Migrating from Per-Job Analysis to Per-Resource Analysis for Tighter Bounds of End-to-End Response Times

Man-Ki Yoon, Student Member, IEEE, Chang-Gun Lee, Member, IEEE, and Junhee Han, Member, IEEE

Abstract—As the software complexity drastically increases for multiresource real-time systems, industries have great needs for analytically validating real-time behaviors of their complex software systems. Possible candidates for such analytic validations are the end-to-end response time analysis techniques that can analytically find the worst-case response times of real-time transactions over multiple resources. The existing techniques, however, exhibit severe overestimation when real-time transactions visit the same resource multiple times, which we call a multiple visit problem. To address the problem, this paper proposes a novel analysis that completely changes its analysis viewpoint from classical per-job basis—aggregation of per-job response times—to per-resource basis—aggregation of per-resource total delays. Our experiments show that the proposed analysis can find significantly tighter bounds of end-to-end response times compared with the existing per-job-based analysis.

Index Terms—Per-resource analysis, end-to-end response time analysis, controller area network, real-time and embedded systems.

1 INTRODUCTION

Recently, many cyber-physical systems such as automobiles and aircrafts are increasingly employing electronic parts because of their functional diversity and low cost. Accordingly, the complexity of software that works with those electronic parts is also increasing. Thus, industries have great needs for analytically validating real-time behaviors of such complex software systems.

For that purpose, we may use the holistic analysis technique proposed by Tindell and Clark [1] and its extensions [2], [3], [4], [5], [6], [7] that can analytically compute the end-to-end response times of transactions that consist of a sequence of tasks running over chains of multiple resources such as electronic sensors/actuators, networks, and microprocessors. However, we identify that the existing techniques give very loose bounds on the end-to-end response time of each job assuming the worst-case scenario of high-priority arrivals and then add up all the per-job response times to compute the end-to-end response time as shown in Fig. 1b. Such per-job-based analysis, however, severely double counts the high-priority arrivals when a low-priority transaction visits the same resource multiple times, which we call a multiple visit problem.

As a simple example to motivate the multiple visit problem, let us consider Fig. 1a, which shows three

Electronic Control Units (ECUs) connected through a Controller Area Network (CAN) bus. On top of these resources, a high-priority periodic transaction consists of five tasks (1, 2, 3, 4, 5) that visit ECU2, CAN, ECU1, CAN, and ECU3, respectively. In addition, a low-priority periodic transaction consists of five tasks (6, 7, 8, 9, 10) that visit ECU1, CAN, ECU3, CAN, and ECU2, respectively. For this system, existing techniques compute the response time of each job assuming the worst-case scenario of high-priority arrivals and then add up all the per-job response times to compute the end-to-end response time as shown in Fig. 1b. Such per-job-based analysis, however, severely double counts the high-priority arrivals when a low-priority transaction visits the same resource multiple times. In the example, the low-priority transaction visits CAN twice, i.e., jobs 7 and 10. For each of these two visits, the per-job-based analysis assumes the worst-case delay by the high-priority jobs on CAN, i.e., 2 and 4. Thus, execution times of 2 and 4 are doubly counted in the end-to-end response time of the low-priority transaction as shown in Fig. 1b. However, considering the period of the high-priority transaction, we can note that at most, a single instance of the high-priority transaction can delay the given instance of the low-priority transaction at CAN.

This overestimation becomes more serious as the multiple visit count becomes larger due to the complex long transactions. Due to this reason, the traditional per-job-based end-to-end response time analysis may conclude that the system is not schedulable even when the resources are severely underutilized. Our preliminary experimental results in Fig. 2 show that when the multiple visit count on a resource gets larger, the traditional per-job analysis can utilize only up to 30 percent while the simulation states that the system is still schedulable up to 80 percent utilization. (The detailed experimental settings will be given later.)
To address this problem, this paper proposes a per-resource-based end-to-end response time analysis. The new analysis completely changes the analysis viewpoint from “per-job basis” to “per-resource basis.” That is, it computes the total delay at each resource. By adding the total delays at all the resources, we can find a bound of the end-to-end response time. Fig. 3 conceptually compares the per-job-based analysis (see the horizontal addition) and the per-resource-based analysis (see the vertical addition). The per-resource-based analysis does not suffer from “double counting for multiple visits,” and hence, gives a much tighter bound on the end-to-end response time, especially when a transaction is complex and long, and thus, visits the same resource many times.

The rest of the paper is organized as follows: Section 2 summarizes the related work. Section 3 formally defines the problem and motivates our new analysis. In Section 4, we propose our per-resource-based holistic analysis that can find a significantly tighter bound on the end-to-end response time. Section 5 presents the experimental results. Finally, Section 6 concludes the paper.

2 RELATED WORK

The famous classical work [8], [9] presents the worst-case response time analysis for multiple tasks on a single processor fixed-priority scheduling system. This analysis is first extended by Tindell and Clark for distributed systems with multiple resources [1]. It basically computes the end-to-end response time of a transaction consisting of a sequence of jobs (and messages) over multiple resources by aggregating the per-job response times. Its correctness is revisited by Palencia et al. [2].

