Schedulability analysis of globally scheduled preemptive applications

Srinidhi Srinivasan

Supervisor: Dr. Geoffrey Nelissen
Co-supervisor: Dr. Mitra Nasri
Safety critical systems

Even under the worst case scenarios, the timing constraints of the system are met.

Worst-case response time (WCRT) < Deadline → Schedulable
What is the problem being tackled?

Workload Model
- Periodic preemptive tasks

Platform Model
- Multiprocessor platform

Scheduler Model
- Global job-level-fixed-priority

How to find the worst-case response time (WCRT)?
What are tasks?

**Tasks** → A task is a piece of code that implements one of the system functionalities.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Execution time variation [Min,Max]</th>
<th>Release jitter</th>
<th>Period</th>
<th>Deadline</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[4,5]</td>
<td>1</td>
<td>10</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

Task 1

Job 1

Task 1

Job 2

Task 1

Job 3

Schedulability analysis of globally scheduled preemptive parallel applications
Jobs in an observation window

Observation window
Smallest duration of time after which the schedules formed by these jobs repeat themselves

Hyperperiod – lowest common multiple of the periods of all tasks
What are preemptive tasks/jobs?

A preemptive task/job can be interrupted.

Job 1 (High priority)  Job 2 (Low priority)

Execution can be interrupted.
Global job level fixed priority scheduler

- Priority is assigned to each job based on parameters like period or deadline
- Many algorithms follow such a scheduler:
  - Rate-Monotonic
  - Deadline-Monotonic
  - Earliest Deadline First
  - etc.
Build a schedulability analysis technique for preemptive tasks scheduled by a job-level fixed priority scheduler on a multiprocessor platform
Schedulability analysis of globally scheduled preemptive parallel applications

Sufficient and exact solutions

Universal set of all tasksets

Set of all schedulable tasksets by algorithm A

A subset of schedulable tasksets by algorithm A

Exact solutions

Our goal

Sufficient solutions
State of the art – Multiprocessor case

**Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks**


- Exact by using finite state machines
- Only if period of task is 3, 4 or 5
- State space explosion occurs very soon

*Scalability issues*

A. Burmyakov, E. Bini, and E. Tovar,
“An exact schedulability test for global FP using state space pruning”

V. Bonifaci and A. Marchetti-Spaccamela, “Feasibility analysis of sporadic real-time multiprocessor task systems”
State of the art – Multiprocessor case

Uniprocessor platform with preemptive tasks

Response-Time Analysis (RTA) (Audsley et al., 1995)

Multiprocessor platform with preemptive tasks

Marko Bertogna and Michele Cirinei. 2007. “Response-time analysis for globally scheduled symmetric multiprocessor platforms”


Many more...

+ Can handle multiple types of tasksets
+ Very fast in analysis

- A sufficient solution
- Highly inaccurate
Build a solution that bridges the gap

- Accurate
- Ability to scale to realistic system sizes

Finite state machine solutions
- Exact
- Only work for tasks with periods 3, 4 or 5

RTA based solutions
- Very fast
- Extremely inaccurate
Schedule abstraction graph [1]

What is it?
• Non-preemptive jobsets
• Considers job orderings of schedules to build a schedule abstraction graph

Why does it provide hope?
• New technique
• Performs simulation of events instead of simulation of time
• Proven to behave much better than RTA based solutions for non-preemptive tasks

Schedule abstraction graph for non-preemptive (Example)

<table>
<thead>
<tr>
<th>Task</th>
<th>Period</th>
<th>Execution time [Min,Max]</th>
</tr>
</thead>
<tbody>
<tr>
<td>τ1</td>
<td>30</td>
<td>[3,13]</td>
</tr>
<tr>
<td>τ2</td>
<td>30</td>
<td>[7,8]</td>
</tr>
<tr>
<td>τ3</td>
<td>10</td>
<td>[1,2]</td>
</tr>
</tbody>
</table>

Task Period Exact execution time τ1 30 [3,13] τ2 30 [7,8] τ3 10 [1,2]
Schedulability analysis of globally scheduled preemptive parallel applications

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>τ₁</td>
<td>[3,13]</td>
<td>Low</td>
</tr>
<tr>
<td>τ₂</td>
<td>[7,8]</td>
<td>Medium</td>
</tr>
<tr>
<td>τ₃</td>
<td>[1,2]</td>
<td>High</td>
</tr>
</tbody>
</table>

Schedule abstraction graph for non-preemptive (Example)
Schedule abstraction graph for non-preemptive (Example)
Why are preemptive tasks difficult to analyze?

