Temporal Partitioning and Verification in Distributed Cyber-Physical Systems

Dionisio de Niz
Bjorn Andersson

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Why Is Timing Verification Important?
Why Temporal Protection?
Boeing 787 Suppliers

FAA Needs Modular Certification

Source: Boeing / Reuters
Temporal Partitioning in Distributed CPS

Critical to Verify Timing of CPS

- Late interaction with physical world can be catastrophic

Complexity of CPS Requires Modularization

- Large CPS integrated from components from different organizations
- Prevent modification of component from affecting all system

Everything is Distributed

- Any medium-size CPS runs on a network of computers
OS Dual Objective

Enhanced Machine

Multiple Machines

File System

Disk

Icons credit: http://www.doublejdesign.co.uk
Time-Sharing CPU – Round robin

Same time requirement – Fair Scheduling

Icons credit: http://www.doublejdesign.co.uk
Fixed-Priority Scheduling + Rate Monotonic

Different Requirements – Guaranteed Timing

Icons credit: http://www.doublejdesign.co.uk
Periodic Server

“Shape” Execution

Icons credit: http://www.doublejdesign.co.uk
Periodic Server

Blackout Interval

Icons credit: http://www.doublejdesign.co.uk
Deferrable Server

Back-to-Back Preemption

Icons credit: http://www.doublejdesign.co.uk
Sporadic Server

No back-to-back but multiple replenishment pieces

Icons credit: http://www.doublejdesign.co.uk
Resource Reserves OS Abstraction

- RT-Mach
- Linux/RK
- CBS
- ZSRM
- Time-Triggered (ARINC 653)


Icons credit: http://www.doublejdesign.co.uk
What Type of Temporal Protection?
Boeing 787 Suppliers

FAA Needs Modular Certification

Source: Boeing / Reuters
Mixed-Criticality

FAA / CO-178C

- Criticality ≡ Failure Condition Categories
  - Catastrophic
  - Hazardous
  - Major
  - Minor
  - No Safety Effects

- Higher The Failure Consequence Higher Level of Correctness Certainty

- Low-criticality allowed lower correctness certainty iff prevented from interfering with high-criticality

Scheduling (S. Vestal)

- Higher Estimate of Worst-Case Execution Time => Higher Confidence of Correctness

- Protect Higher-Criticality Tasks from Lower-Criticality Ones
  - ASYMMETRIC PROTECTION!
Consolidation of Mixed-Criticality Tasks

Shared Hardware
Can lead to cycle stealing
Consolidation of Mixed-Criticality Tasks

To avoid interference add temporal protection
Consolidation of Mixed-Criticality Tasks

BUT
Symmetric protection leads to *criticality inversion*
Rate-Monotonic Priority

Shorter Period $\rightarrow$ Higher Priority

- Ideal utilization

BUT: Poor Criticality Protection Due to Criticality Inversion

- If criticality order is opposite to rate-monotonic priority order

![Diagram showing Rate-Monotonic Priority and Criticality Protection]

- High Criticality
  - 10/100
  - 100

- Low Criticality
  - 1/10
Criticality As Priority Assignment (CAPA)

Higher Criticality $\rightarrow$ Higher Priority

- Ideal criticality protection:
  - lower criticality cannot interfere with higher criticality

BUT: Poor Utilization Due to Priority Inversion

- If criticality order is opposite to rate-monotonic priority order
Zero-Slack Scheduling

Tasks with:
- Period (T), Normal (C), and Overloaded (C°) Execution Times

Start with rate-monotonic scheduling

Calculate the last instant before $\tau_{HC}$ misses its deadline
- this is called the **zero-sack** instant

Switch to criticality-as-priority
- Splits the execution window into
  - Normal mode (RM)
  - Critical mode (CAPA)

Zero-Slack Scheduling

Tasks with:
- Period (T), Normal (C), and Overloaded (C°) Execution Times

Start with rate-monotonic scheduling

Calculate the last instant before τ_{HC} misses its deadline
- this is called the zero-slack instant

Switch to criticality-as-priority
- Splits the execution window into
  - Normal mode (RM)
  - Critical mode (CAPA)

In Overload

\[ \tau_{LC} = (2,2,4) \]
\[ \tau_{HC} = (2.5,5,8) \]

Zero-Slack Scheduling

Tasks with:
- Period (T), Normal (C), and Overloaded (C⁰) Execution Times

Start with rate-monotonic scheduling

Calculate the last instant before \( \tau_{HC} \) misses its deadline
- this is called the zero-slack instant

Switch to criticality-as-priority
- Splits the execution window into
  - Normal mode (RM)
  - Critical mode (CAPA)

Without Overload

\[ \tau_{LC} = (2,2,4) \]
\[ \tau_{HC} = (2.5,5,8) \]

ZSRM Properties

Subsumes RM
- If criticalities are aligned to priorities
- No critical mode

Subsumes CAPA
- If not enough slack, only critical mode

Graceful Degradation
- In overloads, deadlines are missed in reverse criticality order
Implementation

ZSRM

Scheduling algorithm calculates zero-slack instants offline

Linux/ RK

- Resource reservation in Linux
  - CPU, Net, Mem, Disk
- Bundled into resource sets that provide a form of virtual machine
- Multiple implementations
  - Nano/RK for sensor networks

Special Zero-Slack Reserves

- Switch to critical mode
  - Stop lower-criticality tasks on zero-slack instant
- Tasks in critical mode in stack

Library prototype implementation for VxWorks
What if criticality exhibits diminishing returns?

