Towards an Efficient and Accurate Schedulability Analysis for Real-Time Cyber-Physical Systems

Mitra Nasri

Assistant professor
Delft University of Technology
Embracing future challenges

**Back then**
- A few computing nodes and control loops
- Simple hardware and software architecture
- Liu and Layland task model was a relevant thing for those systems

**Now (and future)**
- Complex (and usually parallelizable) application workloads running on heterogeneous multi- and many-core platforms
- Intensive I/O accesses
- Use of hardware accelerators (GPUs, FPGA, DSP, co-processors, etc.)
- Computation offloading (to the cloud, edge, etc.)

**Simple, predictable, and easier-to-analyze computing models**

**Complex, less predictable, and harder-to-analyze computing models**
A wish list

Obtain the **worst-case** and **best-case response time**

**Parallel heterogeneous DAG**
**tasks with conditional branches**

- Each resource may have its own scheduling policy
- Schedulers may have different runtime overheads

**Task**

- Arrivial model
- Execution model

**Occupation time of a resource**

- Bounded or unbounded uncertainty
- Deadline
- Time
- Precedence

- Machine 1
- Machine 2
- Network

**Resource affinity**
State of the art

Closed-form analyses (e.g., problem-window analysis)

- Fast
- Pessimistic
- Hard to extend

Experiment:
10 limited-preemptive parallel DAG tasks scheduled by global FP on 16 cores


State of the art

Closed-form analyses (e.g., problem-window analysis)
- Fast
- Pessimistic
- Hard to extend

Exact tests in generic formal verification tools (e.g., UPPAAL)
- Accurate
- Easy to extend
- Not scalable

The “tool” does all the labor (to find the worst case)

Generic verification tools are very slow and do not scale to reasonable problem sizes
Results from formal verification-based analyses

Sequential non-preemptive periodic tasks (scheduled by global FP)

4 cores, 30% utilization

Number of tasks vs. schedulability ratio:
- Exact test (timeout)
- Exact test (UPPAAL)

Runtime (sec) vs. number of tasks:
- 4 cores
- 2 cores
- 1 core
- 8 cores

[Reference: Nasri’19]

(8 cores)
State of the art

Closed-form analyses
(e.g., problem-window analysis)
- Fast
- Pessimistic
- Hard to extend

Exact tests in generic formal verification tools (e.g., UPPAAL)
- Accurate
- Easy to extend
- Not scalable

Our new line of work

Idea: efficiently explore the space of all possible schedules

Response-time analysis using schedule abstraction
- Applicable to complex problems
- Easy to extend
- Highly accurate
- Relatively fast
Some results on parallel DAG tasks

- [ECRTS’19] (m=4) vs Serrano (m=4)
- [ECRTS’19] (m=8) vs Serrano (m=8)
- [ECRTS’19] (m=16) vs Serrano (m=16)

Response-time analysis using schedule abstraction
An example: the problem of global non-preemptive scheduling

Obtain the **worst-case** and **best-case response time**

<table>
<thead>
<tr>
<th>Workload model</th>
<th>Platform model</th>
<th>Scheduler model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Non-preemptive</strong> job sets</td>
<td><strong>Multicore</strong> (identical cores)</td>
<td><strong>Global job-level fixed-priority</strong> (JLFP)</td>
</tr>
</tbody>
</table>

### Job model

- **Release jitter**
- **Deadline**
- **Execution time variation**
- **Deadline**

The job set is provided for an **observation window**, e.g., a hyperperiod.

This job model supports bounded non-deterministic arrivals, but **not** sporadic tasks (un-bounded non-deterministic arrivals)
Solution highlights

A sound analysis must consider all possible execution scenarios (i.e., combination of release times and execution times)

Observation

There are fewer permissible job orderings than schedules

Solution

Use job-ordering abstraction to analyze schedulability by building a graph that represents all possible schedules
Response-time analysis using schedule-abstraction graphs

A path aggregates all schedules with the same job ordering

A path represents a set of similar schedules

Different paths have different job orders
Response-time analysis using schedule-abstraction graphs

A path aggregates all schedules with the same job ordering.

