Schedulability Analysis and Software Synthesis for Graph-based Task Models with Resource Sharing

Jakaria Abdullah, Gaoyang Dai, Morteza Mohaqeqi, Wang Yi

Uppsala University

April 13, 2018
Outline

1. Problem
2. Algorithm
3. Evaluation
4. Synthesis
5. Conclusion
Outline

1 Problem
2 Algorithm
3 Evaluation
4 Synthesis
5 Conclusion
# Graph-based Task Model

## Motivation
- Execute different functions in different contexts
- Model multi-rate execution

## General Features
- Supports different jobtypes
- Uses graph as release pattern

## Applications
- Stateflow blocks of Simulink
- Angle-synchronous task (automotive)
- Frame processing (multimedia)
Digraph Real-Time (DRT) Task Model
(Stigge et al, RTAS 2011)

Features

- Arbitrary directed graph as sporadic release pattern
- Generalizes graph-based task models like Generalized MultiFrame (GMF), Recurring Branching (RB), ...
What about resource sharing?

Resource sharing

- Sharing memory for inter-task communication
- Original DRT model supports
  - Fully preemptive execution - *no resource sharing*
  - Fully non-preemptive execution - *all jobs can share*
What about resource sharing?

Resource sharing

- Sharing memory for inter-task communication
- Original DRT model supports
  - Fully preemptive execution - *no resource sharing*
  - Fully non-preemptive execution - *all jobs can share*

Observation

In a *multi-periodic* system all jobs of a task do not require resource sharing
Communication-by-Sampling

**Oversampling in Communication**

Reuse old data for slower producer

Producer

Consumer

**Undersampling in Communication**

Write data only for potential reader

Producer

Consumer
Our Proposal

Resource sharing DRT

- DRT with Preemptive + Non-preemptive execution
- Resource sharing job types - *non-preemptive execution*
- Other job types - preemptive execution

![Diagram showing resource sharing DRT with job types and their execution times]

\[ J_1 \langle 2, 10 \rangle \quad J_2 \langle 4, 20 \rangle \quad J_3 \langle 1, 20 \rangle \quad J_4 \langle 3, 15 \rangle \]

- J1 with start time 2 and duration 10
- J2 with start time 4 and duration 20
- J3 with start time 1 and duration 20
- J4 with start time 3 and duration 15

- J2 and J3 are connected with a dashed line indicating non-preemptive execution.
Schedulability Analysis

Settings

- DRT tasks with preemptive + non-preemptive jobs
- Fixed task-level unique priority
- Constrained deadline
- Uniprocessor

Existing analysis

1. Fully preemptive execution (Stigge et al. 13)
2. Fully non-preemptive execution (Stigge et al. 15)

Research question

Does mixed execution require new analysis? Let’s look deep
Preemptive Vs. Non-preemptive

**Fully Preemptive Case**

- Lower priority has no effect on higher priority execution
- *Critical instant*: Simultaneous release of all higher priority tasks

![Diagram showing preemptive and non-preemptive execution of tasks](image-url)
Preemptive Vs. Non-preemptive

Fully Non-preemptive Case

- Lower priority can push execution of higher priority
- First job released in critical instant does not give worst-case situation
- Requires checking multiple jobs in continuously busy execution interval known as *busy window (BW)*

\[ \text{H Task} \]

\[ \text{M Task} \]

\[ \text{L Task} \]
Preemptive Vs. Non-preemptive

### Preemptive Job Test

- **Response time**
- **Interference**
- **Start time**
- **Latest start**
- **Interference**

### Non-preemptive Job Test

- **Response time**
- **Interference**
- **Start time**
- **Latest start**

---

Jakaria Abdullah | jakaria.abdullah@it.uu.se
Challenges in our case

Preemptive job in mixed execution

- DRT releases different job types following different paths
- Predecessor jobs can be non-preemptive
- Interfering jobs (other tasks) can be non-preemptive
- BW analysis for non-preemptive job will not work
- Requires new BW based analysis

Busy window

- Exact length of worst-case BW for a job is unknown
- Need to check different intervals to see whether a job under test can be part of a BW
Contributions

Timing Analysis
An *exact test* for fixed priority schedulability analysis of DRT task with preemptive + non-preemptive execution

