Mixed Criticality Systems: Beyond Transient Faults

Abhilash Thekkilakattil, Alan Burns, Radu Dobrin and Sasikumar Punnekkat
Motivation and Contribution

- State of the art of mixed criticality scheduling mainly focuses on WCET overruns.

- WCET overruns are one example of transient faults.

- We propose an approach for design and scheduling of mixed criticality systems under permanent faults.
Introduction

- Mixed criticality scheduling deals with scheduling real-time tasks with varying levels of WCET assurances.

- Growing interest in mixed criticality scheduling since Vestal’s RTSS’07 paper.
  - 230 citations according to Google Scholar.
  - Over 200 follow-up papers according to “Mixed Criticality Systems- A Review” (6th ed.) by Burns and Davis.
Goals of Mixed Criticality Scheduling

- Enable **certification** by different **certifying authorities**
  - Demonstrate **timeliness** under different WCETs

- Enable **efficient utilization** of the underlying computing infrastructure
  - Enabling **safe** sharing of the computing infrastructure
  - Ensuring **isolation** of critical from less critical tasks
State of the Art Mixed Criticality Scheduling

- Criticality monotonic priority ordering
- Adaptive and static scheduling
- Scheduling with virtual deadlines/periods
- Mixed criticality scheduling under faults
The Dependability Perspective

Dependability

Attributes
- Reliability
- Safety
- Maintainability
- Confidentiality
- Integrity
- Availability

Means
- Fault tolerance
- Fault Prevention
- Fault Removal
- Fault Forecasting

Threats
- Faults
- Errors
- Failures

Focus of MC scheduling

Faults, Errors and Failures

Fault \rightarrow Error \rightarrow Failure

- A bit flip
- Wrong computed value
- Incorrect actuation
- WCET overrun
- Task deadline miss
- High criticality deadline miss

Many different types of faults (except WCET overruns) are not covered by Vestal-like models.
Classification of Faults

Faults

Transient Faults
- Fault whose presence is **limited** in time
- Examples include bit flips and WCET overruns
- Solution: **temporal redundancy** e.g., task re-executions

Permanent Faults
- Fault whose presence is **continuous** in time
- Examples include memory and processor failures
- Solution: **spatial redundancy** e.g., using additional hardware
Transient Fault Tolerance

Temporal redundancy: replicate the tasks in time

- **Re-execute** the task
- Execute an **alternate** task

The time for re-execution/alternate task execution can be seen as the “**extra time**” needed in Vestal’s model.
Classification of Faults

Fault

 transient faults

• Fault whose presence is limited in time
• Examples include bit flips and WCET overruns
• Solution: temporal redundancy e.g., task re-executions

permanent faults

• Fault whose presence is continuous in time
• Examples include memory and processor failures
• Solution: spatial redundancy e.g., using additional hardware

Focus of this paper
Focus of this Paper

How to design mixed criticality real-time architectures to tolerate permanent faults?

Contribution:

1. Propose a fault coverage based mapping of criticalities
2. Present a taxonomy of fault tolerance mechanisms in the context of mixed criticality systems
Classification of Permanent Faults

- **Design Faults**
  - Faults due to deficiencies in design and development e.g., manufacturing defects in computers
  - **Hardware** and **software** design faults

- **Random Faults**
  - Faults whose **time** of occurrence nor the **cause** can be determined e.g., faults due to wear and tear

- **Byzantine faults**
  - Faults in which **replicas** behave arbitrarily differently
  - **Worst** kind of faults: requires high amount of redundancy
Tolerating Permanent Faults

Requires additional hardware (N-modular paradigm)
- Replicate the tasks on multiple hardware
- Perform voting to determine and mask failures
- Diversity to prevent common cause failures
Goals of Mixed Criticality Scheduling

- Enable **certification** by different **certifying authorities**
  - Demonstrate **timeliness** under different WCETs

