Mixed-Criticality Federated Scheduling for Parallel Real-Time Tasks

Jing Li, David Ferry, Shaurya Ahuja, Kunal Agrawal, Christopher Gill, Chenyang Lu
Washington University in St. Louis
MC System in Self-Driving Cars

Features with different criticality levels (based on the severity of injuries it can cause):

- Crash avoidance features
- Route planning features
- ...
- Infotainment features
Toy Example of MC System

**High criticality task** (e.g., crash avoidance), deadline 40ms

- Work 10ms
- Most cases

**Low criticality task** (e.g. route planning), deadline 40ms

- Work 80ms

**Parallel task:**
- can utilize multiple cores at the same time
Common-Case vs. Worst-Case Scenarios

Single-criticality systems:
need to model **worst-case scenario**

**Most cases**
- core 1: 10ms
- core 2: 80ms
- core 3: 10ms
- core 4
- core 5

**Very rare cases**
- core 1: 100ms
- core 2
- core 3
- core 4
- core 5

5/5/16
MC Model Improves Resource Efficiency

**Mixed-criticality system:**

Provide different levels of real-time guarantees

- **core 1:** 80ms
- **core 2:** 10ms
- **core 3:** 100ms

**Most cases:** guarantee that **both high and low-criticality tasks** meet deadlines

**Very rare cases:** only guarantee that **high-criticality tasks** meet deadlines

5/5/16
Parallel Real-Time Task Model

Directed Acyclic Graph (DAG) task model

- General parallel task model
- Nodes: sequential subtasks
- Edges: dependences

\[ C_i = 30 \]
\[ L_i = 6 \]
Properties of Parallel Real-Time Task

Total work $C_i$: execution time on one core
Critical-path length $L_i$: execution time on $\infty$ cores

Implicit deadline (= period) $D_i = T_i$

Utilization $u_i = \frac{\text{total work } C_i}{\text{period } T_i}$

If $D_1 = 60$, $u_1 = 0.5$;
If $D_1 = 30$, $u_1 = 1$;
If $D_1 = 20$, $u_1 = 1.5$;
A scheduler $S$ provides a capacity augmentation bound of $\alpha$ if it can always schedule a task set $\tau$ on $m$ processors if:

\[
U_\Sigma \leq \frac{m}{\alpha}
\]

Notes: No scheduler can provide $\alpha < 1$. The conditions do not depend on the structure of the DAG.
Theoretical Bounds for Multicore Systems

Sequential Tasks  
Utilization bound

Parallel Tasks (general DAG)  
Capacity augmentation bound

Scheduler S can afford 95% utilization

<table>
<thead>
<tr>
<th>Task</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>0.9</td>
</tr>
<tr>
<td>$u_2$</td>
<td>0.2</td>
</tr>
<tr>
<td>$u_3$</td>
<td>0.8</td>
</tr>
<tr>
<td>$u_4$</td>
<td>0.6</td>
</tr>
<tr>
<td>$u_5$</td>
<td>1.0</td>
</tr>
<tr>
<td>$u_6$</td>
<td>1.0</td>
</tr>
</tbody>
</table>

$m = 5$

Scheduler S can afford 75% utilization*

<table>
<thead>
<tr>
<th>Task</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>0.9</td>
</tr>
<tr>
<td>$u_2$</td>
<td>0.2</td>
</tr>
<tr>
<td>$u_3$</td>
<td>0.8</td>
</tr>
<tr>
<td>$u_4$</td>
<td>1.6</td>
</tr>
<tr>
<td>$u_5$</td>
<td>0.6</td>
</tr>
<tr>
<td>$u_6$</td>
<td>1.0</td>
</tr>
</tbody>
</table>

$m = 5$

Smaller augmentation is better ($b = m/u$)

95\% utilization

95\% utilization
# Theoretical Bounds for Multicore Systems

## Single Criticality

<table>
<thead>
<tr>
<th>Sequential Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Utilization bound</strong></td>
</tr>
<tr>
<td>100%</td>
</tr>
<tr>
<td>RM-US [Baruah 1996]</td>
</tr>
<tr>
<td>EDF-US [Baker 2005]</td>
</tr>
<tr>
<td>P-EDF [Baruah 2010]</td>
</tr>
</tbody>
</table>

## Multiple Criticalities

<table>
<thead>
<tr>
<th>Parallel Tasks (general DAG)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Capacity augmentation bound</strong></td>
</tr>
<tr>
<td>100%</td>
</tr>
<tr>
<td>FS [Li 2014]</td>
</tr>
<tr>
<td>G-EDF [Li 2014]</td>
</tr>
<tr>
<td>Decomp-based [Kim 2013]</td>
</tr>
</tbody>
</table>