**Find worst-case response time (WCRT)**

**Non-preemptive tasks**
- Calculate when the job can start executing
- WCRT = Start time + Execution time

**Preemptive tasks**
- Calculate when the job can start executing
- Keep track of all the preemptions that the job undergoes
- Determine the WCRT
Naïve extension

<table>
<thead>
<tr>
<th>Job</th>
<th>Release Time [Min,Max]</th>
<th>Deadline</th>
<th>Execution Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>[2,2]</td>
<td>6</td>
<td>2</td>
<td>High</td>
</tr>
<tr>
<td>J2</td>
<td>[4,5]</td>
<td>9</td>
<td>2</td>
<td>Medium</td>
</tr>
<tr>
<td>J3</td>
<td>[0,0]</td>
<td>10</td>
<td>4</td>
<td>Low</td>
</tr>
</tbody>
</table>

Schedulability analysis of globally scheduled preemptive parallel applications
Issues with the naïve extension

When J2 releases at 4

When J2 releases at 5
Issues with the naïve extension

Not being able to merge → State space explosion → Build a new solution with changed semantics
Our solution

Preemptive tasks on a **uniprocessor** platform

Extend to **multiprocessor** platform
Our solution - Uniprocessor

A taskset

Find jobs in a observation window

Sort jobs based on earliest release and add jobs onto the graph

Calculate worst-case response time from the graph

State of the system before the job was added

Job being added to the system

State of the system after the job was added

Priority level-i completion interval

Priority of Job $i$: [Earliest finish time, Latest finish time]
Best case scenario

<table>
<thead>
<tr>
<th>Job</th>
<th>Execution Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>[1,2]</td>
<td>High</td>
</tr>
<tr>
<td>J2</td>
<td>[2,3]</td>
<td>Medium</td>
</tr>
<tr>
<td>J3</td>
<td>[4,5]</td>
<td>Low</td>
</tr>
</tbody>
</table>

Worst case scenario
Schedulability analysis of globally scheduled preemptive parallel applications

### Jobs

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Priority</th>
<th>Latest finish time</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>High</td>
<td>&lt;= 10</td>
<td>10</td>
</tr>
<tr>
<td>J2</td>
<td>Medium</td>
<td>&lt;= 9</td>
<td></td>
</tr>
<tr>
<td>J3</td>
<td>Low</td>
<td>&lt;= 10</td>
<td></td>
</tr>
</tbody>
</table>

Taskset → Schedulable ✔️
Analytical view of our solution

1. Add the job with the smallest earliest release
2. Calculate the interference of the newly added job
3. Update intervals for lower priority jobs
4. Remove jobs that are no longer affected
Calculating maximum interference depends on the latest release.

Calculating maximum interference depends on the latest release.

Earliest release

Latest release
Analytical view of our solution

1. Add the job with the smallest earliest release
2. Calculate the interference of the newly added job
3. Update intervals for lower priority jobs
4. Remove jobs that are no longer affected
**Adding a job**

**Scenario 1:**
When latest release of J3 is after the latest finish of all the higher priority jobs

- **J1:** Priority - High
- **J2:** Priority - Medium
- **J3:** Priority - Low

**Newly added job**

Interference(J3) = 0
Adding a job

**Scenario 2:**
When latest release of J3 is before the latest release of all the higher priority jobs

- **J1:** Priority - High
- **J2:** Priority - Medium
- **J3:** Priority - Low
- **Newly added job**

**Interference**

Interference(J3) = Sum of largest execution time of J1 and J2
Adding a job

**Scenario 3:**
In all other scenarios

**J1:** Priority - High

**J2:** Priority - Medium

**J3:** Priority - Low

Newly added job

Interference ($J_3$) = LFT($J_2$) – Latest release($J_3$)
Analytical view of our solution

1. Add the job with the smallest earliest release
2. Calculate the interference of the newly added job
3. Update intervals for lower priority jobs
4. Remove jobs that are no longer affected
Busy period