Timeliness **does not** imply certification

Safety standards **mandate** redundancy for safety

- Enable **efficient utilization** of the underlying computing infrastructure
  - Enabling **safe** sharing of the computing infrastructure
  - Ensuring **isolation** of critical from lesser critical tasks
Goals of Mixed Criticality Scheduling

- Enable **certification** by different **certifying authorities**
  - Demonstrate **timeliness** under different WCETs

**Timeliness does not imply certification**

**Safety standards mandate** redundancy for safety

- Enable **efficient utilization** of the underlying computing infrastructure
  - Enabling **safe** sharing of the computing infrastructure
  - Ensuring **isolation** of critical from lesser critical tasks

**Highest level of “protection” for all tasks?**
## Mapping Criticalities Based on Fault Coverage

<table>
<thead>
<tr>
<th>Criticality</th>
<th>Transient Faults</th>
<th>Random Faults</th>
<th>Software Faults</th>
<th>Hardware Faults</th>
<th>Byzantine Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Medium</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Low</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Non-critical</td>
<td>Partially covered</td>
<td>Partially covered</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
High Criticality Tasks

- Dedicated hardware to guarantee isolation
- $3b+1$ replicas and byzantine fault tolerance mechanism to tolerate $b$ byzantine faults
- Hardware and Software diversity to protect against design faults
Medium Criticality Tasks

- High integrity hardware that is shared among medium criticality tasks
- Time triggered scheduling and lock-step execution
- Replication for protection against random faults
- Hardware and software diversity for protection against design faults
Low Criticality Tasks

- COTS hardware, e.g., a multicore processor, that is shared among low criticality tasks
- Time aware voter and loose synchronization: less development effort
- Replication for protection against random faults
- Software diversity for protection against software design faults
Non-Critical Tasks

• Scheduled along with low criticality tasks
• Timeliness is guaranteed in the absence of faults
• Discarded upon failures
• Possibility of using existing MC scheduling algorithms
• Guarantees isolation of higher criticality tasks
• Limited form of redundancy can be provided exploiting spare processing capacity
## Mapping Criticalities Based on Fault Coverage

### Criticality Matrix

<table>
<thead>
<tr>
<th>Criticality</th>
<th>Transient Faults</th>
<th>Random Faults</th>
<th>Software Faults</th>
<th>Hardware Faults</th>
<th>Byzantine Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>redundancy</td>
<td>redundancy</td>
<td>software diversity</td>
<td>hardware diversity</td>
<td>byzantine fault tolerance</td>
</tr>
<tr>
<td>Medium</td>
<td>redundancy</td>
<td>redundancy</td>
<td>software diversity</td>
<td>hardware diversity</td>
<td>x</td>
</tr>
<tr>
<td>Low</td>
<td>redundancy</td>
<td>redundancy</td>
<td>software diversity</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Non-critical</td>
<td>Limited redundancy</td>
<td>Limited redundancy</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

### Design Faults

- **Redundancy:** Redundancy measures the system's ability to maintain functionality in the presence of faults.
- **Software Diversity:** Diversity in software reduces the risk of failure by ensuring that different parts of the system are less likely to fail simultaneously.
- **Hardware Diversity:** Diversity in hardware increases system reliability by providing multiple paths for data to travel, reducing single points of failure.
- **Byzantine Fault Tolerance:** The system's ability to identify and handle malicious or incorrect inputs from faulty components.
Conclusions

• **Approach** for design of mixed criticality systems in the context of **permanent faults** through:
  – Fault **coverage** based mapping of criticalities
  – Criticality based provisioning of resources
  – Isolation of higher criticality tasks
  – Implicit coverage of WCET overrun faults

• **Future Work**
  – Methods for **efficient allocation of replicas to processors**
  – Consideration of **safety analysis** in the allocation and scheduling of tasks
  – Providing **better-than-average** service to non-critical tasks
Thank You!

Questions?