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

UPPSALA UNIVERSITET

Outline

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

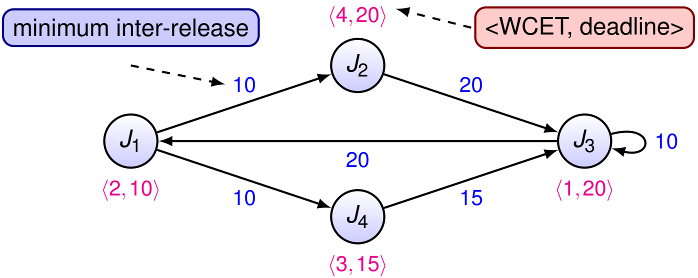
- Stateflow blocks of Simulink
- Angle-synchronous task (automotive)
- Frame processing (multimedia)

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), . . .



UPPSALA UNIVERSITET

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*

Jakaria Abdullah | *jakaria.abdullah@it.uu.se*

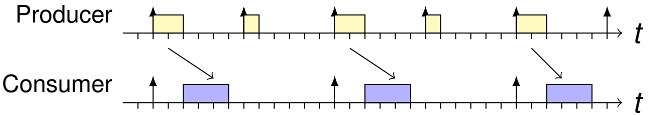
What about resource sharing?

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

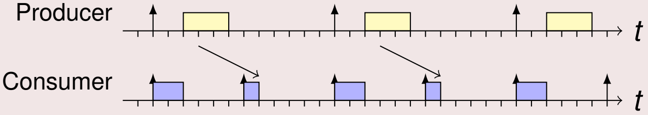
In a *multi-periodic* system all jobs of a task do not require resource sharing



Communication-by-Sampling

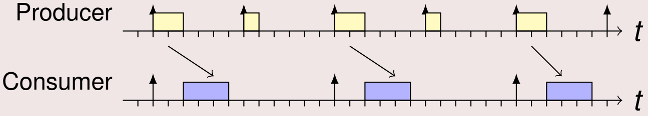
Oversampling in Communication

Reuse old data for slower producer



Undersampling in Communication

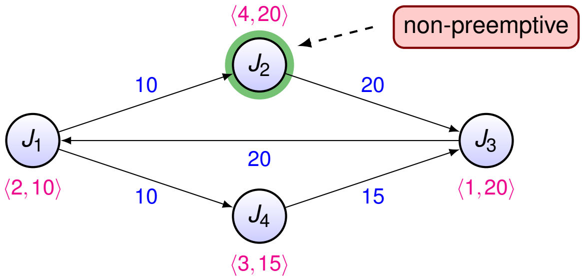
Write data only for potential reader



Our Proposal

Resource sharing DRT

- DRT with Preemptive + Non-preemptive execution
- Resource sharing jobtypes - *non-preemptive execution*
- Other jobtypes - 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)

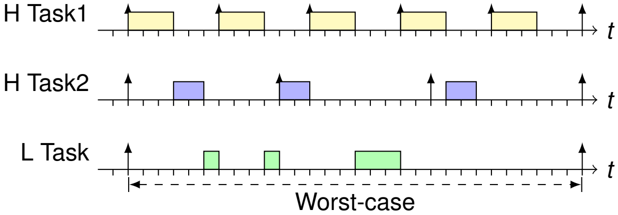
Research question

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

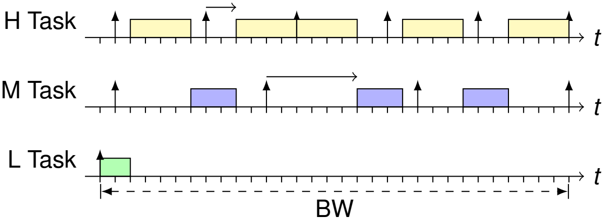


UPPSALA UNIVERSITET

Preemptive Vs. Non-preemptive

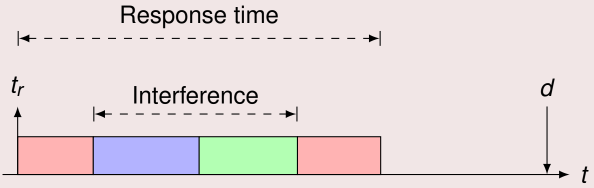
Fully Non-preemptive Case

- 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)*

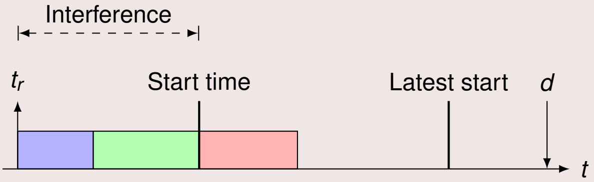


Preemptive Vs. Non-preemptive

Preemptive Job Test



Non-preemptive Job Test



UPPSALA UNIVERSITET

Problem

Algorithm

Evaluation

Synthesis

Conclusion

Challenges in our case

Preemptive job in mixed execution

- DRT releases different jobtypes 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

