SafeMC: A System for the Design and Evaluation of Mode Change Protocols

Tianyang Chen  Linh Thi Xuan Phan

University of Pennsylvania
Scenario: Self-driving cars

- **Mode**: a functionality, a system configuration
  - a set of tasks that need to execute
  - e.g., adaptive cruise control, lane keeping assistance, …

- **Mode transition**: a change in the system mode
  - triggered by a mode change request (MCR)
  - e.g., a driver’s request to take back control of the vehicle
Problems with mode transitions

- Need to execute:
  - Unfinished jobs of the old mode
  - Newly released jobs of the new mode

- Result: Potential overloads and deadline misses
Solution: Mode-change protocol

- Specify execution semantics of a mode transition
  - How to handle unfinished jobs?
  - When to release new jobs?
  - When to apply the modified parameters? etc.

- Goal: Timing guarantees and safety during a transition
Limitations of existing work

• Lack experimental evaluations
  – Timing may be violated in practice due to overheads

• Evaluations performed on different environments
  – Hard to compare between protocols

• Focus on relatively simple protocols
  – Agnostic of mode transitions and tasks’ safety criticality
  – Require customizations in practice

How to determine which MCP is the best-fit for a given system?
Contributions

• An extensible set of mode change primitives
  – implement a broad spectrum of mode change behaviors

• SafeMC system design
  – enables efficient specification and evaluation of MCPs

• A prototype implementation of SafeMC in Xen

• Experimental evaluations
  – comparisons of existing MCPs
  – case study: design of novel protocols specific for an application
Outline

- Introduction
- Mode change primitives
- SafeMC design and implementation
- Evaluation
- Case study
**Insight**

- MCPs are built on a common set of key primitives
  - Operate on the smallest *scheduling entity* (a job)
  - Specify the *actions* that the MCP must perform
  - Specify *conditions* under which the actions to be invoked

- Primitives are composed at different levels of granularity to form MCPs
  - Job type, task, task type, or transition

- Example: Maximum Period Offset
  - For all transitions: *transition type (global)*
    - New jobs of *job type*
    - Changed tasks of *task type*
    - Released after an offset of *action* condition
Types of tasks and jobs

- **Job type**
  - **Pending**: Unfinished jobs that are not currently executing
  - **Executing**: Currently executing jobs
  - **New**: New jobs to be released after the MCR instant
Basic primitives

- Describe actions for jobs of a specific type, for a particular task, or for all tasks of a specific type

\[
\text{Primitive} = [\text{Action}; \text{Guard}] 
\]

- Action to be applied to the considered jobs (e.g., abort)
- Conditions for when the action should be applied (e.g., after a certain offset)

- Compose primitives to form an MCP for the system
# Basic primitive: Actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTINUE</td>
<td>The jobs are continued to be scheduled with current timing attributes (WCET, deadline, period, criticality)</td>
<td>Pending or Executing job types</td>
</tr>
<tr>
<td>UPDATE</td>
<td>The jobs continued to be scheduled but timing attributes are changed to that of the new mode</td>
<td></td>
</tr>
<tr>
<td>ABORT</td>
<td>The jobs are aborted and removed from the system</td>
<td>Pending or Executing job types</td>
</tr>
<tr>
<td>ABORT[K]</td>
<td>The oldest K jobs are aborted and removed from the system</td>
<td></td>
</tr>
<tr>
<td>RELEASE</td>
<td>New jobs whose attributes are defined in the new mode will be released when the associated guard becomes true</td>
<td>New job type</td>
</tr>
<tr>
<td>RELEASE_O</td>
<td>New jobs whose attributes are defined in the old mode will be released as usual while the associated guard is true</td>
<td></td>
</tr>
</tbody>
</table>

*If an action has no guard, it will be applied immediately*
**Basic primitive: Offset guards**

- Apply the action after an offset $\Delta$ from MCR or LR instant
  - $\Delta$ is a constant threshold, or based on task’s parameters