<table>
<thead>
<tr>
<th>Task</th>
<th>Period</th>
<th>Criticality</th>
<th>WCET</th>
<th>NCET</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 ) Surveillance Cov.</td>
<td>4</td>
<td>Mission</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>( t_2 ) Collision Avoid.</td>
<td>8</td>
<td>Safety</td>
<td>5</td>
<td>2.5</td>
</tr>
</tbody>
</table>

What if criticality exhibits diminishing returns?

![Diagram](image)
Reclaiming Resources in Mixed-Criticality Systems

<table>
<thead>
<tr>
<th>Task</th>
<th>Period</th>
<th>Criticality</th>
<th>WCET</th>
<th>NCET</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$ Surveillance Cov.</td>
<td>4</td>
<td>Mission</td>
<td>2</td>
<td>2</td>
<td>{2,2.5}</td>
</tr>
<tr>
<td>$t_2$ Collision Avoid.</td>
<td>8</td>
<td>Safety</td>
<td>5</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>$t_3$ Amount of Intelligence</td>
<td>4</td>
<td>Mission</td>
<td>2</td>
<td>2</td>
<td>{2,2.5}</td>
</tr>
</tbody>
</table>

Reclaimed Resources
Using Reclaimed Resources to Maximized Utility

<table>
<thead>
<tr>
<th>Task</th>
<th>Period</th>
<th>Criticality</th>
<th>WCET</th>
<th>NCET</th>
<th>Utility Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$ Surveillance Cov.</td>
<td>4</td>
<td>Mission</td>
<td>2</td>
<td>2</td>
<td>{2, 2.5}</td>
</tr>
<tr>
<td>$t_2$ Collision Avoid.</td>
<td>8</td>
<td>Safety</td>
<td>5</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>$t_3$ Amount of Intelligence</td>
<td>4</td>
<td>Mission</td>
<td>2</td>
<td>2</td>
<td>{2, 2.5}</td>
</tr>
</tbody>
</table>

Utility Diminishes: $\text{Utility} \neq \text{Criticality}$
## Using Reclaimed Resources to Maximized Utility

### Table: Task Period Criticality WCET NCET Utility Levels

<table>
<thead>
<tr>
<th>Task</th>
<th>Period</th>
<th>Criticality</th>
<th>WCET</th>
<th>NCET</th>
<th>Utility Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁ Surveillance Cov.</td>
<td>4</td>
<td>Mission</td>
<td>2</td>
<td>2</td>
<td>{2,2.5}</td>
</tr>
<tr>
<td>t₂ Collision Avoid.</td>
<td>8</td>
<td>Safety</td>
<td>5</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>t₃ Amount of Intelligence</td>
<td>4</td>
<td>Mission</td>
<td>2</td>
<td>2</td>
<td>{2,2.5}</td>
</tr>
</tbody>
</table>

### Diagram

- **t₁**: Mission critical task with WCET 2 and NCET 2, utility levels {2,2.5}.
- **t₂**: Safety critical task with WCET 5 and NCET 2.5.
- **t₃**: Mission critical task with WCET 2 and NCET 2, utility levels {2,2.5}.

### ZS-QRAM: More mission-critical utility from same resources

Total Utility = 4
D. de Niz, L. Wrage, N. Storer, A. Rowe, and R. Rajkumar.
End-to-End Timing Requirements

Sensors: Trust, Airspeed

Calculate Stall risk

Display Stall alarm

Display stall alarm before too late to correct

Movement

Move continuously or Stop before collision
From Single Processor Preemptions

![Diagram of single processor preemptions](image)
To Preemptions in Different Processors (Pipeline)

Processor 1

Task1

Task2

Processor 2

Task1

Task2

Calculate
Stall risk

Processor 3

Task1

Task2

Display
Stall alarm

Sensors:
Trust, Airspeed

Sensors:
Trust, Airspeed

Sensors:
Trust, Airspeed

Airspeed
ZSRM Pipeline
ZSRM Pipeline Performance

![Graph showing ZSRM Pipeline Performance with plots for Overload and Number of Stages]
Future Challenges

Increasingly Complex Hardware
- Massive multicores: Tile processors
- Heterogeneous multicores: hardware accelerators
- Adaptive Power: Dynamic Thermal Management

Increasingly Complex Software
- Machine Learning
  - Learning while executing could lead to unpredictable WCET
  - How to bound unpredictable behavior On Time (before crash)
- Complex distributed systems
  - Coordinated autonomy: smart highways
  - Global Internet of Things

Increasing Development Complexity
- Components from Different Suppliers
- Potential Safety Enforcement from Safety/Certification Authorities