Algorithm

Main Idea

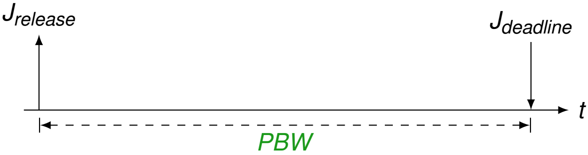
- *Different BW based analysis* for preemptive and non-preemptive jobs
- Start with an interval for smallest possible BW
- Check whether the job under test can *finish* (preemptive) or *start* (non-preemptive) within it
- Increment the interval to include predecessor jobs until it reaches *upper bound* 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 = (High priority interference + Non-preemptive blocking + task under test) in PBW

Algorithm Step 2
Preemptive Job

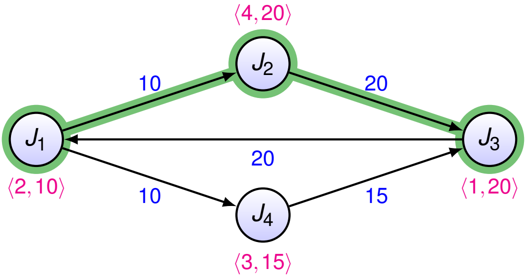
Workload Computation

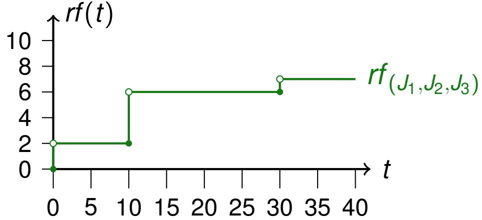
Compute total workload TW = (High priority interference + Non-preemptive blocking + task under test) in PBW

Workload Abstraction

Use path based workload abstraction for DRT tasks (Stigge et al. 13)

Request Functions

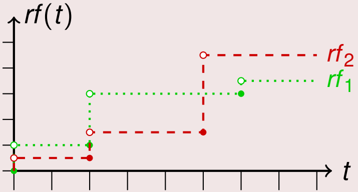




UPPSALA UNIVERSITET

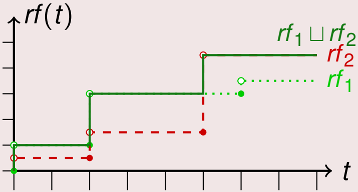
Workload Abstraction

Over-approximation



Workload Abstraction

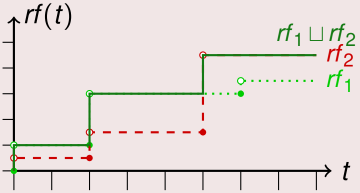
Over-approximation



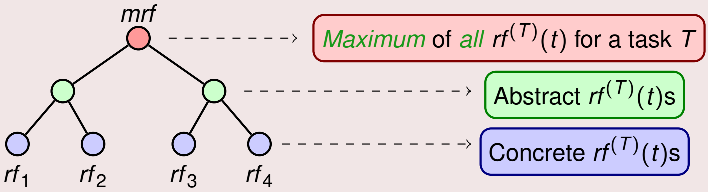
UPPSALA UNIVERSITET

Workload Abstraction

Over-approximation



Abstraction tree

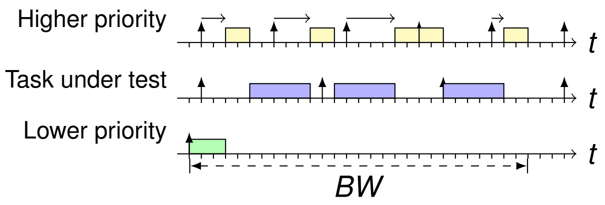


Algorithm Step 2

Preemptive Job

Workload Computation

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



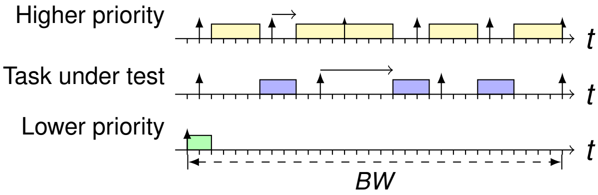
Maximum Blocking

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 = (High priority interference + Non-preemptive blocking + task under test) in PBW

