An Algorithm for Scheduling Certifiable Mixed-Criticality Sporadic Task Systems

Haohan Li        Sanjoy Baruah
Department of Computer Science
The University of North Carolina at Chapel Hill
Tasks are in **multiple criticality levels** on safety-related embedded systems

**Certification** is required for each criticality

Classic real-time scheduling algorithms don’t work efficiently
Motivation: An Example

Typical real-time system model

Certification:
Checking temporal correctness of safety-critical tasks
Motivation: An Example

- The problem with certification
  - The WCET is estimated
  - Schedulability relies on the estimation

![Diagram showing task scheduling with different criticalities](image-url)
The problem with certification

Certifiers may use more pessimistic estimations

- Non-critical task
- Safety-critical task
- Safety-critical task
The problem with certification

- In this case, the system fails to pass certification
  - Though we know that T₂ and T₃’s using so much time is nearly impossible
The problem with certification

We can use criticality monotonic to pass the certification

Diagram:

- Non-critical task
- Safety-critical task
- Safety-critical task

T1, T2, T3

Regular
(2,1)
(5,1)
(5,2)

Pessimistic
(5,1)
(5,2)
Motivation: An Example

- The problem with certification

We can use **criticality monotonic** to pass the certification

- Non-critical task
- Safety-critical task
- Safety-critical task

Tasks:
- \( T_1 \) with criticality (2,1)
- \( T_2 \) with criticality (5,1)
- \( T_3 \) with criticality (5,2)

Categories:
- Regular
- Pessimistic
The problem with certification

We can use **criticality monotonic** to pass the certification
Motivation: Dilemma

- **Urgency and importance may differ**

  - **Non-critical task**
  - **EDF/RM**: Hard to pass certification
  - **Safety-critical task**

  - **T1**: Low criticality, Earlier deadline
  - **T2**: High criticality, Later deadline

- **Balance**:
  - **Huge utilization loss**
  - **Criticality Monotonic**
  - **High criticality Later deadline**
Motivation: Balance

Balanced Schedule:
With **pessimistic** estimations, it passes certification;
With **normal** estimations, it guarantees schedulability.
Formal definition of the mixed-criticality sporadic task system

Period: $T_i$
Formal definition of the **mixed-criticiality sporadic task system**

Deadline: \( D_i \)
Formal definition of the mixed-criticality sporadic task system

Deadline: \( D_i \)
Formal definition of the **mixed-criticality** sporadic task system

Criticality: $x_i$

- $x_1 = \text{LOW}$
- $x_2 = \text{HIGH}$
- $x_3 = \text{HIGH}$
The key idea is to specify multiple WCETs for different criticality levels

**WCET at each criticality:** \([C_i(A), C_i(B), \ldots ]\)

- **Low-criticality task**
- **High-criticality task**
- **High-criticality task**

- \((P, D, x, [C(H), C(L)])\)
  - \((2,2,LOW,[1,1])\)
  - \((5,5,HIGH,[2,1])\)
  - \((5,5,HIGH,[2,1])\)
Correctness of mixed-criticality schedule

Temporal correctness of a schedule:
All tasks at level X and above should meet their deadlines if no task exceeds its level-X WCET.
Correctness of mixed-criticality schedule

Temporal correctness of a schedule:
All tasks at level X and above should meet their deadlines if no task exceeds its level-X WCET

T_2 and T_3 only use low-criticality execution time, we’ll guarantee T_1’s correctness
Correctness of mixed-criticality schedule

Temporal correctness of a schedule:
All tasks at level X and above should meet their deadlines if no task exceeds its level-X WCET

- Low-criticality task
- High-criticality task
- High-criticality task

$T_2$ and $T_3$ use high-criticality execution time, we are at certification scenario
Correctness of mixed-criticality schedule

Temporal correctness of a schedule: All tasks at level X and above should meet their deadlines if no task exceeds its level-X WCET.

