Optimization of Fault-Tolerant Distributed Real-Time Systems

Zebo Peng, Viacheslav Izosimov, Petru Eles, Paul Pop
Embedded Systems Lab (ESLAB)
Linköping University

Soft Errors

Transient faults

- Occur for a short time
- Lead to corruptions of data in memory, and miscalculation in logic
- Do not cause a permanent damage of the hardware
Soft Errors

Transient faults
- Electromagnetic interference
- Crosstalk
- Radiation
- Power supply fluctuations
- Internal EMI

The Problems

- Soft error rates have been increasing rapidly due to:
  - Continuous device scaling.
  - Increased clock frequency.
  - Voltage scaling.

- Classical fault tolerance techniques are not very efficient here:
  - Significant hardware overhead.
  - Many soft errors are masked at architecture level, therefore low-level solutions can be over-killed.
  - May cause the missing of real-time deadlines.
Optimization of FT Real-Time System

- Hard real-time applications
  - Time-constrained
  - Cost-constrained
  - Fault-tolerant
  - etc.

The Need for Design Optimization of Embedded Systems with Fault Tolerance

Outline

- Motivation
- The optimization problems
- Scheduling with fault tolerance requirements
- Checkpoint optimization
- Conclusions
Basic Fault Tolerance Mechanisms

Re-execution

Error-detection overhead $\alpha$

Recovery overhead $\mu$

Rollback recovery with checkpointing

Checkpointing overhead $\chi$

Active replication

Issues to be Considered

- Design optimization together with fault tolerance consideration:
  - Policy assignment strategy: which mechanism to be used for each process.
  - Process mapping: in general, and for active replication.
  - Optimization of the number of checkpoints.
- The main focus of our work:
  - Global strategy to integrate several optimizations issues in the framework of static cyclic scheduling for distributed real-time systems.
Integration into System-Level Design

- System Specification
- Architecture/platform Selection
- Mapping
- Scheduling
- Feedback loops
- Fault Tolerance Techniques

Fault-Tolerant Time-Triggered Systems

- Processes: Re-execution, Active Replication, Rollback Recovery with Checkpointing
- Messages: Fault-tolerant predictable protocol

Maximum k transient faults within each application run (system period)
Scheduling

\[ k = 2 \]

\[
\begin{array}{c}
P_1 \\
m_1 \\
P_2
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{true} & \text{false} & \text{Error-detection overhead } \alpha \\
P_1 & 0 & 40 \\
P_2 & \\
\end{array}
\]

Conditional Scheduling

\[ k = 2 \]

\[
\begin{array}{c}
P_1 \\
m_1 \\
P_2
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{true} & \text{false} & \text{Error-detection overhead } \alpha \\
P_1 & 0 & 40 \\
P_2 & \\
\end{array}
\]
Conditional Scheduling

\[ k = 2 \]

\[ \begin{align*}
\text{true} & \quad F_{R_1} & \quad F_{R_2} & \quad F_{R_1} \land F_{R_2} & \quad \overline{F_{R_1}} \land \overline{F_{R_2}} & \quad F_{R_1} \land \overline{F_{R_2}} & \quad \overline{F_{R_1}} \land F_{R_2} & \quad F_{R_1} \land F_{R_2} & \quad \overline{F_{R_1}} \land F_{R_2} \land F_{R_3} \\
\text{P}_1 & \quad 0 & \quad 45 & \quad 90 & & & & & \\
\text{P}_2 & \quad 40 & \quad 130 & \quad 85 & \quad 140 & \quad 95 & \quad 150 & & \\
\end{align*} \]

The Worst Execution Time = 150 + 50 = 200

Conditional Schedule Tables

\[ k = 2 \]

\[ \begin{align*}
\text{true} & \quad F_{R_1} & \quad F_{R_2} & \quad F_{R_1} \land F_{R_2} & \quad \overline{F_{R_1}} \land \overline{F_{R_2}} & \quad F_{R_1} \land \overline{F_{R_2}} & \quad \overline{F_{R_1}} \land F_{R_2} & \quad F_{R_1} \land F_{R_2} & \quad \overline{F_{R_1}} \land F_{R_2} \land F_{R_3} \\
\text{P}_1 & \quad 0 & \quad 45 & \quad 90 & & & & & \\
m_1 & \quad 40 & \quad 130 & \quad 85 & & & & & \\
\text{P}_2 & \quad 50 & \quad 140 & \quad 95 & \quad 150 & \quad 105 & \quad 160 & & \\
\end{align*} \]
Fault-Tolerance Conditional Process Graph

\[ k = 2 \]