This end-to-end response time analysis has been improved in many ways. For example, its pessimism due to release jitters is addressed by reducing or eliminating the jitters with a sporadic server, release guards, or best-case response time considerations [4], [10], [11]. The work in [7] explicitly considers precedence relations among jobs in the same transactions and also their priorities to improve the accuracy of the analysis. Another group of work [5], [12], [13], [14] explicitly uses time correlations among jobs, called offsets, in order to less conservatively estimate the preemptions by high-priority jobs. This consideration is further improved in [6] by more accurately estimating the offsets.
relative to arrival/completion times of predecessor tasks instead of single reference point of the external event time.

However, all these works have their basis on Tindell and Clark’s per-job analysis [1], and hence, do not provide a fundamental solution for the aforementioned multiple visit problem.

The delay composition theorem [15] is an innovative idea for analyzing the end-to-end delay of a pipelined distributed system. Unlike the previous work that assumes the worst-case preemption pattern at each stage of a pipelined transaction, the delay composition theorem considers the overlapped executions in different pipeline stages, and hence, reduces the pessimism of the end-to-end delay analysis. More importantly, the theorem provides a way of transforming a pipelined distributed system into a uniprocessor system. Due to this transformation, rigorous schedulability analysis techniques developed for uniprocessor systems can be applied to pipelined distributed systems. One limitation, however, is that all the transactions in the system should follow the same path along the same resource stages. This limitation is addressed in [16] by combining transactions following different paths into a Directed Acyclic Graph (DAG) and extending the delay composition theorem to a DAG. Regardless of this extension, it still assumes that each transaction follows an acyclic path without revisiting the same resources. Therefore, the delay composition theorem is not applicable to our target system, where transactions visit resources multiple times in arbitrary manners.

There also have been efforts to apply the analysis techniques to automotive applications [17], [18], [19]. The analysis techniques used, however, are again per-job-based. Therefore, they are not free from the multiple visit problem either.

In contrast, our analysis proposed in this paper changes the analysis viewpoint to the per-resource basis in order to fundamentally address the pessimism due to the multiple visit problem.

3 Problem Description

In this paper, we consider a system that consists of M resources denoted by \{R_1, R_2, ..., R_M\}. Some resources are processors and others are communication links to deliver messages among tasks on processors. However, for the simplicity of explanation, we do not differentiate the two resource types assuming that all the resources schedule their transactions following different paths into a Directed Acyclic Graph (DAG) and extending the delay composition theorem to a DAG. Regardless of this extension, it still assumes that each transaction follows an acyclic path without revisiting the same resources. Therefore, the delay composition theorem is not applicable to our target system, where transactions visit resources multiple times in arbitrary manners.

One limitation, however, is that all the transactions in the system should follow the same path along the same resource stages. This limitation is addressed in [16] by combining transactions following different paths into a Directed Acyclic Graph (DAG) and extending the delay composition theorem to a DAG. Regardless of this extension, it still assumes that each transaction follows an acyclic path without revisiting the same resources. Therefore, the delay composition theorem is not applicable to our target system, where transactions visit resources multiple times in arbitrary manners.

There also have been efforts to apply the analysis techniques to automotive applications [17], [18], [19]. The analysis techniques used, however, are again per-job-based. Therefore, they are not free from the multiple visit problem either.

In contrast, our analysis proposed in this paper changes the analysis viewpoint to the per-resource basis in order to fundamentally address the pessimism due to the multiple visit problem.

\[ \Gamma_i = (p_1, \{ \tau_{i,1} = (r_{i,1}, e_{i,1}), \tau_{i,2} = (r_{i,2}, e_{i,2}), \ldots, \tau_{i,|\Gamma_i|} = (r_{i,|\Gamma_i|}, e_{i,|\Gamma_i|}) \}). \]

One instance of the whole sequence of \( \tau_{i,1}, \tau_{i,2}, \ldots, \tau_{i,|\Gamma_i|} \) is called a \( \Gamma_i \) transaction instance. Fig. 4 shows two periodic instances of an example transaction \( \Gamma_1 \) to visualize the meanings of its notations. Note that each instance of this transaction visits ECU\(_1\) twice, ECU\(_2\) once, and CAN twice. For every instance, the whole sequence of \( \Gamma_i \)’s tasks must be completed within a period, that is, the end-to-end deadline is equal to the period \( p_i \). Although we assume that the end-to-end deadline is equal to the period for the simplicity, the proposed analysis still works even when the end-to-end deadline is shorter than the period.

**Problem description.** For such a given set of \( N \) periodic transactions \( \{ \Gamma_1, \Gamma_2, \ldots, \Gamma_N \} \) over \( M \) resources \( \{R_1, R_2, \ldots, R_M\} \), our problem is to check whether every transaction \( \Gamma_i \) can always meet the end-to-end deadline \( p_i \).