A vertex abstracts a system state and an edge represents a dispatched job.

A vertex abstracts a system state and an edge represents a dispatched job.

Earliest and latest finish time of $J_1$ when it is dispatched after state $v$.

$J_1: [4, 8]$
Response-time analysis using schedule-abstraction graphs

A path aggregates all schedules with the same job ordering.

A vertex abstracts a system state and an edge represents a dispatched job.

A state is labeled with the finish-time interval of any path reaching the state.

A system state

Interpretation of an uncertainty interval:
- Certainly not available
- Possibly available
- Certainly available

Core 1: [10, 30]
Core 2: [15, 20]
Response-time analysis using schedule-abstraction graphs

A path aggregates all schedules with the same job ordering

A vertex abstracts a system state and an edge represents a dispatched job

A state represents the finish-time interval of any path reaching that state

Obtaining the response time:

Best-case response time = min \{ completion times of the job \} = 2
Worst-case response time = max \{ completion times of the job \} = 15
Building the schedule-abstraction graph

Building the graph (a breadth-first method)

Repeat until every path includes all jobs
1. Find the shortest path
2. For each not-yet-dispatched job that can be dispatched after the path:
   2.1. Expand (add a new vertex)
   2.2. Merge (if possible, merge the new vertex with an existing vertex)

System is idle and no job has been scheduled
Building the schedule-abstraction graph

Expanding a vertex:
(reasoning on uncertainty intervals)

Expansion rules imply the scheduling policy

State $v_i$

Available jobs
(at the state)

Next states

High priority

Medium priority

Low priority
Define the state abstraction

Define the expansion rules

Define merging rules

How to use schedule-abstraction graphs to solve a new problem?

What is encoded by an edge? What is encoded by a state?

How to create new states?

How to identify similar states?

And then, prove soundness

“the expansion rules must cover all possible schedules of the job set”
Challenges

https://www.globallanguageservices.co.uk/30-days-of-language-challenges/
Challenge: handling release jitter

No jitter

\[ J_1 \]

High priority \( C_1 \in [10, 15] \)

\[ J_2 \]

Medium priority \( C_2 \in [15, 20] \)

\[ J_3 \]

Low priority \( C_3 \in [12, 16] \)
Challenge: handling release jitter

Small jitter

\[ J_1 \]: 
\[ J_2 \]: 
\[ J_3 \]:

High priority \( C_1 \in [10, 15] \)

Medium priority \( C_2 \in [15, 20] \)

Low priority \( C_3 \in [12, 16] \)

uniprocessor

\[ J_1: [10, 19]\]
\[ J_2: [15, 24]\]
\[ J_3: [12, 20]\]

\[ J_1: [25, 39]\]
\[ J_2: [25, 39]\]
\[ J_3: [37, 55]\]

\[ J_1: [22, 35]\]
\[ J_2: [37, 55]\]
\[ J_3: [37, 55]\]
Challenge: handling release jitter

Large release jitter (or sporadic release) may result in a combinatorial state space

Larger jitter

$J_1$

0 35 60
High priority $C_1 \in [10, 15]$

$J_2$

0 30 54
Medium priority $C_2 \in [15, 20]$

$J_3$

0 40 60
Low priority $C_3 \in [12, 16]$

The maximum number of branches follows the binomial co-efficient
Challenge: handling release jitter

Large release jitter (or sporadic release) may result in a combinatorial state space

Potential solutions

Partial-order reduction

Avoid exploring paths that do not contribute to the worst-case scenario.

Use approximation to derive the worst-case completion time of the remaining jobs in that path
Challenges: handling release jitter
Large release jitter may result in a combinatorial state space

Potential solutions

Partial-order reduction

Batch processing (rather than processing a single job at a time)

Derive expansion rules for a set of jobs

Priority

\( J_4 \)  Highest

\( J_1 \)  High

\( J_2 \)  Medium

\( J_3 \)  Low
Challenges: handling release jitter

Large release jitter may result in a combinatorial state space

Potential solutions

- Partial-order reduction
- Batch processing
- Using memorization (to avoid exploring previously seen patterns)

What else?

Thank you.