Conditional Scheduling Strategy
- List scheduling algorithm
- Partial critical path as priority function

Conditional Scheduling
- Conditional scheduling:
  - Generates short schedules
  - (Allows to trade-off between transparency and performance)
    - Requires a lot of memory to store schedule tables
    - The controllers will be very complex
    - Scheduling algorithm is very slow
- **Alternative:** shifting-based scheduling
Shifting-based Scheduling

- All processes, mapped on the same computation node, are executed with a fixed order.
- Messages sent over the bus will be scheduled at a fixed time (frozen message → transparency)
- Faults on one computation node will not affect the execution on other computation nodes.

+ Requires less memory
+ Schedule generation is very fast
- Schedules will be longer

Ordered FT-CPG

\( k = 2 \)
Root Schedules

Each message will be sent on one single fixed time.

Extracting Execution Scenarios
Memory Required for Conditional S Tables

<table>
<thead>
<tr>
<th>% of frozen ms</th>
<th>20 proc.</th>
<th>40 proc.</th>
<th>60 proc.</th>
<th>80 proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>k=1</td>
<td>k=2</td>
<td>k=3</td>
<td>k=1</td>
</tr>
<tr>
<td>100%</td>
<td>0.13</td>
<td>0.28</td>
<td>0.54</td>
<td>0.36</td>
</tr>
<tr>
<td>75%</td>
<td>0.22</td>
<td>0.57</td>
<td>1.37</td>
<td>0.62</td>
</tr>
<tr>
<td>50%</td>
<td>0.28</td>
<td>0.82</td>
<td>1.94</td>
<td>0.82</td>
</tr>
<tr>
<td>25%</td>
<td>0.34</td>
<td>1.17</td>
<td>2.95</td>
<td>1.03</td>
</tr>
<tr>
<td>0%</td>
<td>0.39</td>
<td>1.42</td>
<td>3.74</td>
<td>1.17</td>
</tr>
</tbody>
</table>

In Kbytes

Applications with more frozen nodes require less memory

Memory Required to Store Root Schedule

<table>
<thead>
<tr>
<th></th>
<th>20 proc.</th>
<th>40 proc.</th>
<th>60 proc.</th>
<th>80 proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>k=1</td>
<td>k=2</td>
<td>k=3</td>
<td>k=1</td>
</tr>
<tr>
<td>100%</td>
<td>0.016</td>
<td>0.034</td>
<td>0.054</td>
<td>0.070</td>
</tr>
</tbody>
</table>

Shifting-based scheduling requires very little memory

But it generate ~15% worse schedules than conditional scheduling.
**Checkpoint Optimization**

- $N_1$
- $P_1$
- $P_1^1$
- $P_1^2$
- $P_{1/1}^2$
- $P_{1/2}^2$

**Locally Optimal Number of Checkpoints**

- $\mu_1 = 15$ ms
- $k = 2$
- $\chi_1 = 5$ ms
- $\alpha_1 = 10$ ms
- $\beta_1 = 15$ ms
- $P_1$
- $C_1 = 50$ ms

No. of checkpoints:

1. $P_1$
2. $P_1^1$, $P_1^2$
3. $P_1^1$, $P_1^2$, $P_1^3$
4. $P_1^1$, $P_1^2$, $P_1^3$, $P_1^4$
5. $P_1^1$, $P_1^2$, $P_1^3$, $P_1^4$, $P_1^5$
Global Optimization vs. Local Optimization

Does the optimization reduce the fault tolerance overheads on the schedule length?

4 nodes, 3 faults

Application size (the number of tasks)

Global Optimization of Checkpoint Distribution (MC)
Local Optimization of Checkpoint Distribution (MCO)

Conclusions

- **Scheduling with fault tolerance requirements**
  - Two novel scheduling techniques
  - Fast scheduling alternative with low memory requirements for schedules.

- **Design optimization with fault tolerance**
  - Optimization of the number of checkpoints

Approaches and algorithms have been evaluated on the large number of synthetic applications and a real life example - vehicle cruise controller