Fault-Tolerant Hierarchical Real-Time Scheduling with Backup Partitions on Single Processor

Hyun-Wook Jin
Department of Computer Science and Engineering
Konkuk University
Seoul, Korea 143–701
Email: jinh@konkuk.ac.kr

Abstract—The resource partitioning has been suggested to provide efficient composition of multi-threaded real-time applications. Partitioning can provide reliable and flexible software upgrade as partitions are strongly isolated in terms of resources. However, there are always possibility of experiencing software faults while operating on a real plant. To avoid entering a hazardous state due to a partition that is yet to be fully verified, we can deploy a backup partition that may implement inefficient algorithms or limited features but is verified with respect to reliability. The backup partition performs failover to carry out missions of the corresponding primary partition when a software fault is detected. There have been significant researches for fault-tolerant real-time scheduling but considerations for partitioned systems have not been studied. In this paper, we extend the resource model for hierarchical real-time scheduling to support primary and backup partitions. Our model can support context-dependent and context-independent tasks in the backup partition efficiently. In addition, we provide the schedulability analysis for suggested model.

I. INTRODUCTION

In vehicular Cyber-Physical Systems (CPS), software controls many electronic components in real-time. For example, an automobile is equipped with over 100 Electric Control Units (ECUs), which manage powertrain, chassis, body, safety, and infotainment systems. The internal system architecture becomes very complicated as the number of electronic devices in vehicles continues to increase, which makes difficult to efficiently resolve issues of Size, Weight, and Power (SWaP). To simplify the physical system architecture and address SWaP issues, there is a growing demand for consolidating multiple in-vehicle software applications within a single computing device [1], [2].

The concept of partitioning has been introduced to provide efficient composition of multi-threaded real-time applications. Partitioning provides a framework that reserves system resources, such as processor and memory, for each real-time application. Open software platforms, such as AUTOSAR [3] and ARINC 653 [4], also define support for partitioning. Hierarchical real-time scheduling [5], [6], [7], [8] is the core technology for realizing temporal partitioning. In the hierarchical scheduling, the partition scheduler (a.k.a., global scheduler) assigns computation resources across partitions according to their period and execution time. In the second level, the task scheduler (a.k.a., local scheduler) runs processes of a partition during the time window given by the partition scheduler.

Partitioning can provide reliable and flexible software upgrade as partitions are strongly isolated in terms of resources; thus, a partition can be changed or newly inserted transparently as far as schedulability test is passed. For example, a flight control program running as a partition on an avionics system can be changed with a new version that may implement better control algorithms without impact on other partitions. However, there are always possibility of experiencing software faults while operating on a real plant, though safety and correctness of software are rigorously verified through several steps of development process by means of formal verification, hardware-in-the-loop simulation, etc.

To avoid facing a hazardous state due to a new partition that is yet to be fully verified, we can run a backup partition that may implement inefficient algorithms or limited features but is verified with respect to safety for a long period of time. The backup partition performs failover to carry out missions of the corresponding primary partition when a fault (e.g., deadline miss or no heartbeat) is detected. There have been significant researches for fault-tolerant real-time scheduling [9], [10], [11], [12], [13], [14], [15] but considerations for partitioned systems have not been studied. Though Hyun and Kim [16] recently introduced a fault-tolerant hierarchical real-time scheduling, they focused on adding a recovery job into a partition.

In this paper, we extend the resource model for hierarchical real-time scheduling to support primary and backup partitions and provide schedulability analysis. We classify tasks in a backup partition into context-dependent and context-independent tasks based on whether the tendency of recent computation and control results (i.e., context) affects the next behavior of a task. In order to correctly reflect the recent tendency in control after failover, the context-dependent tasks have to run as background even while the primary partition works fine. In this paper, we target single processor environment as the first step.