For this problem, we may use the traditional end-to-end response time analysis [1]. Its short overview is given in the following. For each task of a transaction \( \Gamma_i \), its per-job worst-case response time \( \omega_{i,k} \) is calculated with the following response time equation 1:

\[ \omega_{i,k} = e_{i,k} + \sum_{\forall j < i} \sum_{\forall \tau_{j,a} = r_{i,k}} \left[ J_{j,a} + \omega_{i,k} \right] \frac{e_{j,a}}{p_j}, \]

where \( J_{j,a} \) is the worst-case release jitter of the task \( \tau_{j,a} \) of a higher priority transaction \( \Gamma_j \). \( J_{j,a} \) is simply given as the worst-case response time until the completion of the \((a-1)\)th task \( \tau_{j,a-1} \) of \( \Gamma_j \). The equation means that the per-job worst-case response time of \( \omega_{i,k} \) can be calculated by adding 1) its own execution time \( e_{i,k} \) and 2) the largest possible delay due to the higher priority jobs on the same resource, which is given as \( \sum_{\forall j < i} \sum_{\forall \tau_{j,a} = r_{i,k}} \left[ J_{j,a} + \omega_{i,k} \right] \frac{e_{j,a}}{p_j} \). In the second term, \( j_{a,k+1}^w \) is the largest number of releases of \( \tau_{j,a} \) of \( \Gamma_i \) during the time window \( w_{i,k} \) assuming the worst-case release pattern, where its first release of \( \tau_{j,a} \) is delayed the most, i.e., \( J_{j,a} \) and succeeding releases are with the maximum rate, i.e., \( 1/p_j \). Applying the above equation for all the tasks \( \tau_{i,k} \) of \( \Gamma_i \), the worst-case end-to-end response time denoted by \( e2eRspTime_i \) can be calculated by adding up all the per-job response times, i.e.,

\[ e2eRspTime_i = \sum_{k=1}^{\Gamma_i} \omega_{i,k}. \]

This per-job-based analysis can severely overestimate the end-to-end response time when a transaction \( \Gamma_i \) visits the

1. When the deadline is less than or equal to the period, we do not have to consider the delay by the previous instances of the same transaction. Omitting such factor, (2) is a simplified one from the original in [1].
same resource multiple times. In Fig. 4, if we individually apply (2) for \( \tau_{i,2} \) and \( \tau_{i,4} \) of \( \Gamma_i \) that visit the CAN resource, the worst-case delay by higher priority jobs represented by the second term of (2) is counted twice in the final end-to-end response time calculation of (3). The same problem happens for \( \tau_{i,1} \) and \( \tau_{i,2} \) that visit \( ECU_i \).

Our new analysis aims at addressing this pessimism by changing the analysis viewpoint from per-job to per-resource.

### 4 Per-Resource-Based Analysis for End-to-End Response Time

For our per-resource-based analysis, we introduce a notion of “per-resource total delay.” The worst-case total delay that one \( \Gamma_i \) instance experiences due to a higher priority transaction \( \Gamma_j \) at resource \( R_l \) is denoted by \( TD_l^i(R_l) \). Using this notion, the time that one \( \Gamma_i \) instance spends at \( R_l \) can be calculated by adding its execution times at \( R_l \) and its total delays by all the higher priority transactions at \( R_l \) that is,

\[
\left( \sum_{j \in \{i, k\} | r_j = R_l} e_{i,k} \right) + \sum_{j=1}^{i-1} TD_l^j(R_l).
\]

Therefore, the end-to-end response time of \( \Gamma_i \) can be calculated by summing up the times spent at all the visiting resources as follows:

\[
e_{2eRspTime_i} = \sum_{R_l \in \{R_1, \ldots, R_m\}} \left( \left( \sum_{j \in \{i, k\} | r_j = R_l} e_{i,k} \right) + \sum_{j=1}^{i-1} TD_l^j(R_l) \right).
\]

Note that for any resource \( R_l \) that \( \Gamma_i \) does not visit, both parts of \( e \) values and \( TD \) values in (4) are zero.

Now, the remaining problem is to find an upper bound on the per-resource total delay \( TD_l^i(R_l) \). In the following, we explain how to find it focusing on a transaction \( \Gamma_i \) assuming that analysis for all the higher priority transactions \( \Gamma_j \) \((j = 1, 2, \ldots, i-1)\) has been completed.

#### 4.1 Total Delay Bound for a Known Total Window

In order to find an upper bound on the per-resource total delay \( TD_l^i(R_l) \), let us introduce another notion of a “per-resource total window.” The per-resource total window denoted by \( TW_l^i(R_l) \) represents the time window during which an instance of a transaction \( \Gamma_i \) has uncompleted jobs to be executed on a resource \( R_l \). In order to explain the concept of the per-resource total window, consider an example in Fig. 5, where two transactions \( \Gamma_1 \) and \( \Gamma_2 \) are concurrently running on three resources, \( R_1, R_2, \) and \( R_3 \). In this example, the total window of \( \Gamma_2 \) at resource \( R_2 \) is the time span from the release time of its first visit on \( R_2 \) i.e., \( \tau_{2,1} \), to the completion time of its last visit on \( R_2 \) i.e., \( \tau_{2,6} \). The total window is depicted as dotted boxes in Fig. 5.

Supposing that we can somehow find an upper bound of the total window \( TW_l^i(R_l) \), the number of instances of a higher priority task \( \tau_{j,a} \) that can be released during \( TW_l^i(R_l) \) and hence delay one \( \Gamma_i \) instance at resource \( R_l \) (i.e., \( \tau_{j,a} = R_l \)) can be upper bounded by

\[
Z_{j,a}(TW_l^i(R_l)) = \left[ \frac{J_{j,a} + TW_l^i(R_l)}{p_j} \right] ,
\]

assuming the worst-case release pattern of \( \tau_{j,a} \), i.e., the first release is delayed the most by the amount of jitter \( J_{j,a} \) and subsequent releases are most packed with the maximum release rate of \( 1/p_j \). Later, we will explain how to calculate \( J_{j,a} \) in Section 4.3. In Fig. 5, considering \( p_1 \), the maximum number of instances of \( \tau_{i,4} \) during \( TW_2^i(R_2) \) is one. In contrast, the per-job analysis counts one instance for each job of the three visits on \( R_2 \), i.e., \( \tau_{2,2}, \tau_{2,4}, \) and \( \tau_{2,6} \), and thus, totally three instances are pessimistically counted in the final end-to-end response time of \( \Gamma_2 \).