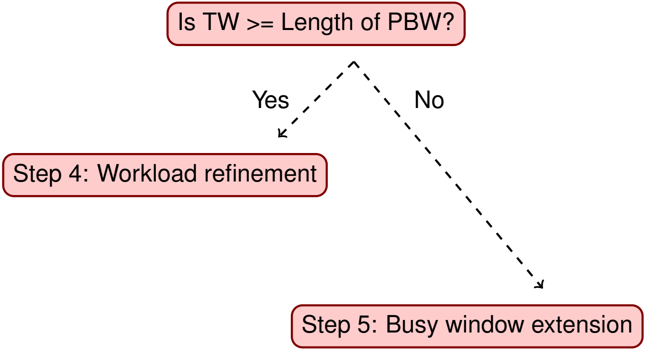


Multiple jobs in BW

Include predecessor jobs for BW > scheduling window

UPPSALA UNIVERSITET

Algorithm Step 3
Preemptive Job

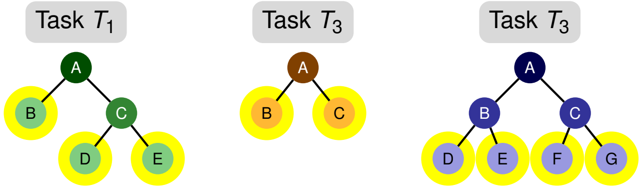


Problem
Algorithm
Evaluation
Synthesis
Conclusion

Algorithm Step 4
Preemptive Job

Workload Refinement

- If TW includes only *concrete workload*, then the job is unschedulable



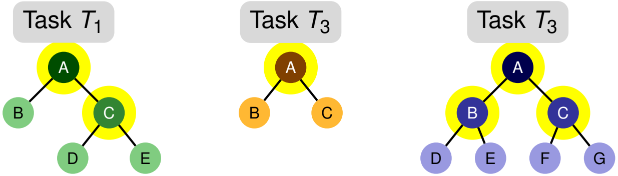
concrete request functions

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



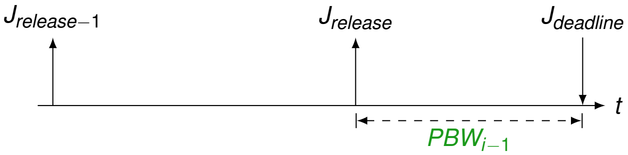
abstract request functions: candidates for refinement

UPPSALA UNIVERSITET

Algorithm Step 5
Preemptive Job

Busy Window (BW) extension

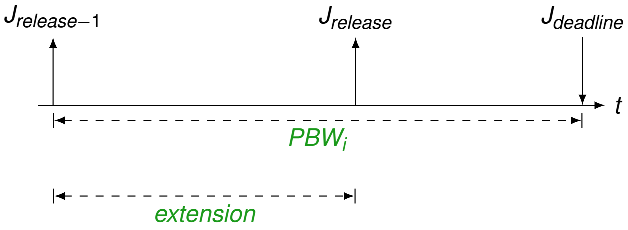
Extend PBW *backwards* to include another release of task under test



Algorithm Step 5
Preemptive Job

Busy Window (BW) extension

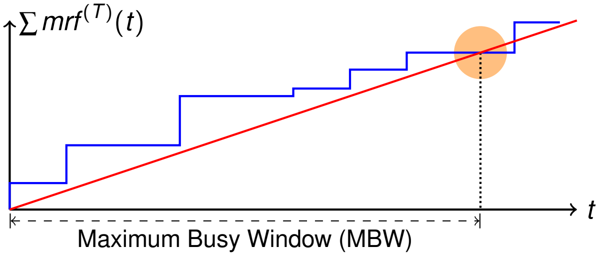
Extend PBW *backwards* to include another release of task under test



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 $\sum_{T \in \tau} mrf^{(T)}(t) \leqslant t$