- T₁ is sacrificed because T₂ and T₃ overrun.
- T₂ and T₃ use high-criticality execution time, we are at certification scenario.
Correctness of mixed-criticality schedule

Temporal correctness of a schedule:
All tasks at level X and above should meet their deadlines if no task exceeds its level-X WCET.
Correctness of mixed-criticality schedule

Temporal correctness of a schedule:
All tasks at level X and above should meet their deadlines if no task exceeds its level-X WCET
Correctness of mixed-criticality schedule

Temporal correctness of a schedule:
All tasks at level X and above should meet their deadlines if no task exceeds its level-X WCET
Correctness of mixed-criticality schedule

Temporal correctness of a schedule:
All tasks at level X and above should meet their deadlines if no task exceeds its level-X WCET
Informal interpretation of correctness

- **WCET Specification**
  - High
  - Mid
  - Low

- **Temporal Correctness**
  - High

- **Behavior**
  - Actual execution times in a given run

- **Schedule**
  - No deadline miss in specified criticality

---

An Algorithm for Scheduling Certifiable Mixed-Criticality Sporadic Task Systems, RTSS 2010
Informal interpretation of correctness

- **Behavior**
  - High
  - Mid
  - Low

- **WCET Specification**
  - High
  - Mid
  - Low

- **Temporal Correctness**
  - High
  - Mid
  - Low

- **Schedule**
  - No deadline miss in specified criticality

Actual execution times in a given run
Informal interpretation of correctness

- **Behavior**
  - High
  - Mid
  - Low

- **WCET Specification**
  - High
  - Mid
  - Low

- **Temporal Correctness**
  - High
  - Mid
  - Low

- **Schedule**
  - No deadline miss in specified criticality

Actual execution times in a given run
Previous Results

- The problem of scheduling mixed-criticality independent jobs has been solved
  - Baruah, Li, and Stougie. RTAS 2010
  - Li and Baruah. EMSOFT 2010

- We call our solution “OCBP algorithm”
  - Own Criticality-Based Priority algorithm
Previous Results

- OCBP algorithm generates a static priority list for a set of mixed-criticality independent jobs
  - It requires specified release-time and deadline for each job
  - It guarantees mixed-criticality correctness
  - It has a speedup factor 1.618 for two-criticality job sets
    - More criticality levels has higher speedup factor
    - Scheduling optimally is NP-hard in the strong sense
  - It runs in $O(n \log n)$ time in the number of jobs
Main Contribution

- In this paper, we apply OCBP algorithm to the mixed-criticality sporadic task systems
  - Speedup factors are maintained
  - A sufficient load-based schedulability test is also maintained
The principle of OCBP algorithm is to recursively find the lowest-priority job

- Very similar to “Audsley’s Approach”

- If a level-X job would meet its deadline as the lowest-priority job, when all jobs execute at their level-X WCETs, then this job can be assigned lowest priority
  
  - Repeat this procedure to get a full priority list
  
  - When such lowest-priority job can not be found, claim the job set unschedulable
The principle of OCBP algorithm is to recursively find the lowest-priority job.

Determine if a job can get the lowest priority: Compute *maximum time demand* in its criticality.
The principle of OCBP algorithm is to recursively find the lowest-priority job.

For J₄, the maximum possible time demand (in the worst case) exceeds its deadline.
**The principle of OCBP algorithm is to recursively find the lowest-priority job**

For $J_2$, the maximum time demand is smaller (if $J_3$ or $J_4$ overruns, $J_2$ can miss its deadline)
The principle of OCBP algorithm is to recursively find the **lowest-priority** job.

Assign $J_2$ the lowest priority, and delete it from the job set.
The principle of OCBP algorithm is to recursively find the **lowest-priority** job.

Now the maximum time demand for $J_4$ decreases ($J_2$ won’t affect $J_4$ now).

Now $J_4$ can get the lowest priority.
The principle of OCBP algorithm is to recursively find the lowest-priority job.

Repeat the procedure, we can get a priority list: $J_3 < J_1 < J_4 < J_2$.
From Jobs to Sporadic Tasks

- **Sporadic tasks have infinite jobs**
  - But the processor can’t be infinitely busy
    - We assume that the processor isn’t fully utilized

- **To apply the job-oriented OCBP algorithm, we consider jobs in the longest busy interval**
  - The number of jobs in the longest busy interval is bounded
    - The number of jobs is pseudo-polynomial to the number of tasks
From Jobs to Sporadic Tasks

- We run OCBP algorithm in the beginning of the longest busy interval
  - We specify every job to get the earliest possible release-time and deadline
    - Assuming that the sporadic tasks run periodically
- A priority list for all jobs in the longest busy interval will be generated
  - This priority list may be eventually erroneous
    - Because jobs are not released immediately
We proved that the priority list can be erroneous only after the following two violations:

- When the processor is idle
- When a high-priority job is released, and preempts a low-priority job
We proved that the priority list can be erroneous only after the following two violations:

- We also proved that when the violation happens, we can re-compute the priority list:
  - Use OCBP algorithm again
  - The re-computation will always generate a valid priority list
We proved that the priority list can be erroneous only after the following two violations:

- We also proved that when the violation happens, we can **re-compute** the priority list.