MC-Fluid [Baruah 2015]  
MC-P-EDF [Baruah 2013]  
MC-G-EDF [Li 2012]
## Theoretical Bounds for Multicore Systems

<table>
<thead>
<tr>
<th></th>
<th>Sequential Tasks Utilization bound</th>
<th>Parallel Tasks (general DAG) Capacity augmentation bound</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single Criticality</strong></td>
<td><img src="image" alt="100%" /></td>
<td><img src="image" alt="100%" /></td>
</tr>
<tr>
<td><strong>Multiple Criticalities</strong></td>
<td><img src="image" alt="100%" /></td>
<td><img src="image" alt="100%" /></td>
</tr>
<tr>
<td><strong>Parallel Tasks (general DAG) Capacity augmentation bound</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Contributions – Capacity Augmentation Bounds

Dual-criticality systems
- with high-utilization tasks \((u_i > 1)\): 3.41
- with both high- and low-utilization tasks: \(3.67m/(m - 1)\)

Multi-criticality systems
- with high-utilization tasks \((u_i > 1)\): 3.73

Smaller augmentation is better \((b = m/u)\)
Contributions – Graceful Degradation

Implement MCFS scheduler for OpenMP tasks
Perform both numerical and empirical evaluations

Fraction of tasks with no deadline miss (per criticality)

Number of high-criticality tasks overrun
Mixed-Criticality System Model

Two worst-case execution time estimates:

- Pessimistic overload work $C^O_i$ for certification
- Less pessimistic nominal work $C^N_i$ from empirical measurement

For tasks with implicit deadlines (deadline $D_i = period T_i$)

- Overload utilization $u^O_i = C^O_i / T_i$
- Nominal utilization $u^N_i = C^N_i / T_i$
Mixed-Criticality System Model

Two criticality levels

nominal work $C^N_i$

High-criticality tasks

overload work $C^O_i$

Low-criticality tasks

5/5/16
Two system states

Typical-state:
If each job’s actual work $\leq C^N_i$

Critical-state:
If a job’s actual work $> C^N_i$
Correctness Criterion of MC System

Schedulability guarantees

Typical-state:
If each job’s actual work $\leq C^N_i$, guarantee schedulability for both high and low-criticality tasks.

Critical-state:
If a job’s actual work $> C^N_i$, guarantee schedulability only for high-criticality tasks and could abandon low-criticality jobs.
MCFS Algorithm at a High Level

Classify tasks into three types
MCFS Algorithm at a High Level

Classify tasks into three types

For each task in each type, calculate and assign:

1) virtual deadline
MCFS Algorithm at a High Level

Classify tasks into three types

For each task in each type, calculate and assign:
(1) virtual deadline
(2) dedicated cores in typical state

$m$ cores

Typical-state

Type 1 task
Type 3 task
Type 2 task

dedicated cores in typical state
MCFS Algorithm at a High Level

Classify tasks into three types

For each task in each type, calculate and assign:

1. virtual deadline
2. dedicated cores in typical state
3. dedicated cores in critical state
MCFS Algorithm at a High Level

**Challenge:** how to maximize utilization while guaranteeing tasks to meet deadlines in both states?

Jointly design task classification, virtual deadline and core assignments

$m$ cores
**MCFS Task Classification**

**High-utilization tasks:** require parallelism to meet deadline

<table>
<thead>
<tr>
<th>Task Type</th>
<th>Criticality</th>
<th>Nominal Utilization</th>
<th>Overload Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>LO-High (LH)</td>
<td>LO</td>
<td>High $1 &lt; u_{N_i}$</td>
<td>NA</td>
</tr>
<tr>
<td>HI-Moderate-High (HMH)</td>
<td>HI</td>
<td>Moderate $1/(b-1) &lt; u_{N_i} \leq u_{O_i}$</td>
<td>High $1 &lt; u_{O_i}$</td>
</tr>
<tr>
<td>HI-VeryLow-High (HVH)</td>
<td>HI</td>
<td>VeryLow $1/(b-1) \leq u_{N_i}$</td>
<td>High $1 &lt; u_{O_i}$</td>
</tr>
</tbody>
</table>

Proved capacity augmentation bound $b = 3.14$
MCFS – Virtual Deadline

How to distinguish jobs with nominal and overload work?
When to transition from typical to critical state?

Challenge: do not know the work of a job before execution

<table>
<thead>
<tr>
<th>core 1</th>
<th>0</th>
<th>10ms</th>
<th>40</th>
<th>400</th>
<th>30ms</th>
<th>440</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5/5/16
MCFS – Virtual Deadline

How to distinguish jobs with nominal and overload work?
When to transition from typical to critical state?

**Requirements:**
- Must be able to finish nominal work before the virtual deadline.
MCFS – Virtual Deadline

How to distinguish jobs with nominal and overload work?
When to transition from typical to critical state?