- **Example:** [ RELEASE; OffsetLR: MAX_PERIOD ]
  - The first new job is released after a delay equal to the max period of tasks in old and new modes from the last release
Basic primitive: Backlog guards

- Specify a formula on the backlog of the job queue
  - $\text{Op} \in \{\leq, \geq, =, <, >\}$
  - $\Delta$: a constant, or based on the queue size

- Example: $[\text{ABORT}; \text{Backlog:} (\geq, 0.5 \cdot \text{MAX\_VAL})]$  
  - Abort old/pending jobs of the task if its queue is at least half full

- Example: $[\text{RELEASE}; \text{BacklogGlobal:} (=, 0)]$  
  - Release new jobs when the global queue is empty
Basic primitive: **Criticality guards**

- Specify a formula on the task’s old and new criticality

- Example: `[ RELEASE; Criticality: (<, OLD_VAL, NEW_VAL) ]`
  - Release new jobs immediately if the task’s criticality is increased

- Example: `[ ABORT; Criticality: (≤, OLD_VAL, 2) ]`
  - Abort all existing jobs with criticality less than or equal to 2

More primitives are available in the paper
## Examples of MCP specs

<table>
<thead>
<tr>
<th>JOB TYPE</th>
<th>Task Type</th>
<th>New (Action = RELEASE)</th>
<th>Exec/Pending</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>UNCHANGED</td>
<td>CHANGED</td>
</tr>
<tr>
<td>Max Period Offset</td>
<td>OffsetLR: OLD_PERIOD</td>
<td>OffsetMCR: MAX_PERIOD</td>
<td></td>
</tr>
<tr>
<td>Min Offset w/ Periodicity</td>
<td>OffsetLR: OLD_PERIOD</td>
<td>BacklogGlobal: (=, 0)</td>
<td></td>
</tr>
<tr>
<td>Min Offset w/o Periodicity</td>
<td>BacklogGlobal: (=, 0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asyn. with Periodicity</td>
<td>OffsetLR: OLD_PERIOD</td>
<td>OffsetMCR: Δ</td>
<td></td>
</tr>
<tr>
<td>Asyn. w/o Periodicity</td>
<td>OffsetMCR: Δ_i</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Outline

• Introduction

• Mode change primitives

• SafeMC design and implementation

• Evaluation

• Case study
SafeMC Design

MCP-specific algorithmic-core

- Define mode change primitives
- Easily customized or re-written from scratch

MCP-independent common runtime

- Implement the algorithm
- Simplify experimentation
- Enable fair comparisons on a real platform
SafeMC Runtime

- **User Input**

  - SafeMC spec language
  - Multi-mode system model
  - System-level protocol (globally for all transitions, or for each transition ID)
  - Task-level protocol per transition

- **System Parser**

- **Mode Change Initializer**

- **Mode Change Manager**

  - load MC structures

  - triggerMCR(domID, cpuID, transID)

- **MCR Dispatcher**

  - Run-time input from the command line, an automatic script, a user domain, ...

  - [domID, cpuID, transID]
Prototype

- Xen-based implementation of SafeMC
  - Support mode changes at the hypervisor level
  - Rationale: substantial interests in real-time virtualization, potential applications to beyond traditional RT systems,…

Dom0 toolset
- run_*.sh
  - Run a specific test
- proc_trace.sh
  - Parse traces for post processing
- load_scripts
  - Invoke parser and load transitions to hypervisor
- trigger_scripts
  - Trigger an existing transition

User space
- System Parser
- MC Initializer
- MCR Dispatcher
- Test Generator

Hypervisor
- sched_rt.c
- schedule.c
- xc_rt.c
- xenalyze.c

Mode Change Manager
Tracing tool
Outline

• Introduction
• Mode change primitives
• SafeMC design and implementation
• Evaluation
• Case study
Evaluation setup & goals