Inductively, we’ve proved that:

For a mixed-criticality sporadic system, if OCBP algorithm succeeds in the beginning, there exists a correct scheduling algorithm.
An example of running OCBP algorithm on mixed-criticality sporadic tasks
Example of running OCBP algorithm in the beginning

A priority list (in the beginning):

\[ T_{2,1} < T_{1,1} < T_{3,1} < T_{1,2} < T_{1,3} \]
Example of executing tasks following the OCBP priority list

When no violation happens, follow $T_{2,1} < T_{1,1} < T_{3,1} < T_{1,2} < T_{1,3}$
OCBP Algorithm on Tasks

- Example showing when a violation happens

Violation: When the processor is idle
Example showing when a violation happens

Violation: When the processor is **idle**

\[ T_{2,1} < T_{1,1} < T_{3,1} < T_{1,2} < T_{1,3} \]
Example showing when a violation happens

Violation: When the processor is idle

\[ T_{2,1} < T_{1,1} < T_{3,1} < T_{1,2} < T_{1,3} \]

re-compute

\[ T_{2,1} < T_{1,2} < T_{3,1} < T_{1,3} < T_{1,4} \]
Example showing when a violation happens

Violation: When the processor is **idle**

Low-criticality task

High-criticality task

High-criticality task
Violation: When the processor is idle

- The re-computation is valid
The re-computation is **valid**

The situation for $T_1$, $T_2$ and $T_3$ is no worse than the beginning.
OCBP Algorithm on Tasks

Example showing when a violation happens

Violation: When a high-priority job is released, and preempts a low-priority job.
The re-computation is also **valid**

- T_{3,1} is guaranteed to meet its deadline when T_{1,1} and T_{2,1} preempt it
- The situation for T_{1,1} and T_{2,1} is no worse than the beginning
Our scheduling algorithm for mixed-criticality sporadic task system

- Attempt to compute a priority list for jobs in the longest busy interval using OCBP algorithm
  - The schedulability will be determined

- Execute jobs according to the priority list
- If any following violations happen, re-compute the priority list using OCBP algorithm
  - When the processor is idle
  - When a high-priority job preempts a low-priority job
Performance Measurement

- **Speedup factor**
  - $\Phi = 1.618$ for OCBP algorithm with two criticalities
    - $\Phi = 2$ for EDF with two criticalities
    - $\Phi = \infty$ for criticality-monotonic
  - The result for multiple criticalities is available at
    - *Baruah et al.* MFCS 2010
Schedulability test

- OCBP algorithm is itself a schedulability test
  - It runs in pseudo-polynomial time to number of tasks
- We also proved that for any two-criticality system
  \[ l_{LO}^2 + l_{HI} \leq 1 \]
  is a sufficient schedulability test
  - This schedulability test dominates currently known EDF schedulability test for mixed-criticality systems
Run-time performance

- Every time a new job arrives, re-computation may happen
- The re-computation could require pseudopolynomial time in the worst case
Future Work

- **Scheduling algorithm with better run-time performance**
- **Multiprocessor scheduling**
- **Non-preemptions**
Load-Based Schedulability Test

- **Schedulability test**
  - For any two-criticality system
    \[ l_{LO}^2 + l_{HI} \leq 1 \]
    is schedulability test for OCBP
  - And
    \[ l_{LO} + l_{HI} \leq 1 \]
    is schedulability test for EDF

Figure 1: Bound on the LO-criticality load \( l_{LO} \) as a function of HI-criticality load \( l_{HI} \).
Our previous speedup factor result

Figure 1: Bound on the $\ell_{LO}$-criticality load ($\ell_{LO}$) as a function of $\ell_{HI}$-criticality load ($\ell_{HI}$).
Figure 1: Bound on the $LO$-criticality load ($\ell_{LO}$) as a function of $HI$-criticality load ($\ell_{HI}$).

EDF can definitely schedule.
Load-Based Schedulability Test

Figure 1: Bound on the $\ell_{\text{LO}}$-criticality load ($\ell_{\text{LO}}$) as a function of $\ell_{\text{HI}}$-criticality load ($\ell_{\text{HI}}$).
We can try OCBP as a test

Figure 1: Bound on the LO-criticality load ($\ell_{LO}$) as a function of HI-criticality load ($\ell_{HI}$).
Load-Based Schedulability Test

Figure 1: Bound on the LO-criticality load ($\ell_{LO}$) as a function of HI-criticality load ($\ell_{HI}$).

No known test exists for EDF.