Software Synthesis
Ada code synthesis of DRT tasks with resource sharing
Outline

1. Problem
2. Algorithm
3. Evaluation
4. Synthesis
5. Conclusion
Algorithm

Main Idea

- **Different BW based analysis** for preemptive and non-preemptive jobs
- Start with an interval for smallest possible BW
- Check whether the job under test can *finish* (preemptive) or *start* (non-preemptive) within it
- Increment the interval to include predecessor jobs until it reaches *upper bound* on BW length
- If a job passes all possible scenario, it is schedulable
- If *all jobs* of a task pass, the task is schedulable
Algorithm Step 1
Preemptive Job

Initialization
Start with scheduling window of job under test $J$ as potential busy window (PBW)
Algorithm Step 2
Preemptive Job

Workload Computation
Compute total workload \( TW = (\text{High priority interference} + \text{Non-preemptive blocking} + \text{task under test}) \) in PBW
Algorithm Step 2
Preemptive Job

Workload Computation
Compute total workload $TW = (\text{High priority interference} + \text{Non-preemptive blocking} + \text{task under test})$ in PBW

Workload Abstraction
Use path based workload abstraction for DRT tasks (Stigge et al. 13)
Request Functions

J₁ \langle 2, 10 \rangle 

J₂ \langle 4, 20 \rangle 

J₃ \langle 1, 20 \rangle 

J₄ \langle 3, 15 \rangle 

rf(t)

rf(J₁,J₂,J₃)

rf(J₃,J₄,J₂)

Departments of Information Technology - 19 - Jakaria Abdullah | jakaria.abdullah@it.uu.se
Workload Abstraction

Over-approximation

\[ rf(t) \]

\[ rf_1 \sqcup rf_2 \]

Abstract \( rf(T)(t) \)

Concrete \( rf(T)(t) \)

Maximum of all \( rf(T)(t) \) for a task \( T \)
Workload Abstraction

Over-approximation

\[ rf(t) \]

\[ rf_1 \sqcup rf_2 \]

Abstract \( rf(T)(t) \)

Concrete \( rf(T)(t) \)

Maximum of all \( rf(T)(t) \) for a task \( T \)
Workload Abstraction

Over-approximation

\[ rf_1 \sqcup rf_2 \]

Abstraction tree

Maximum of all \( rf^{(T)}(t) \) for a task \( T \)

Abstract \( rf^{(T)}(t) \)s

Concrete \( rf^{(T)}(t) \)s

\( rf(t) \)

\( t \)
Algorithm Step 2
Preemptive Job

Workload Computation
Compute total workload $TW = (\text{High priority interference} + \text{Non-preemptive blocking} + \text{task under test})$ in PBW

Maximum Blocking
① One lower priority blocking per BW
② Longest lower priority non-preemptive job maximizes
Algorithm Step 2
Preemptive Job

**Workload Computation**

Compute total workload $TW = (\text{High priority interference} + \text{Non-preemptive blocking} + \text{task under test})$ in PBW

- **Higher priority**
  - Task under test
  - **Lower priority**

Multiple jobs in BW

Include predecessor jobs for BW > scheduling window
Algorithm Step 3
Preemptive Job

Is TW >= Length of PBW?

Yes

Step 4: Workload refinement

No

Step 5: Busy window extension
Algorithm Step 4
Preemptive Job

Workload Refinement

- If TW includes only \textit{concrete workload}, then the job is unschedulable

![Task T_1](image1)

![Task T_3](image2)

- concrete request functions
Algorithm Step 4
Preemptive Job

Workload Refinement

- If TW includes only concrete workload, then the job is unschedulable
- If TW includes any abstract workload then refine it and go back to step 2

abstract request functions: candidates for refinement
Algorithm Step 5
Preemptive Job

Busy Window (BW) extension

Extend PBW *backwards* to include another release of task under test

\[ J_{\text{release} - 1} \quad J_{\text{release}} \quad J_{\text{deadline}} \]

\[ PBW_{i-1} \]
Algorithm Step 5
Preemptive Job

Busy Window (BW) extension

Extend PBW \textit{backwards} to include another release of task under test

\[ J_{\text{release} - 1} \quad J_{\text{release}} \quad J_{\text{deadline}} \]

\[ \text{extension} \]

\[ \text{PBW}_i \]

\[ t \]
Algorithm Step 5

Preemptive Job

Busy Window (BW) extension

- If new PBW < MBW, go to step 2, otherwise feasible
- MBW is the smallest $t$ where $\sum_{T \in \tau} mrf^{(T)}(t) \leq t$
**Algorithm**

**Non-preemptive Job**

**Differences**

- Start time computation excludes job under test
- High priority jobs cannot preempt non-preemptive execution