The rest of the paper is organized as follows: In Section II, we present a basic system model, extend the existing resource model, and formally state problems addressed in this paper. In Section III, we provide schedulability analysis and an example. Finally, we conclude this paper in Section IV.
II. System Model and Problem Statement

A. Basic System Model

The basic system model in this paper is based on Shin and Lee [17]. In the hierarchical real-time scheduling, the scheduling unit (i.e., partition) \( S_i \) is defined as \( S(W_i, \Gamma_i, A_i) \), where \( W_i, \Gamma_i, \) and \( A_i \) represent workload, resource model, and scheduling algorithm, respectively. The workload \( W_i \) can be defined as a set of tasks \( \{T_1, T_2, \ldots, T_n\} \). We use the periodic task model that defines a task \( T_i \) as \( T(p_i, e_i) \), where \( p_i \) and \( e_i \) are the period and execution time of task \( T_i \), respectively. Similarly, we also use the periodic resource model \( \Gamma_i(\Pi_i, \Theta_i) \) for partitions, where \( \Pi_i \) and \( \Theta_i \) represent the period and supply time of resources for partition \( S_i \), respectively. We assume the rate-monotonic scheduling algorithm for both partition and process scheduling. The lower \( i \) value means the higher priority for both task and partition.

The supply-bound function \( sbf_{\Pi}(t) \) calculates the minimum resource supplies that resource \( \Pi \) can provide during time interval \( t \) as follows:

\[
\text{sbf}_{\Pi}(t) = \begin{cases} 
(t - (k + 1)(\Pi - \Theta)), & \text{if } t \in ([k + 1]\Pi - 2\Theta, (k + 1)\Pi - \Theta], \\
(k - 1)\Theta, & \text{otherwise},
\end{cases}
\]

where \( k = \max([(t - (\Pi - \Theta))/\Pi], 1) \).

The demand-bound function \( dbf_{\Pi}(W, t) \) calculates the maximum resource demand that workload \( W \) can request during time interval \( t \) under the scheduling algorithm \( A \). Since we assume the rate-monotonic scheduling algorithm as mentioned earlier, \( dbf_{\Pi RM}(W, t) \) is defined as follows:

\[
dbf_{\Pi RM}(W, t) = e_i + \sum_{T_k \in hp(W_i)} \left[ \frac{t}{p_k} \right] \cdot e_k,
\]

where \( hp(W_i) \) represents workloads that have higher priority than \( W_i \).

Thus, a scheduling unit \( S(W_i, \Gamma_i, A_i) \) is schedulable if \( \forall t \ sbf_{\Pi}(t) \geq dbf_{\Pi RM}(W_i, t) \). We refer Shin and Lee [17] for the proof.

B. Fault Model

In our model, a primary partition and its corresponding backup partition are indexed consecutively. Thus, a set of scheduling units can be represented as \( \{S_1, S_2, \ldots, S_{2k-1}, S_{2k}, \ldots, S_{n-1}, S_n\} \), where \( S_{2k-1} \) and \( S_{2k} \) denote primary and backup partitions, respectively. It is noteworthy that we assume that the primary and backup partitions run different versions of software; therefore, \( S_{2k-1} \) and \( S_{2k} \) may have different workload and resource model.

For a backup partition \( S_{2k} \), \( CDT(W_{2k}) \) defines a set of context-dependent tasks in \( W_{2k} \), and \( CIT(W_{2k}) \) defines a set of context-independent tasks in \( W_{2k} \). Thus \( CDT(W_{2k}) \cup CIT(W_{2k}) = W_{2k} \). As we have mentioned earlier, context-dependent tasks have to run all the time because these tasks need to keep track of the tendency of recent situation (e.g., sensor values), while context-independent tasks become active only if the primary partition fails.