**Level-i busy period** → Interval of time during which only jobs with priority higher or equal job $J_i$ execute.

**J1**: Priority-High

**J2**: Priority - Low

**Start of level-2 busy period**: SB2

**Level-2 busy period**
Scenario 1: When SB3 > finish time of newly added job

J1: Priority – High
Newly added job

J2: Priority - Medium

J3: Priority – Low
Job to be updated

Interference(J3, J1) = 0
Scenario 2:
When $SB_3 < $ release time of newly added job

$J_1$: Priority – High
Newly added job

$J_2$: Priority - Medium

$J_3$: Priority – Low
Job to be updated

Interference($J_3, J_1$) = Maximum execution time($J_1$)
**Scenario 3:**
In all other cases

\[ \text{Interference}(J_3, J_1) = LFT(J_1) - SB_3 \]
Experiments - Uniprocessor

Tasksets

- Synthetic tasksets were generated using the Emberson and Davis Tool \(^1\)
- Tasksets properties:
  - Ranged from 3 tasks to 18 tasks per taskset
  - Periods were either uniform and loguniform

State of the art

- **Response time analysis** for uniprocessor systems and preemptive tasks.

How does our solution perform in terms of accuracy for a given taskset in comparison to the state of the art?

\(^1\) Techniques For The Synthesis Of Multiprocessor Tasksets, Emberson et al. 2010
State of the art – Uniprocessor

- **Response time analysis - Audsley RTA** [1]
- **Park test** [2]
- **Distance constrained taskset algorithm (DCT)** [3]

**Jitter Offset**

- **Scheduling of hard real-time periodic systems with various kinds of deadline and offset constraints - Goossens** [4]

**Jitter Offset**

- **Calculating exact worst-case response times for static priority scheduled tasks with offsets and jitter - Redell et al.** [5]

[1] Audsley et al., 1995  
[2] Park et al., 2014  
[5] Redell et al., 2002
Evaluation

The higher the schedulability ratio, the better
NO Jitter, NO Offset

NO Jitter but with offset

With both jitter and offset

Our solution overlaps with the exact solution in each case!

Taskset Size: 9 Tasks
Period Properties: Loguniform
Take away from the uniprocessor solution

Our solution performs with high accuracy for preemptive tasks on a uniprocessor system while avoiding state space explosion.

Provides hope to build an accurate analysis for preemptive tasks on a multiprocessor system.
Extension to multiprocessor platform

1. Add the job with the smallest earliest release
2. Calculate the interference of the newly added job
3. Update intervals for lower priority jobs
4. Remove jobs that are no longer affected
Extension to multiprocessor platform

While adding a job – Interference is calculated in the same way as for uniprocessor

While updating a job – Interference is further simplified than in the uniprocessor platform as the busy period concept was not extended

For a multiprocessor platform with ‘m’ cores:

In both cases,

\[ \text{Interference on multiprocessor platform} = \frac{\text{Interference on a single processor}}{\text{Number of cores (m)}} \]
Amount of interference faced: 9 units

Job that faces 9 units of interference

Scenario 1: When the job is scheduled on a single core processor

Processor:

9 units

Scenario 2: When the job is scheduled on a multi core processor

Processor 1:

Processor 2:

Processor 3:

3 units
Experiments - Multiprocessor

Tasksets

- Synthetic tasksets were generated using the Emberson and Davis Tool \cite{1}
- Tasksets properties:
  - Number of cores (m): 2 to 8
  - Number of tasks ranged from \(1\times(m)\) to \(5\times(m)\)
  - Periods were loguniform

State of the art

- Response time analysis (RTA) for multiprocessor systems and preemptive tasks. The paper by Guan et.al \cite{2} was chosen

How does our solution perform in terms of accuracy for a given taskset in comparison to the state of the art?

\cite{1} Techniques For The Synthesis Of Multiprocessor Tasksets, Emberson et al. 2010
Varying utilization

Number of cores = 4
Number of tasks in a taskset = 20

Varying number of tasks

Number of cores = 5
Utilization = 0.25

Varying number of cores

Number of tasks are twice the number of cores
Utilization = 0.35

Our solution performs slightly worse than the state of the art
Our solution performs with **high accuracy** on a **uniprocessor system** while avoiding **state space explosion**

Pessimism is introduced when extending to the multiprocessor case

This method has never been applied to this problem before, and hence more research will help bring down the pessimism
Summary

Thesis timeline:

Solution for uniprocessor → Solution for multiprocessor → Future Work

Thank You!