Now, for the known duration of total window, i.e., \( TW_l^i(R_l) \), we are clear in that no more than \( Z_{j,a}(TW_l^i(R_l)) \) instances of \( \tau_{j,a} \) can delay one \( \Gamma_i \) instance at resource \( R_l \). However, if we count all of \( Z_{j,a}(TW_l^i(R_l)) \) instances as contributions to the total delay \( TD_l^i(R_l) \), it would be too pessimistic, especially when the period of a high-priority transaction \( \Gamma_j \) is much shorter than the total window \( TW_l^i(R_l) \). As an example, Fig. 6 shows two cases: 1) Case I, where the high-priority period \( p_j \) is longer than the total window \( TW_l^i(R_l) \) and 2) Case II, where \( p_j \) is shorter than \( TW_l^i(R_l) \). In Case I, it is okay to count \( Z_{j,a}(TW_l^i(R_l)) = 1 \) in the total delay estimation. In Case II, however, it is pessimistic to count all the four instances of \( \tau_{j,a} \) as contributions to the total delay, since \( \Gamma_i \) may not be “busy”—having jobs released but not completed, in the whole duration of \( TW_l^i(R_l) \).

In both Cases I and II, in order to more tightly but still sufficiently count the number of instances of \( \tau_{j,a} \) instances that should be included in the total delay, we compute the worst-case per-job busy interval one by one by using \( Z_{j,a}(TW_l^i(R_l)) \) as the limit of delay as follows:

\[
\sum_{j<i} \sum_{R_l \in \{R_1, \ldots, R_m\}} \left( \min \left( \frac{J_{j,a} + w_{j,a}}{p_j}, Z_{j,a}(TW_l^i(R_l)) \right) e_{j,a} \right).
\]

In Case I of Fig. 6, if we apply the above equation to the first job with \( Z_{j,a}(TW_l^i(R_l)) = 1 \), the one \( \tau_{j,a} \) instance is included in the first busy interval and there is no more \( \tau_{j,a} \) instance in the duration of the total window \( TW_l^i(R_l) \). Thus, when we apply the above equation to the second and third jobs of \( \Gamma_i \), \( Z_{j,a}(TW_l^i(R_l)) = 0 \) is used. Thus, no more instances of \( \tau_{j,a} \) are counted for the second and third job delays. Overall, only one \( \tau_{j,a} \) instance is counted for all the three jobs of \( \Gamma_i \). Although the Case I of Fig. 6 shows a
specific scenario where the first visit of $\Gamma_i$ is delayed by the single instance of $\tau_{j,a}$, it does not matter which visit is actually delayed. The thing that matters is that the total delay by $\tau_{j,a}$ that a single $\Gamma_i$ instance experiences at resource $R_l$ is upper bounded by one $e_{j,a}$. This is the case where we can find a tighter estimation of the total delay than the per-job analysis [1] that counts one $e_{j,a}$ for each of the three jobs.

In Case II of Fig. 6, when we apply the above equation to the first job, the given value of 4 is used as $Z_{j,a}(TW_i(R_l))$. When calculating the first job’s busy interval, we assume the worst-case release pattern of the four $\tau_{j,a}$-instances as shown in the middle of the Case II (Fig. 6). Hence, we include $J_{j,a}$ in the ceiling function of (6). This way, we can maximally include $e_{j,a}$ in the first job’s busy interval, regardless of various release scenarios. Once the above recursive equation converges, we can notice how many $e_{j,a}$s are included in the first job’s busy interval. Suppose that the number is one as shown in Case II of the figure. Then, the leftover two instances of $\tau_{j,a}$ have no way to be included in the total delay $TD_j(R_l)$ because all the $R_l$ visiting jobs of $\Gamma_i$ have already included maximal delay effects by $\tau_{j,a}$ assuming the worst case, just like the per-job analysis [1]. Therefore, we can sufficiently count only two as contributions to the total delay $TD_j(R_l)$ as shown in Case II of the figure. This is the case where we find the same estimation of the total delay as the per-job analysis [1].

This way, in both Cases I and II, the delay contributions counted in our analysis are always smaller than or equal to that of the per-job analysis by Tindell and Clark [1].

Applying (6) for all the tasks $\tau_{j,a}$ of $\Gamma_i$ that visit $R_l$, we can obtain the worst-case numbers of instances of $e_{j,a}$ that can contribute to the total delay $TD_j(R_l)$ within the given duration of the total window $TW_i(R_l)$. Denoting such a number of instances of $e_{j,a}$ by $C_{j,a}^{\tau_{j,a}}(TW_i(R_l))$, the total delay $TD_j(R_l)$ of $\Gamma_i$ caused by $\tau_{j,a}$ at resource $R_l$ for the given total window $TW_i(R_l)$ can be computed as follows:

$$TD_j(R_l) = \sum_{\forall\{\tau_{j,a} = R_l\}} C_{j,a}^{\tau_{j,a}}(TW_i(R_l)) \times e_{j,a}. \quad (7)$$

Lemma 1. $TD_j(R_l)$ computed by (7) is an upper bound of the total delay that an instance of $\Gamma_i$ experiences by tasks of $\Gamma_j$ at resource $R_l$ during the given total window $TW_i(R_l)$.