Each pair of partitions is in either primary, recovery, or backup mode as shown in Fig. 1. In the primary mode, \( W_{2k-1} \) and \( CDT(W_{2k}) \) are active. In the backup mode, whole \( W_{2k} \) is active but \( W_{2k-1} \) is stopped due to a fault. Once the state becomes the backup mode, it does not switch back to the primary mode supposing that the primary partition includes an unrecoverable software fault. We assume that only one fault can happen for \( \Pi_{\max} | \Pi_{\max} = \max(\Pi_i), i = 1, \ldots, n \). In the recovery mode, \( W_{2k} \) becomes active and finish its jobs by \( \Pi_{2k} \). We assume that a fault happens at the end of the execution of a job because this is the worst case in the sense that the time left until \( \Pi_{2k} \) becomes minimal.

We extend the resource model \( \Gamma_i \) to describe primary and backup modes, which is denoted as \( \Gamma_i(\Pi_i, \Theta^p_i, \Theta^b_i) \), where \( \Theta^p_i \) and \( \Theta^b_i \) represent resource supply time in primary mode and backup mode, respectively. Therefore, \( \Theta^b_i = 0 \) for \( i = 2k - 1 \) (i.e., primary partition), and \( \Theta^b_i \leq \Theta^p_i \) for \( i = 2k \) (i.e., backup partition). In the perspective of traditional resource model, \( \Gamma_{2k-1}(\Pi_{2k-1}, \Theta^p_{2k-1}) \) and \( \Gamma_{2k}(\Pi_{2k}, \Theta^b_{2k}) \) are supplied in the primary mode. On the other hand, \( \Gamma_{2k-1}(\Pi_{2k-1}, \Theta^b_{2k-1}) \) is supplied in the backup mode. We assume \( \Pi_{2k-1} \leq \Pi_{2k} < \Pi_{2k+1} \).

C. Problem Statement

In this paper, we try to address following two problems:

**Schedulability analysis for CIT(W_{2k}) during a recovery phase:** As we have mentioned, tasks in \( CIT(W_{2k}) \) have to finish their jobs by \( \Pi_{2k} \) when a fault occurs in \( W_{2k-1} \). However, since the partition that has a higher priority also can run before \( \Pi_{2k} \) as shown in Fig. 1 (white rectangles and \( B_{2k-2} \)), we analyze the schedulability of \( CIT(W_{2k}) \) for recovery mode as follows:

\[
V_{2k} \geq \sum_{T_i \in CIT(W_{2k})} e_i,
\]

where \( V_{2k} \) denotes the resource idle time (i.e., vacant time) observed at the level \( 2k \) during the time interval between fault detection point and the deadline of fault recovery.

**Schedulability analysis of partitions:** Workloads and resource requirements are changed as mode changes from primary mode to backup mode. Thus, we need to analyze schedulability of partitions accordingly. In addition, while a fault recovery is performed, other partitions still have to meet their deadline.
III. Hierarchical Real-Time Scheduling for Backup Partitions

In this section, we analyze the schedulability. Table 1 summarizes notations used throughout the paper.

A. Schedulability of CITs during a Recovery Phase

As we have discussed in Section II.C, tasks in CIT($W_{2k}$) are schedulable during a recovery phase if they satisfy Equation (1). The vacant time $V_{2k}$ can be calculated as follows:

$$V_{2k} = \Pi_{2k} - R_{2k} - B_{2k-2}(R_{2k}, \Pi_{2k}),$$

where $R_{2k}$ denotes the worst-case response time of partition $S_{2k}$ when it is in the primary mode. $B_{2k}(R_{2i}, t_e)$ denotes cumulative busy time during time interval $[R_{2i}, t_e)$ for $S_i$. Fig. 1 also shows $V_{2k}$ and $B_{2k-2}$. Although Fig. 1 shows a case that $V_{2k}$ and $B_{2k-2}$ are consecutive time slots, they can be fragmented.

