b)]	DED[c)]	PRB[d)]
RTCSA 2016 [2]	✓	✓			
RTAS 2019 [17]	✓	✓			
ECRTS 2019 [4]	✓	✓	✓		
IEEE Access 2023 [19]	✓	✓	✓	✓	
RTSS 2002 [3]					✓
IEEE Transactions on Computers 2022 [10]	✓	✓			✓
RTAS 2023 [6]	✓	✓	✓		✓
Proposed method	✓	✓	✓	✓	✓

[a)] MPD: Multi-period DAG
[b)] MTE: Mixture of *timer-driven* and *event-driven nodes*
[c)] DED: Deadline miss early detection
[d)] PRB: Probabilistic method

7 Conclusion

In this paper, we have proposed a method for deadline miss early detection for real-time systems that takes into account variations in task execution time. The proposed method takes the execution time of each task in the system as a random variable and convolves them to place an appropriate time constraint called *plaxity* on each job in the DAG. This constraint allows the system to detect deadline misses early at any level required by users. In this way, the proposed method provides deadline miss early detection at an arbitrary probability threshold to suppress the pessimism of the conventional method. The experimental evaluation showed that the proposed method successfully reduced pessimism and significantly improved accuracy compared to the conventional method, which does not account for variations in task execution time. By setting a sufficiently high probability threshold with the proposed method, false positives can be significantly reduced with little loss of completeness in deadline miss early detection. Experiments also indicated that the calculation of the proposed method can be completed in a sufficiently practical time, assuming that the proposed method is applied to an actual automated driving system.

In future work, consideration of execution time variation in job-level dependency analysis will be of interest. The proposed method uses WCET for this analysis, but more accurate detection can be expected by probabilistically capturing job-level dependencies as well. In this case, since the computational complexity is expected to increase as the number of combinations increases, efforts to reduce the complexity are needed to be done. For example, circular convolution [12], appropriate re-sampling [5, 13], and parallelization [14] are seen as promising options.

References

1 Baidu Apollo project. URL: `https://www.apollo.auto/`.

2 Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. Synthesizing job-level dependencies for automotive multi-rate effect chains. In *Proceedings of the 22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 159–169, 2016.

3 Guillem Bernat, Antoine Colin, and Stefan M. Petters. WCET analysis of probabilistic hard real-time systems. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS)*, pages 279–288, 2002.

4 Daniel Casini, Tobias Blaß, Ingo Lütkebohle, and Björn B. Brandenburg. Response-Time Analysis of ROS 2 Processing Chains Under Reservation-Based Scheduling. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS)*, pages 1–23, 2019.

5 José Luis Díaz, José María López, Manuel García, Antonio Manuel Campos, Kanghee Kim, and Lucia Lo Bello. Pessimism in the stochastic analysis of real-time systems: concept and applications. In *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS)*, pages 197–207, 2004.

6 Taeho Han and Kanghee Kim. Minimizing probabilistic end-to-end latencies of autonomous driving systems. In *Proceedings of the 29th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 27–39, 2023.

7 Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on board: Enabling autonomous vehicles with embedded systems. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, pages 287–296, 2018.

8 Alix Munier Kordon and Ning Tang. Evaluation of the age latency of a real-time communicating system using the LET paradigm. In *Proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS)*, pages 1–21, 2020.

9 Takahisa Kuboichi, Atsushi Hasegawa, Bo Peng, Keita Miura, Kenji Funaoka, Shinpei Kato, and Takuya Azumi. CARET: Chain-Aware ROS 2 Evaluation Tool. In *Proceedings of the 20th IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, pages 1–8, 2022.

10 Hyoeun Lee, Youngjoon Choi, Taeho Han, and Kanghee Kim. Probabilistically guaranteeing end-to-end latencies in autonomous vehicle computing systems. *IEEE Transactions on Computers*, 71(12):3361–3374, 2022.

11 Jing Li, Jian Jia Chen, Kunal Agrawal, Chenyang Lu, Chris Gill, and Abusayeed Saifullah. Analysis of federated and global scheduling for parallel real-time tasks. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 85–96, 2014.

12 Filip Marković, Alessandro Vittorio Papadopoulos, and Thomas Nolte. On the Convolution Efficiency for Probabilistic Analysis of Real-Time Systems. In *Proceedings of the 33rd Euromicro Conference on Real-Time Systems (ECRTS)*, pages 16:1–16:22, 2021.

13 Dorin Maxim, Mike Houston, Luca Santinelli, Guillem Bernat, Robert I. Davis, and Liliana Cucu-Grosjean. Re-sampling for statistical timing analysis of real-time systems. In *Proceedings of the 20th ACM International Conference on Real-Time and Network Systems (RTNS)*, pages 111–120, 2012.

14 Suzana Milutinovic, Jaume Abella, Damien Hardy, Eduardo Quiñones, Isabelle Puaut, and Francisco J. Cazorla. Speeding up static probabilistic timing analysis. In *Proceedings of the 28th International Conference on Architecture of Computing Systems (ARCS)*, pages 236–247, 2015.

15 SAE International. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, 4th edition, April 2021.

16 John A Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio Buttazzo. *Deadline scheduling for real-time systems: EDF and related algorithms*, volume 460. Springer Science & Business Media, 1998.

17 Micaela Verucchi, Mirco Theile, Marco Caccamo, and Marko Bertogna. Latency-aware generation of single-rate DAGs from multi-rate task sets. In *Proceedings of the 26th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 226–238, 2020.

18 Waymo driver. URL: `https://waymo.com/waymo-driver/`.

19 Atsushi Yano and Takuya Azumi. Deadline miss early detection method for mixed timer-driven and event-driven DAG tasks. *IEEE Access*, 11:22187–22200, 2023.

20 Atsushi Yano and Takuya Azumi. RD-Gen: Random DAG generator considering multi-rate applications for reproducible scheduling evaluation. In *Proceedings of the 26th IEEE International Symposium on Real-Time Distributed Computing (ISORC)*, pages 21–31, 2023.