• Platform: Xen with SafeMC enabled

• Workload: Randomly-generated multi-mode systems
  – Each mode has 4/8/16/32 VCPUs and utilization ≤ 96%
  – VCPU’s WCET in [1.5ms,12ms], max period = 50ms

• Goal #1: Run-time overhead of SafeMC

• Goal #2: Protocol comparisons
  – Release-new: How fast the new mode is activated
  – Finish-old: How fast the system completely leaves the old mode
  – Finish-old-new: How fast the system completely enters the new mode

• Goal #3: Develop novel protocols for a given application
  – using a self-driving case study (based on Google’s patent)
# SafeMC Run-time overhead (ns)

Overhead averaged across 1000 measurements per protocol per multi-mode system

<table>
<thead>
<tr>
<th>#VCPUs</th>
<th>Schedule</th>
<th>Context switch</th>
<th>Release</th>
<th>Mode change</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>540.5</td>
<td>125.4</td>
<td>1025.0</td>
<td>1986.2</td>
</tr>
<tr>
<td>8</td>
<td>478.2</td>
<td>86.8</td>
<td>1465.5</td>
<td>3697.5</td>
</tr>
<tr>
<td>16</td>
<td>399.8</td>
<td>44.5</td>
<td>1900.8</td>
<td>7063.2</td>
</tr>
<tr>
<td>32</td>
<td>366.2</td>
<td>64.0</td>
<td>3267.0</td>
<td>12753.5</td>
</tr>
</tbody>
</table>

Negligible overhead (comparable to vanilla Xen’s RTDS*)

Reasonable MC overhead

[*] Xi et al. Real-time multi-core virtual machine scheduling in Xen. EMSOFT2014
Result: Protocol comparisons

Maximum mode-change delay (ms) across 1000 runs

- Asyn w/ periodicity
- Asyn w/o periodicity
- Min offset w/ periodicity
- Min offset w/o periodicity
- Max period offset

Min offset without periodicity

- Delay new releases until existing jobs complete → smallest MC delay
- Suitable for applications where periodicity is not critical
Result: Protocol comparisons

- Delay releases of new and changed tasks by an offset = max period
- Perform poorly in activating/completely entering the new mode
- Should be avoided if tasks in the new mode are highly critical
Result: Protocol comparisons

- Unchanged tasks are not affected by the MC
- **Min offset w/ periodicity performs well in entering new mode**
- Good choice when periodicity is required and the new mode is critical
# Case study: Result

*Max mode change delay (ms) across 100 runs*

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MCR&lt;sub&gt;0&lt;/sub&gt; Existing</td>
<td>32.984</td>
<td>2.005</td>
<td>39.927</td>
<td></td>
</tr>
<tr>
<td>Hybrid</td>
<td>3.441</td>
<td>15.223</td>
<td>26.176</td>
<td></td>
</tr>
<tr>
<td>MCR&lt;sub&gt;1&lt;/sub&gt; Existing</td>
<td>46.312</td>
<td>8.305</td>
<td>51.036</td>
<td></td>
</tr>
<tr>
<td>Hybrid</td>
<td>0.005</td>
<td>1.908</td>
<td>11.947</td>
<td></td>
</tr>
</tbody>
</table>

- **MCR<sub>0</sub>:** Hybrid protocol activates and completely enter the Decision mode much more quickly
- **MCR<sub>1</sub>:** Hybrid protocol outperforms in all three metrics

**SafeMC enables the design of novel MCPs specific to the applications**
Conclusion

• Choosing the right MCP is critical for safe and efficient design of a multi-mode system

• **SafeMC**: a system for designing, experimentally evaluating and comparing MCPs
  – MCP-specific algorithmic “core” that implements a broad extensible set of common mode change primitives
  – MCP-independent, shared runtime for executing and evaluating an MCP in a multi-mode system

• Benefits
  – Enable side-by-side experimental comparison
  – Support rapid prototyping and customization