The worst-case response time $R_{2k}$ of $S_{2k}$ in the primary mode can be calculated as follows, which is the same with that of rate-monotonic algorithm [18], [19], [20]:

$$R_{2k} = \sum_{\forall j | j \leq k} \max(U_{2j-1}^p(R_{2k} + U_{2j}^p(R_{2k}), U_{2j}^b(R_{2k}))) + U_{2k-1}^p(R_{2k}) + \Theta_{2k}^p,$$

where $U_p^i(t)$ and $U_b^i(t)$ represent over-estimated worst-case work during the time interval $t$ for partition $S_i$ in the primary mode and backup mode, respectively. Since we do not know which mode demands more resources, we take the maximum value between primary and backup modes for each pair of high-priority partitions. $U_p^i(t)$ and $U_b^i(t)$ can be calculated as follows:

$$U_p^i(t) = \left[ \frac{t}{\Pi_i} \right] \cdot \Theta_i^p$$

and

$$U_b^i(t) = \left[ \frac{t}{\Pi_i} \right] \cdot \Theta_i^b.$$

The cumulative busy time $B_i$ in Equation (2) is calculated as follows:

$$B_{2k}(R_{2i}, t_e) = \sum_{\forall j | j \leq k} \max(U_{2j-1}^p(bti_{2j-1}(R_{2i}, t_e)) + U_{2j}^b(bti_{2j}(R_{2i}, t_e)),$$

where $bti_{2k}(R_{2i}, t_e)$ is the refined busy time interval of $S_{2k}$ for $R_{2i}$ and is defined as follows:

$$bti_{2k}(R_{2i}, t_e) = \begin{cases} t_e - R_{2i}, & \text{if } k > i, \\ \left[ \frac{R_{2i}}{\Pi_k} \right] \cdot \Pi_k, & \text{if } t_e \geq \left[ \frac{R_{2i}}{\Pi_k} \right], \Pi_k, \\ 0, & \text{otherwise}. \end{cases}$$

In order to reduce pessimism, if $k \leq i$, we consider the time interval $[\left[ \frac{R_{2i}}{\Pi_k} \right], \Pi_k, t_e)$ instead of $[R_{2i}, t_e)$ because $S_k$ has already executed for the deadline $[\left[ \frac{R_{2i}}{\Pi_k} \right], \Pi_k$ assuming the critical instant phasing.

B. Schedulability Test of Partitions

In order to analyze schedulability of partitions, we first need to guarantee that supplied resources can fulfill demands from workloads for both primary and backup modes. This can be done easily by utilizing $sbf_{IR}(t)$ and $dbf_{IR}(W_{2k}, t)$ described in Section II.A as follows:

$$sbf_{2k-1}(t) \geq dbf_{RM}(W_{2k-1}, t),$$

$$\Gamma_{2k-1} = \Gamma(\Pi_{2k-1}, \Theta_{2k-1}^p),$$

$$sbf_{2k}(t) \geq dbf_{RM}(CDP(W_{2k}), t),$$

$$\Gamma_{2k} = \Gamma(\Pi_{2k}, \Theta_{2k}^p),$$

$$sbf_{2k}(t) \geq dbf_{RM}(W_{2k}, t),$$

$$\Gamma_{2k} = \Gamma(\Pi_{2k}, \Theta_{2k}^p).$$

We now consider the recovery mode, where tasks in CIT($W_{2k}$) have to run and finish their jobs until $\Pi_{2k}$ as we have described in Section II.B. In the critical instant phasing, partitions in a lower priority cannot begin until all the tasks in higher priority partitions are completed. Thus, when a fault occurs at level $2k - 1$, the execution of partitions that have a lower priority than $2k$ is additionally delayed as much as $\sum t_{j \in CIT(W_{2k})} t_j$. Moreover, partitions with higher priority can preempt the delayed execution, which delays further the execution of lower-priority partitions as shown in Fig. 2.