Proof. If we omit $Z_{j,a}(TW_i(R_l))$ from (6), the resulting count $C_{j,a}^{\tau_{j,a}}(TW_i(R_l))$ used in (7) is exactly the same as the Tindell
The total delay are interdependent. To find an upper bound of the traditional recursive response time equation [8], [9], we use an iterative convergence approach, which is similar to the iterative solving of (6). Thus, Tindell’s count is an upper bound of the total delay that one instance at \( R_i \) can delay all the tasks \( \tau_{i,k} \) of \( \Gamma_i \) during its busy period \( TW_i(R_i) \), as proved in [1]. Only if Tindell’s count is larger than \( Z_{j,a}(TW_i(R_i)) \) (as in Case I of Fig. 6), (6) gives an upper bound of the number of \( R_i \)'s tasks visiting \( R_i \) during the whole duration of \( TW_i(R_i) \) as the worst-case busy period of \( \Gamma_i \) at \( R_i \). Therefore, \( Z_{j,a}(TW_i(R_i)) \) is also an upper bound of the number of \( \tau_{j,a} \) instances that can delay all the tasks \( \tau_{i,k} \) of one \( \Gamma_i \) instance at \( R_0 \). Supposing that \( TW_i(R_i) \) is an upper bound of the total window of \( \Gamma_i \) at \( R_i \), we can pessimistically consider the whole duration of \( TW_i(R_i) \) as the worst-case busy period of \( \Gamma_i \) at \( R_i \). The former can simply be represented by the first term of (8), which adds all the execution times of the subsequence \( \tau_{i,2} \), \( \tau_{i,4} \), \( \tau_{i,5} \), \( \tau_{i,6} \), i.e.,

\[
\sum_{j=1}^{i-1} TD_j(R_i) = \left( \tau_{i,2}, \tau_{i,4}, \tau_{i,5}, \tau_{i,6} \right)
\]

As a side note, if we additionally consider intertask offsets of a higher priority transaction as in [5], [6], [12], counting the contributions to the total delay can be more accurate. This improvement will be made in our future work.

### 4.2 Iterative Calculation of Total Delay and Total Window

The total delay \( TD_i(R_i) \) and the total window \( TW_i(R_i) \), in fact, are interdependent. To find an upper bound of \( TD_i(R_i) \) addressing the interdependency, we use an iterative convergence approach, which is similar to the iterative solving of traditional recursive response time equation [8], [9].

Initially, we set \( TD_i(R_i) = 0 \) for all the high-priority transactions \( \Gamma_i \in \{ \Gamma_1, \ldots, \Gamma_{i-1} \} \) and for all the resources \( R \in \{ R_1, \ldots, R_M \} \).

Once the total delay values \( TD_i(R_i) \) for all \( j \) and \( R \) are given (initial values of them are zero), we can compute an upper bound of the total window \( TW_i(R_i) \) for all \( R \). To explain this, let us denote the \( \Gamma_i \)'s tasks visiting \( R \) by \( (\tau_{i,v_1}, \tau_{i,v_2}, \ldots, \tau_{i,v_m}) \),

\[
\tau_{i,v_1} = \tau_{i,v_2} = \cdots = \tau_{i,v_m} = R \quad \text{and} \quad v_1 < v_2 < \cdots < v_m .
\]

Using this notation, an upper bound of \( \Gamma_i \)'s total window at \( R_i \), i.e., \( TW_i(R_i) \), can be computed as follows:

\[
TW_i(R_i) = \sum_{v_1 \leq k \leq v_m} e_{i,k} + \sum_{v_1 \leq k \leq v_m} \sum_{R \in \{ R_1, \ldots, R_M \}} TD_i(R_i) X_{v_1}^v(R_i) .
\]

where \( X_{v_1}^v(R_i) \)

\[
X_{v_1}^v(R_i) = \begin{cases} 
1 & \text{if there is a visit on } R_i \text{ in the subsequence from } \tau_{i,v_1} \text{ to } \tau_{i,v_m}; \\
0 & \text{otherwise.}
\end{cases}
\]

This equation can be best explained in Fig. 7. Suppose that total delays so far at all resources \( R_1, R_2, R_3, \) and \( R_4 \) are given as the shaded boxes in the figure. Then, the total window \( TW_i(R_3) \) at \( R_3 \) is given by adding 1) the execution times of the subsequence \( (\tau_{j,2}, \tau_{j,3}, \tau_{j,4}, \tau_{j,5}, \tau_{j,6}) \) from the first visit on \( R_2 \), i.e., \( \tau_{j,1} \), to the last visit on \( R_2 \), i.e., \( \tau_{j,8} \) and 2) the delays that \( \Gamma_j \) experiences while executing the subsequence. The former can simply be represented by the first term of (8), which adds all the execution times of the subsequence, i.e.,

\[
\sum_{2 \leq k \leq 6} e_{i,k} = e_{i,2} + e_{i,3} + e_{i,4} + e_{i,5} + e_{i,6} .
\]

The latter can be calculated by conservatively assuming that all the portions of the total delays happen within the subsequence as depicted in the figure. This is represented by the second term of (8). Note that there is no visit on \( R_4 \) within the subsequence. Thus, \( X_{v_1}^v(R_i) = 0 \) by definition of \( X \). This prevents the total delay at \( R_4 \) from being added in the second term of (8).

