An Exact Schedulability Test for Non-Preemptive Self-Suspending Real-Time Tasks

Beyazit Yalcinkaya  Mitra Nasri  Björn Brandenburg
The paper in a nutshell

The paper provides the **first exact schedulability test** for the following open research problems:

**Global multiprocessor**

- fixed-priority scheduling of non-preemptive tasks
- limited-preemptive tasks
- limited-preemptive segmented self-suspending tasks

**Uniprocessor**

- fixed-priority scheduling of 

For tasks with **bounded yet non-deterministic**

- Execution time
- Suspension time
- Release jitter

![Diagram](image-url)
Segmented self-suspending task model

A rich model to express systems that use/have

- hardware accelerators (GPUs, co-processors, etc.)
- intensive I/O accesses
- computation offloading (to the cloud, edge, etc.)

Task

- Execution
- Suspension
- Waiting to be scheduled

CPU

- Execution on GPU

GPU

Network 1

Network 2

Timeline:
- Release
- Deadline
- Time

Transmission
Why is analyzing self-suspending tasks hard?

Classic “worst-case release” scenarios cannot be used for self-suspending tasks

From Chen et al. 2018:

(a) Without suspension

(b) With suspension

The release pattern that causes the worst-case interference

Deadline miss
Why is analyzing self-suspending tasks hard?

Suspension-oblivious analysis is unsafe

(i.e., under limited-preemptive scheduling, treating suspension segments as if they were execution segments is unsound)

From this paper:

**(a) suspension oblivious**

```
Task 1
1 3 6 7
Task 2
2 6 9
Task 3
0 3
```

**(b) suspension aware**

```
Task 1
1 3 6 7
Task 2
2 4 7
Task 3
0 3
```

Deadline miss

This counter example is valid for both periodic and sporadic limited-preemptive tasks.
Current challenges

Industry is rapidly moving towards more complex execution models (including self-suspending tasks)

Prior work is focused on sufficient (pessimistic) schedulability tests

Even without self-suspensions, there is no exact analysis for global limited-preemptive scheduling

Given the lack of an exact test, there is no way to know how good or bad the existing tests are

State of the art on self-suspending tasks is not advancing fast enough
Designing an exact test: where to start?

Schedulability analysis problem in real-time systems  
Map to  
Reachability problem in timed automata

Of course, we are not the first to observe this!


Some of the existing analyses based on timed automata use “stop watches” (e.g., David et al 2009)

This makes the reachability problem undecidable (in practice, these tests are only sufficient and very inaccurate)

Other analyses use models that allow for impossible priority inversions and hence are pessimistic (for periodic tasks)

Examples in the paper
Designing an exact test: high-level idea

Model Task, Scheduler, and the Event Synchronizer as timed automata.

(simplified) task automaton:
Designing an exact test: high-level idea

More details in the paper
Experiments
EVALUATION

Questions:

• How much schedulability gain is achieved using our exact analysis?

• How far does the analysis scale w.r.t.
  • Number of tasks
  • Number of processors
  • Number of code segments
  • Length of self-suspensions

Considered task models:

• Segmented self-suspending limited-preemptive tasks

• Limited-preemptive tasks

• Non-preemptive tasks
**Limited-preemptive tasks**

Utilization=30%

10 tasks

- **this paper: 1 core**
- **Serrano et al. 2017: 1 core**

![Graph showing schedulability ratio vs. segments for limited-preemptive tasks with utilizations of 30% and 10 tasks, comparing this paper's results with Serrano et al. 2017's results.](image-url)
Limited-preemptive tasks

Utilization=30%
10 tasks

The true schedulability increases with the increase in the number of cores
Limited-preemptive tasks

Utilization=30%
10 tasks

Serrano’s test becomes very pessimistic when there are multiple cores.

The true schedulability increases with the increase in the number of cores, while Serrano’s test shows the opposite!
Non-preemptive scheduling

4 cores, 30% utilization

- `this paper`
- `this paper (timeout)`
Non-preemptive scheduling

4 cores, 30% utilization

Here, Nasri et al.’s test is as good as the exact one

Nasri et al.’s test is 3 order of magnitude faster!

Nasri et al.’s test explores the space of possible schedules efficiently (with the help of schedule abstraction and effective path merging techniques).

Conclusions

This paper: An extensible timed automata model in UPPAAL that provides the \textbf{first} exact schedulability tests for

Global multiprocessor fixed-priority scheduling of non-preemptive tasks

limited-preemptive tasks

limited-preemptive segmented self-suspending tasks

Uniprocessor fixed-priority scheduling of

In restricted settings, some of the existing tests are \textbf{almost as accurate as the exact test} while being much faster

There is a \textbf{large gap} between the accuracy of various sufficient tests and the new exact baseline

Exact tests can \textbf{quantify the pessimism} of the existing sufficient (but faster) tests
Questions

Beyazit Yalcinkaya, Mitra Nasri, Björn Brandenburg

Thank you
Scalability w.r.t. the number of tasks and cores
(non-preemptive tasks)

This paper

3 orders-of-magnitude difference!

Nasri et al. 2018

Timeout limit was set to 1 hour.