### Table I. Notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIT($W_i$)</td>
<td>Context-dependent tasks in a task set $W_i$.</td>
</tr>
<tr>
<td>CIT($W_{2k}$)</td>
<td>Context-independent tasks in a task set $W_{2k}$.</td>
</tr>
<tr>
<td>$V_{2k}$</td>
<td>Vacant time observed at the level $2k$ during time interval $[R_{2k-1}, \Pi_{2k})$.</td>
</tr>
<tr>
<td>$R_{2k}$</td>
<td>Worst-case response time of $S_{2k}$ in the primary mode.</td>
</tr>
<tr>
<td>$U_i^p(t)$</td>
<td>Over-estimated worst-case work during time $t$ for $S_i$ in the primary mode.</td>
</tr>
<tr>
<td>$U_i^b(t)$</td>
<td>Over-estimated worst-case work during time $t$ for $S_i$ in the backup mode.</td>
</tr>
<tr>
<td>$B_{2k}(R_{2i}, t_e)$</td>
<td>Cumulative busy time during time interval $[R_{2i}, t_e)$ for $S_i$, $\ldots$, $S_{2k}$.</td>
</tr>
<tr>
<td>$bti_{2k}(R_{2i}, t_e)$</td>
<td>Refined busy time interval for $S_{2k}$.</td>
</tr>
</tbody>
</table>
TABLE II. EXAMPLE PARTITIONS

<table>
<thead>
<tr>
<th>Partitions</th>
<th>Tasks</th>
<th>Period</th>
<th>Exec. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$T_1$</td>
<td>40</td>
<td>4</td>
</tr>
<tr>
<td>$T_2$</td>
<td>80</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>$T_3$</td>
<td>160</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>$S_2$</td>
<td>$T_1$</td>
<td>55</td>
<td>4</td>
</tr>
<tr>
<td>$T_2$</td>
<td>80</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>$S_3$</td>
<td>$T_1$</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>$T_2$</td>
<td>40</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 3. Example

this figure, black rectangle represents execution time of CITs at 2k. White boxes represent $B_{2k}(R_{2k}, \Pi_{2k+1})$. Therefore, when an error occurs at level 2k − 1, lower-priority partitions are schedulable, if $\forall i \mid i > k$,

$$\Pi_{2i} - \left( R_{2k} + \sum_{j \in CIT(W_{2k})} e_j + B_{2i}(R_{2k}, \Pi_{2i}) \right) \geq 0. \quad (5)$$

C. Example

We consider a simple example of four partitions $S_1$, $S_2$, $S_3$, and $S_4$. Tasks for each partition are shown in Table II, and resource models are $\Gamma_1(5, 1, 5, 0)$, $\Gamma_2(15, 4, 5)$, $\Gamma_3(20, 2, 0)$, and $\Gamma_4(20, 0, 0)$. In this example we have only one backup partition (i.e., $S_2$), and partition $S_4$ is a dummy. In partition $S_2$, $T_1$ is a CIT while $T_2$ is a CDT. Thus, $T_1$ does not run in the primary mode. When a fault occurs at $S_1$, $V_2$ and $\sum_{j \in CIT(W_{2k})} e_j$ are calculated as 15 − 7 − 0 = 8 and 4, respectively. Thus, these partitions satisfy Equation (1) ($8 \geq 4$) as shown in Fig. 3. That is, $T_1$ of $S_2$ can meet the deadline $\Pi_2$ when a fault occurs at $S_1$. We also test schedulability of $S_3$ by evaluating Equation (5) as 20 − (7 + 4 + 9) = 0. It is noteworthy that $B_2(7, 20)$ should be 5 as shown in Fig. 3 but the value calculated by Equation (3) is 7 because this equation always chooses the maximum value of work though $S_1$ and $S_2$ are in the backup mode.

IV. CONCLUSIONS AND FUTURE WORK

In this paper, we extended the resource model for hierarchal real-time scheduling to support primary and backup partitions. We classified processes in the backup partition into context-dependent tasks and context-independent tasks based on whether a task decides its behavior according to the recent tendency of computation and control. The context-dependent tasks have to run even when the primary partition works correctly in order to accurately reflect the recent tendency in control after failover. We also analyzed schedulability with extended resource model.

As future work, we intend to relax the assumption that a fault can occur at most one every $\Pi_{max}$ time unit. We also plan to carry out simulations with various cases and to implement the suggested model in a real partitioning operating system.

REFERENCES