**Lemma 2.** Supposing that \( TD_i(R_i) \) is an upper bound of the total delay of \( \Gamma_i \) by \( \Gamma_j \) at \( R_i \) for every \( j \) and \( R_0 \), \( TW_i(R_i) \) computed by (8) is an upper bound of the total window of \( \Gamma_i \) at \( R_i \).
Proof. The total window of $\Gamma_i$ at $R$ starts from the release time of $\tau_{i,v_i}$ and ends at the completion time of $\tau_{i,v_n}$. In between these two time points, $\Gamma_i$ may visit other resources and revisit $R$ as shown in Fig. 7. Thus, the total window is given as the sum of 1) execution times of all the tasks of $\Gamma_i$ from $\tau_{i,v_i}$ to $\tau_{i,v_n}$, and 2) delays they experience at their visiting resources by higher priority tasks—see Fig. 7. The former can be exactly counted by the first term of (8). Thus, the remaining problem is to show that the second term of (8) is an upper bound of the delay part. Since $TD_j^i(R_i)$ is an upper bound of the total delay by $\Gamma_i$ that the whole task sequence of one $\Gamma_i$ instance experiences at $R_i$, it also upper bounds the delay that tasks within the subsequence experience at $R_i$ by $\Gamma_i$. By summing up $TD_j^i(R_i)$ for all visiting resources $R_i$ and for all higher priority transactions $\Gamma_j$, the second term of (8) upper bounds the delay part. Therefore, $TW_i(R)$ computed by (8) is an upper bound of the total window of $\Gamma_i$ at $R$.

Once we have the total windows for all the resources, i.e., $TW_i(R_1), TW_i(R_2), \ldots, TW_i(R_M)$, we can compute new upper bounds of total delays, i.e., $TD_j^i(R_1), TD_j^i(R_2), \ldots, TD_j^i(R_M)$ for all $j \in \{1, \ldots, i-1\}$ with (6) and (7). If any of these new total delays is larger than its previous value, we continue the iterations. When all the total delay values no longer increase, that is, “convergence,” we terminate the iteration. With the converged values of the total delays $TD_j^i(R_i)$, we can finally compute an upper bound of the end-to-end response time of $\Gamma_i$, i.e., $e2eRspTime_e$, based on (4).

**Theorem 1.** $e2eRspTime_e$ computed by (4) is an upper bound of the end-to-end response time of one $\Gamma_i$ instance.

Proof. $TW_i(R_j)$ for all $R_i$ and $TD_j^i(R_i)$ for all $j$ and $R_i$ are monotonically nondecreasing. Due to this fact, and by Lemmas 1 and 2, the converged value of $TD_j^i(R_i)$ for every $j$ and $R_i$ is an upper bound of the total delay that one $\Gamma_i$ instance experiences by tasks of $\Gamma_j$ at resource $R_i$. Therefore,

$$
\left( \sum_{\forall (i,k) | (i,k) = R_i} e_{i,k} \right) + \sum_{j=1}^{i-1} TD_j^i(R_i)
$$

is an upper bound of the total time that one $\Gamma_i$ instance spends at resource $R_i$. Consequently,

$$
\sum_{R_i \in \{R_1, \ldots, R_M\}} \left( \sum_{\forall (i,k) | (i,k) = R_i} e_{i,k} \right) + \sum_{j=1}^{i-1} TD_j^i(R_i)
$$

in (4) gives an upper bound of the total time that one $\Gamma_i$ instance spends at all its visiting resources. (Note that for a nonvisiting resource $R_i$, both parts of $e$ values and $TD$ values in (4) are zero.) Therefore, the theorem holds.$\Box$

### 4.3 Jitter Calculation for Analysis of Low-Priority Transactions

For analyzing a low-priority transaction, we have to compute the worst-case release jitters for all the tasks $\{\tau_{i,1}, \tau_{i,2}, \ldots, \tau_{i,j}\}$ of $\Gamma_i$. Since a task $\tau_{i,k}$ is released by the completion of its immediate predecessor task $\tau_{i,k-1}$, its worst-case release jitter $J_{i,k}$ can be given as the worst-case response time until the completion of $\tau_{i,k-1}$. It can simply be calculated by applying the above per-resource-based end-to-end response time analysis to the subsequence $(\tau_{i,1}, \ldots, \tau_{i,k-1})$.

### 5 Experiments

This section validates our proposed analysis in terms of the analysis accuracy. For this, we consider an automotive-style resource model with 10 resources, i.e., $\{R_1, R_2, \ldots, R_{10}\}$, one of which is a CAN bus and other nine resources are ECUs communicating each other through the CAN bus. On top of this resource model, we assume five periodic transactions $\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5$. The set of five periodic transactions is randomly generated as follows: The period of each transaction is randomly selected from the range of $[p_{\text{min}}, p_{\text{max}}]$ following the uniform distribution. The priority of a transaction is assigned according to the period, that is, a transaction with a shorter period is assigned with a higher priority. Every transaction has the same length $L$—the number of tasks in $\Gamma_i$ is $L$ for all $\Gamma_i \in \{\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5\}$. The first task of a transaction is mapped to an ECU randomly selected out of the nine ECUs. The second task is mapped to the CAN modeling the message transmission to the third task. Then, the third task is again mapped to a randomly selected ECU and so on, until we make a sequence of $L$ tasks. By increasing the transaction length $L$, we can control the visit count on CAN. The execution time of a task mapped on a resource is randomly selected from the range of $[1 \text{ ms}, 5 \text{ ms}]$. Table 1 summarizes these experimental parameters. The results below are the averages for such 300 generated random sets.