Algorithm

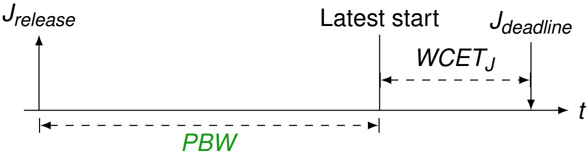
Non-preemptive Job

Differences

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

Change 1

The initial PBW = Scheduling window - WCET of Job under test



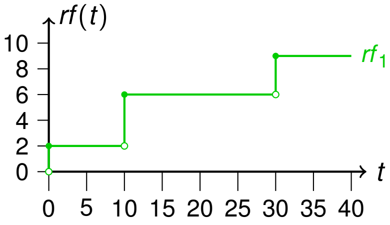
Algorithm
Non-preemptive Job

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



UPPSALA UNIVERSITET

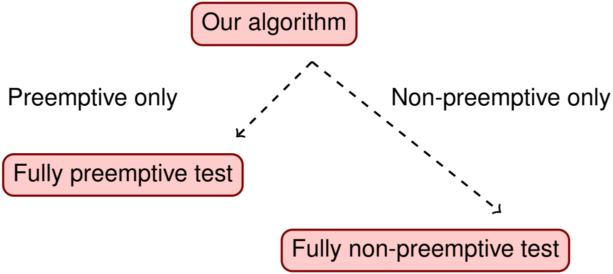
Algorithm Summary

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



UPPSALA UNIVERSITET

Outline

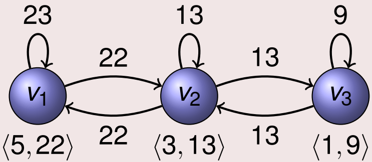
Evaluation
Experimental Settings

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



- Bosch case study (2015) + Mohaqeqi et al. 17

UPPSALA UNIVERSITET

Evaluation

Acceptance Vs. Utilization

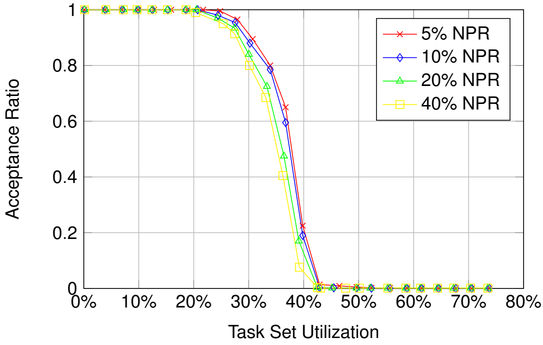


Table: Random Task set parameters

Job types	Branching degree	p	d/p	e/d
$[3,5]$	$[1,3]$	$[50,200]$	$[0.5,1]$	$[0.01,0.05]$

Evaluation
Runtime Vs. Utilization

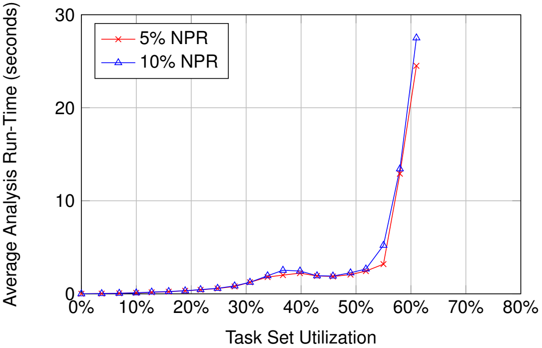


Table: Random Task set parameters

Job types	Branching degree	p	d/p	e/d
$[3,5]$	$[1,3]$	$[50,200]$	$[0.5,1]$	$[0.01,0.05]$

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)

UPPSALA UNIVERSITET

Outline

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

UPPSALA UNIVERSITET

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

```
-- 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

```
-- 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;
...
```

UPPSALA UNIVERSITET

Outline

Summary and Future Work

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

UPPSALA UNIVERSITET

Problem
Algorithm
Evaluation
Synthesis
Conclusion

Questions?

UPPSALA UNIVERSITET

- Problem
- Algorithm
- Evaluation
- Synthesis
- **Conclusion**

References

- M. Stigge, P. Ekberg, N. Guan, and W. Yi, The digraph real-time task model, RTAS, 2011, pp. 71–80.
- M. Stigge and W. Yi, Combinatorial abstraction refinement for feasibility analysis, RTSS, 2013, pp. 340–349.
- M. Stigge and W. Yi, Combinatorial abstraction refinement for feasibility analysis of static priorities, Real-Time Systems, vol. 51, no. 6, pp. 639–674, 2015.
- M. Mohaqeqi, J. Abdullah, P. Ekberg, and W. Yi, Refinement of Workload Models for Engine Controllers by State Space Partitioning, ECRTS, 2017, pp. 11:1–11:22.
- A. Hamann, D. Dasari, S. Kramer, M. Pressler, F. Wurst, and D. Ziegenbein, Waters industrial challenge 2017.