**Change 1**

The initial $PBW = Scheduling \text{ window} - WCET$ of Job under test

\[ J_{release} \quad PBW \quad J_{deadline} \]

\[ WCET_J \]
**Algorithm**

**Non-preemptive Job**

**Differences**

- Start time computation excludes job under test
- High priority jobs can not preempt non-preemptive execution

**Change 2**

High priority interference (request functions) should include jobs released at the end of an interval

![Graph](image)
Algorithm Summary

Generalization

Exactness

Our test is exact for *sporadic* release as

- Pass means taskset is schedulable
- Fail generates a set of job releases that is not schedulable

Our algorithm

- Preemptive only
  - Fully preemptive test
- Non-preemptive only
  - Fully non-preemptive test
Evaluation
Experimental Settings

Random tasksets

- 200 DRT tasksets/3% utilization
- Maximum 25 tasks/taskset
- Randomly assigned *unique priority* to each task
- Randomly marked *jobtypes* as non-preemptive

Realistic workload

- A fully non-preemptive angle-synchronous DRT task

```
\begin{align*}
&\langle 5, 22 \rangle \\
&\langle 3, 13 \rangle \\
&\langle 1, 9 \rangle \\
\end{align*}
```

- Bosch case study (2015) + Mohaqeqi et al. 17
Evaluation
Acceptance Vs. Utilization

Table: Random Task set parameters

<table>
<thead>
<tr>
<th>Job types</th>
<th>Branching degree</th>
<th>p</th>
<th>d/p</th>
<th>e/d</th>
</tr>
</thead>
<tbody>
<tr>
<td>[3, 5]</td>
<td>[1, 3]</td>
<td>[50, 200]</td>
<td>[0.5, 1]</td>
<td>[0.01, 0.05]</td>
</tr>
</tbody>
</table>
Evaluation
Runtime Vs. Utilization

Table: Random Task set parameters

<table>
<thead>
<tr>
<th>Job types</th>
<th>Branching degree</th>
<th>p</th>
<th>d/p</th>
<th>e/d</th>
</tr>
</thead>
<tbody>
<tr>
<td>[3, 5]</td>
<td>[1, 3]</td>
<td>[50, 200]</td>
<td>[0.5, 1]</td>
<td>[0.01, 0.05]</td>
</tr>
</tbody>
</table>
Evaluation

Observations

On acceptance
- Ratio of non-preemptive jobtypes has little effect
- Priority assignment influences acceptance in higher utilization
- Taskset schedulable in mixed execution may not be schedulable in fully preemptive or fully non-preemptive settings

On runtime
- Testing time for schedulable tasksets < 10 seconds
- Depends on length of MBW
- Complexity: Strongly co-NP hard (from fully preemptive case)
Outline

1. Problem
2. Algorithm
3. Evaluation
4. Synthesis
5. Conclusion
Software Synthesis

How to generate code for resource sharing DRT?

Two key features need implementation:
1. Event-triggered (sporadic) release of different jobtypes
2. Mixed preemptive and non-preemptive execution

Our Approach
We use Ada programming language and its runtime for generating DRT code
Software Synthesis

Event-triggered release

- Use Ada Protected Object (PO) to release based on event
- Interrupt handlers are attached with POs
- Jobs can block on a PO entry

Example

```plaintext
-- Event receiver is a protected object
-- Task blocked here for next release
Event_receiver.Wait(event_id);
if event_id = u then
    -- call jobtype for u
else if event_id = v then
    -- call jobtype for v
```
Software Synthesis

Preemptive + Non-preemptive execution

- Assign one global PO maximum system priority
- Put all non-preemptive job procedures in the global PO
- PO uses Immediate Ceiling Priority Protocol (ICPP)

Example

```plaintext
-- Entry for blocking tasks
entry Wait (Event: event_type ID);  
-- Highest System priority
pragma Interrupt_Priority (Priority_Max);  
-- Declaration for non-preemptive procedures
procedure NPR_Job1_task_a;  
procedure NPR_Job2_task_a;  
procedure NPR_Job1_task_b;  
...  
```
Summary

- Exact fixed priority schedulability test for DRT with job-level non-preemptive resource sharing
- Quick analysis for schedulable taskset
- Software synthesis using Ada without runtime modification

Future Work

- Analysis for co-operative scheduling
- Apply classical resource sharing protocols
- Compute end-to-end latency
- Abstraction refinement on busy window extension
Questions?
References