*With these parameters, we compare four analysis methods as follows:

- Tindell and Clark’s per-job analysis [1] denoted by Tindell,
- Palencia and Harbour’s [5] and Turja and Nolin’s [12] enhanced per-job analysis considering intertask offsets denoted by WCDO meaning “worst-case Dynamic Offset,”
- Henia and Ernst’s further improved analysis [6] denoted by Henia, and
- our proposed per-resource analysis denoted by OurPerResource.

We also present simulations results. Although the simulation does not give upper bounds of end-to-end response times, it can give typical end-to-end response times. We use

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of resources $M$</td>
<td>10 (1 CAN and 9 ECUs)</td>
</tr>
<tr>
<td>number of transactions $N$</td>
<td>5</td>
</tr>
<tr>
<td>transaction period $p_i$</td>
<td>uniform from $[p_{\text{min}}, p_{\text{max}}]$</td>
</tr>
<tr>
<td>transaction length $</td>
<td>\Gamma_i</td>
</tr>
<tr>
<td>task sequence $(\tau_{i,1}, \ldots, \tau_{i,L})$</td>
<td>random ECU and CAN alternating</td>
</tr>
<tr>
<td>task execution time $e_{i,k}$</td>
<td>uniform from $[1 \text{ ms}, 5 \text{ ms}]$</td>
</tr>
</tbody>
</table>
the simulation results to see the overestimation by the analysis methods.

Fig. 8 compares the end-to-end response time $e2eRspTime$ of the lowest priority transaction $T_{SC}$ as increasing the transaction length $L$, and hence, the visit count on CAN. The x-axis shows $L$ together with the CAN visit count in the parenthesis. In this experiment, the period selection range $[p_{min}, p_{max}]$ is fixed as $[100 \text{ ms}, 1,000 \text{ ms}]$. When the transaction length is short, all of the above four analysis methods give a pretty tight bound on the end-to-end response time, which is close to the one by simulation. As increasing the transaction length, the end-to-end response time increases since the average workload on all the resources increases. However, the increase rates are quite different. All of the per-job-based analysis methods, i.e., Tindell, WCDO, and Henia, show a sharp increase of the end-to-end response time due to many double counts for increasing multiple visits. On the other hand, our per-resource analysis suffers less from the double counting problem, and thus, shows a much less increase of the end-to-end response time. Consequently, when the transaction length is 19, the end-to-end response time by our analysis is over four times shorter than those by per-job analysis methods. Moreover, our result is quite close to the simulation result.

In order to see how much we can utilize the resources under the schedulability constraint, we perform another experiment as scaling-up all the execution times until the system becomes unschedulable. At the saturation point, we observe the utilization of the resource with the largest utilization, which we call a maximum schedulable utilization. Fig. 9 compares the maximum schedulable utilization as increasing the transaction length $L$ with the period selection range of $[100 \text{ ms}, 1,000 \text{ ms}]$. As expected by Fig. 8, when the transaction length is long, say 19, the maximum schedulable utilizations by the per-job analysis methods are below 30 percent. On the other hand, our analysis can achieve the maximum schedulable utilization above 60 percent. This implies that, with our analysis, industries can better utilize their given resources by accommodating more transactions for advanced features. The other way around is also true—the same set of transactions can be implemented with lower speed resources, which can save the unit cost of production.

Another important factor that affects the analysis accuracy is the period ratio of high and low-priority transactions. In order to study this factor, Fig. 10 compares the maximum schedulable utilization as varying the period selection range $[p_{min}, p_{max}]$ from $[100 \text{ ms}, 100 \text{ ms}]$ to $[100 \text{ ms}, 5,000 \text{ ms}]$ while fixing the transaction length $L = 10$. When the period ratio $p_{max}/p_{min}$ is not that large, our per-resource-based analysis can significantly reduce the double counts made by per-job analysis since Case I of Fig. 6 is common. Thus, our per-resource analysis can achieve a much higher maximum schedulable utilization than the per-job analysis methods, i.e., Tindell, WCDO, and Henia. However, as the period ratio $p_{max}/p_{min}$ becomes large, Case II of Fig. 6 becomes common, and thus, the gap decreases. When the period ratio $p_{max}/p_{min}$ is extremely large as 50, our analysis eventually degenerates into Tindell. However, in practical applications such as automotive systems, many transactions have comparable periods with reasonably small period ratios. Thus, our analysis can have significant improvements in many practical settings.