First used in single core mixed-criticality scheduler EDF-VD [Baruah 2012]

Does not complete before VD
→ job must overrun its nominal work

Transition to critical state
MCFS – Virtual Deadline

How to distinguish jobs with nominal and overload work?
When to transition from typical to critical state?

For low criticality tasks, ignore overload jobs → virtual deadline = actual deadline

First used in single core mixed-criticality scheduler EDF-VD [Baruah 2012]
How to distinguish jobs with nominal and overload work?
When to transition from typical to critical state?

Only check if job completed by virtual deadline, but do not need to monitor job’s actual execution time

⇒ assign dedicated cores to each task

First used in single core mixed-criticality scheduler EDF-VD [Baruah 2012]
MCFS – Dedicated Cores

With given virtual deadline, assign dedicated cores in both states

Dedicated cores guarantee progress for parallel tasks
MCFS – Dedicated Cores

How to meet virtual deadline when having nominal work?

Requirement of virtual deadline: must be able to finish nominal work before the virtual deadline.
MCFS – Dedicated Cores

How to meet virtual deadline when having nominal work?
How to meet actual deadline when having overload work?

Dedicated cores $n^O_i$ guarantee to complete *remaining work* $C^O_i - C^N_i$ in D - VD

Progress guarantee of virtual deadline:
must be finished $C^N_i$ work by virtual deadline
How Many Dedicated Cores?

Calculate minimum number of cores $n$ to complete remaining work $C$ and remaining critical-path length $L$ within time $D$

Critical-path length $L$ of a parallel task: execution time on infinite number of cores

\[ n = \left\lfloor \frac{C - L}{D - L} \right\rfloor \]

From federate scheduling paradigm [Li 2014]
Why Assign Dedicated Cores?

Guarantee progress in typical state before transition and in critical state after transition.

Virtual deadline $D'_i$

Single dedicated core $n^N_i = 1$

Dedicated cores $n^O_i$

$$n^O_i = \left\lfloor \frac{(C_i-D'_i) - L_i}{(D_i-D'_i) - L_i} \right\rfloor$$
Challenge: Balance Two Core Assignments

\( n^N_i \) in typical state guarantees schedulability of nominal work
\( n^O_i \) in critical state guarantees schedulability of remaining work

Virtual deadline affects core assignments in two states

\[
\begin{align*}
\text{More cores} & \quad C^N_i & \quad C^O_i - C^N_i & \quad \text{Shorter } D'_i \\
& & & \\
\text{More cores} & \quad C^N_i & \quad C^O_i - C^N_i & \quad \text{Longer } D'_i
\end{align*}
\]
Intuition for Balancing Core Assignments

Typical state core assignment: $b^N = n^N_i / u^N_i$
Critical state core assignment: $b^O = n^O_i / u^O_i$

Capacity augmentation bound $b > \max\{b^N, b^O\}$ (smaller is better)
$\Rightarrow b^N = b^O = b$

Virtual deadline $D'_i$

$bu^N_i$  $bu^O_i$
Distinguish HVH and HMH Tasks

<table>
<thead>
<tr>
<th>Task Type</th>
<th>Criticality</th>
<th>Nominal Utilization ( u^{N_i} )</th>
<th>Overload Utilization ( u^{O_i} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>HVH</td>
<td>High</td>
<td>( \frac{1}{b-1} )</td>
<td>( 1 &lt; ) ( u^{O_i} )</td>
</tr>
</tbody>
</table>

**Challenge of HVH tasks:**

\[ D_i = 40, \ C^N_i = 4 \rightarrow u^{N_i} = 0.1 \]

Single dedicated core \( n^{N_i} = 1 \) to guarantee progress

Capacity augmentation bound

\[ b > b^N = \frac{n^{N_i}}{u^{N_i}} = 10 \]

Very small utilization \( u^{N_i} = C^N_i / D_i < 0.5 \)
Jointly Design VD and Core Assignment

Jointly design task classification, virtual deadline and core assignments to get best resource efficiency (capacity bound)

Virtual deadline

Typical-state

Critical-state

\[ \text{HMH task} \quad (b-1)u^N_i + u^O_i \]

\[ \text{LH task} \quad (b-1)u^N_i \]

\[ \text{HVH task} \quad u^O_i \]

\[ \text{HMH task} \quad bu^O_i \]

\[ \text{HVH task} \quad bu^O_i \]

\[ m \text{ cores} \]
Conclusion

MCFS assigns virtual deadline and dedicated cores to parallel mixed-criticality tasks

MCFS has good capacity augmentation bounds under various conditions

MCFS supports graceful degradation

Implement MCFS for OpenMP tasks