One interesting observation from the above experiments is that there is no significant improvement by WCDO and Henia over Tindell. This is because the improvement is possible only when a low-priority job can be fit into the
intertask time gaps, called offsets, of a high-priority transaction. Such cases did not commonly happen in our previous experimental setting. In order to give a favor to WCDO and Henia, in the next experiment, we pick execution times of the lowest priority transaction $C_3$ from very short values in $[0.01 \text{ ms}, 0.05 \text{ ms}]$. In addition, for other transactions to have large intertask offsets on CAN, their execution times on ECUs are picked from large values in $[10 \text{ ms}, 50 \text{ ms}]$ but those on CAN are picked from medium values in $[1 \text{ ms}, 5 \text{ ms}]$. With this special setting, Fig. 11 compares the end-to-end response time $e\text{RspTime}_3$ of the lowest priority transaction $\Gamma_3$ as increasing the transaction length of the lowest priority transaction while fixing other transaction lengths as 10. When the length of the lowest priority transaction is 3, it visits an ECU, CAN, and another ECU, in sequence. Thus, it does not have any multiple visits. In this case, our per-resource analysis degenerates into Tindell since we do not take advantage of the intertask offsets in the total delay estimation of (6). On the other hand, WCDO and Henia explicitly consider the intertask offsets, and hence, give a slightly better estimation than ours. However, as soon as the lowest priority transaction length becomes 5 visiting CAN twice, the double count reduction by our per-resource analysis catches up the benefit of the offset consideration by WCDO and Henia. After that, our per-resource analysis gives significantly better results than WCDO and Henia.

In order to further compare OurPerResource with WCDO and Henia, we now consider a new resource model consisting of 10 resources of the same type (e.g., processors). Transactions can visit the resources in any sequence instead of alternating ECUs and CAN. We also generate sets of five transactions in a purely random way:

- Each transaction length is randomly picked from $[5, 30]$.
- Each transaction period is randomly picked from $[100 \text{ ms}, 1,000 \text{ ms}]$.
- Each task of a transaction is mapped to a randomly picked resource out of 10 resources.
- Each task’s execution time is randomly picked from $[1 \text{ ms}, 5 \text{ ms}]$.

The priorities of the transactions are determined according to their random periods. Fig. 12 shows the results for 2,000 random sets. Each dot in Fig. 12a represents a random set showing the ratio of $e\text{RspTime}_3$ by OurPerResource over that of WCDO. On the other hand, each dot in Fig. 12b shows the same ratio between OurPerResource and Henia. The x-axis is the total count that the lowest priority transaction visits the same resources more than once. When the lowest priority transaction length is small without any multiple visit, WCDO and Henia are always better than OurPerResource. However, as we increase the length of the lowest priority transaction making many revisits, our analysis is significantly better than WCDO and Henia in most cases. In addition, there is a potential that our per-resource analysis can be further improved by taking the idea of WCDO and Henia in the total delay estimation of (6) since the total delay estimation is an orthogonal issue to the viewpoint change from per-job to per-resource.

6 CONCLUSION

In this paper, we propose a fundamental change from the per-job-based analysis to the per-resource-based analysis to
find a tighter bound on the end-to-end response time of a real-time transaction over multiple resources. Instead of aggregating the per-job response times, our proposed analysis aggregates the per-resource total delays. An iterative convergence method is proposed to find the per-resource total delays at all the resources, and in turn, the end-to-end response time.

The proposed analysis handles the pessimism caused by the multiple visit problem. Therefore, it can significantly improve the analysis accuracy of the end-to-end response times, especially when there are complex long transactions, and thus, they visit the same resources many times. Our extensive analysis shows that, when the multiple visit count is large, the proposed per-resource analysis can reduce up to 77 percent of the end-to-end response time estimation by existing per-job analysis methods. This improvement of estimation makes the mathematical timing analysis to be practically applicable to emerging distributed real-time application domains such as autonomously driving vehicles and collaborating autonomous robots.

In the future, we plan to extend our per-resource analysis by addressing deadlines longer than periods and also by further improving the accuracy of total delay estimations considering intertask offsets as in WCDO [5], [6], [12]. We also plan to investigate the possibility of combining the idea of capturing pipelining effects as in [15], [16] with our per-resource analysis, in order to fundamentally address the pessimism of our jitter-based total delay estimation. Another direction of our future research is to apply the proposed analysis technique to the actual real-time transactions in automotive systems.

ACKNOWLEDGMENTS
This work was supported in part by the IT R&D program of MKE/IITA (2008-F-045-02).

REFERENCES

Man-Ki Yoon received the BS degree in computer science and engineering from Seoul National University, Korea, in 2009. He is currently working toward the MS degree in the Department of Computer Science at the University of Illinois at Urbana-Champaign. His current research interests include real-time/ embedded systems, cyber-physical systems, real-time scheduling, and assumption management for safety-critical systems. He is a student member of the IEEE.

Chang-Gun Lee received the BS, MS, and PhD degrees in computer engineering from Seoul National University, Korea, in 1991, 1993, and 1998, respectively. He is currently an assistant professor in the Department of Electrical and Computer Engineering at the Ohio State University, Columbus, from 2002 to 2006; a research scientist in the Department of Computer Science at the University of Illinois at Urbana-Champaign from 2000 to 2002; and a research engineer in the Advanced Telecom. Research Lab., LG Information and Communications, Ltd., from 1998 to 2000. His current research interests include real-time systems, complex embedded systems, cyber-physical systems, ubiquitous systems, QoS management, wireless ad hoc networks, and flash memory systems. He is a member of the IEEE and the IEEE Computer Society.

Junghee Han received the BS and MS degrees from Seoul National University and the PhD degree from the University of Michigan. She is currently working toward the MS degree in the Department of Electrical and Computer Engineering at the Ohio State University. Her research interests include grid computing, network security, and wireless sensor networks. She is a member of the IEEE.