Schedule Abstraction Graph

Department of Electrical Engineering

Electronic Systems (ES) Group
Real-time systems

Systems that require both functional correctness and **temporal correctness**
Real-time systems

Systems that require both functional correctness and **temporal correctness**

**Safety**
- Human life
- Environment

**Time-predictability**
- A late (or missed) actuation may cause safety violation
- Example: breaking, air-bag inflation, etc.

**Correct response**
**Functional correctness** + **Temporal correctness**

- Fast ≠ predictable
Why guaranteeing temporal correctness is hard?

Back then

- A few computing nodes and control loops
- Simple hardware and software architecture

Simple, predictable, and easier-to-analyze

Now (and future)

- Complex software (application workloads) running on heterogeneous computing environment
- Intensive I/O accesses
- Use of hardware accelerators (GPUs, FPGA, co-processors, etc.)
- Computation offloading (to the cloud, edge, etc.)
- Heterogeneous communication

Complex, less predictable, and harder-to-analyze
State of the art

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

- Fast
- Pessimistic
- Hard to extend

Exact analyses in generic formal verification tools (e.g., UPPAAL)

- Accurate
- Easy to extend
- Not scalable

\[
R^{(0)}_i = C_i + \sum_{j=1}^{i-1} C_j \\
R^{(k)}_i = C_i + \sum_{j=1}^{i-1} \left( \frac{R^{(k-1)}_j}{T_j} \right) C_j
\]

A response-time analysis of a DAG-based task-set with a limited-preemptive global fixed priority scheduler is computed by iterating the following equation until a fixed point is reached, starting with \( R_k = \text{len}(G_k) + \frac{1}{m} (\text{vol}(G_k) - \text{len}(G_k)) \): \( R_k \leftarrow \text{len}(G_k) + \frac{1}{m} (\text{vol}(G_k) - \text{len}(G_k)) + I^{hp}_k + I^{lp}_k \)
State of the art

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

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

My research
- Applicable to complex problems
- Easy to extend
- Highly accurate
- Relatively fast

Industrial use cases are typically large, complex, and require accurate analysis.
My research in a nutshell

Workload model
(timing features of the workload)

Resource model

Scheduling algorithm
(dynamic and static scheduling algorithms)

Timing-analysis framework
(it is a formal verification engine dedicated to timing models and timing properties)

Partial-order reduction

Merging

Pruning

Worst-case Performance Bounds
(for para-functional properties)

• Lower and upper bounds on
  • response time, latency, throughput, jitter, quality of service/control
  • Counter examples in case of timing violation

3000x faster than generic timing verification tools (e.g., UPPAAL)

Many top-rank conference papers
[RTSS’17, ECRTS’18, ECRTS’19, DATE’19, RTSS’20, RTSS’21, RTAS’22]
The response-time analysis problem

**Workload model**

- **Job**
  - Arrival model
  - Execution model

**Resource model**

- How many resources of which type (and with what access costs) are available?

**Scheduler model**

- How are the resources governed (scheduled)?

**Response-time analysis problem**

*Given a set of jobs, resources, and scheduling policies,*
*Determine the worst-case response time of each job*
Why the problem is hard?

One of the simplest forms of the problem:

Response-time analysis problem

Given
- a set of non-preemptible jobs (with a given arrival interval, execution time, and deadline)
- scheduled by a fixed-priority scheduling policy
- on a uniprocessor platform,

Determine
the worst-case response time of each job

The preemptive version of this problem (which is believed to be easier than the non-preemptive one), is already NP-Complete.
Why the problem is hard?

One of the simplest forms of the problem:

**Response-time analysis problem**

**Given**
- a set of non-preemptible jobs (with a given arrival interval, execution time, and deadline)
- scheduled by a fixed-priority scheduling policy
- on a uniprocessor platform,

**Determine**
the worst-case response time of each job
Why the problem is hard?

**Execution scenario 1:** jobs are released very late and have their largest execution time.

<table>
<thead>
<tr>
<th>Job</th>
<th>Release time</th>
<th>Deadline</th>
<th>Execution time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>0 0 10</td>
<td>10</td>
<td>1 2</td>
<td>high</td>
</tr>
<tr>
<td>$J_2$</td>
<td>0 0 30</td>
<td>7 8</td>
<td>medium</td>
<td></td>
</tr>
<tr>
<td>$J_3$</td>
<td>0 15 30</td>
<td>3 13</td>
<td>low</td>
<td></td>
</tr>
<tr>
<td>$J_4$</td>
<td>10 10 20</td>
<td>1 2</td>
<td>high</td>
<td></td>
</tr>
</tbody>
</table>

**Execution scenario 2:** jobs are released very early and have their largest execution time except for $J_1$.

<table>
<thead>
<tr>
<th>Job</th>
<th>Release time</th>
<th>Deadline</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>1 9 10</td>
<td>24 26</td>
<td>Deadline miss for $J_4$</td>
</tr>
<tr>
<td>$J_2$</td>
<td>0 1 9</td>
<td>13 24</td>
<td></td>
</tr>
<tr>
<td>$J_3$</td>
<td>0 9 13</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>$J_4$</td>
<td>10 10 20</td>
<td>24 26</td>
<td></td>
</tr>
</tbody>
</table>
Why the problem is hard?

Naively enumerating all possible combinations of release times and execution times (a.k.a. execution scenarios) is not practical.

Observation:
There are fewer permissible job orderings than schedules.

Example for path 1

\[ \begin{align*}
J_1 & \quad & 2 \\
J_2 & \quad & 10 \\
J_3 & \quad & 10 \\
J_4 & \quad & 21 \\
\end{align*} \]

Example for path 2

\[ \begin{align*}
J_1 & \quad & 10 \\
J_2 & \quad & 9 \\
J_3 & \quad & \text{miss} \\
J_4 & \quad & 26 \\
\end{align*} \]

- 2 possible job ordering
- 1200 different combinations for release times and execution times
Why the problem is hard?

Naively enumerating all possible combinations of release times and execution times (a.k.a. execution scenarios) is not practical.

Observation:

There are fewer permissible job orderings than schedules.

Solution idea:

We use job-ordering abstraction to build a graph that abstracts all possible schedules.

It is called the “schedule-abstraction graph”.

Goal: an accurate and efficient analysis.
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 path aggregates all schedules with the same job ordering

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

- Start
- Core 1: 10 to 30
- Core 2: 15 to 20
- End

Interpretation of an uncertainty interval:
- Certainly not available
- Possibly available
- Certainly available
Response-time analysis using schedule-abstraction graphs

Best-case response time = \( \min \{\text{completion times of the job}\} = 2 \)
Worst-case response time = \( \max \{\text{completion times of the job}\} = 15 \)
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

[ECRTS’2018]
Building the schedule-abstraction graph

**Building the graph**
(a breadth-first method)

System is idle and no job has been scheduled

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)

Initial state

Diagram showing the process of building the schedule-abstraction graph with nodes and edges indicating the merging and expansion steps.
Comparison with an exact schedulability test implemented in UPPAAL, a generic model-verification tool.

Schedulability = ratio of systems deemed schedulable to the total evaluated systems

Mitra Nasri – Research Directions
Taste of results

Comparison with an exact schedulability test implemented in UPPAAL, a generic model-verification tool.

Almost as accurate as the exact test.

Schedulability = ratio of systems deemed schedulable to the total evaluated systems.

[ECRTS’2019]
Taste of results

Comparison against closed-form analysis

Experiments:
- A multiprocessor platform with 8 identical cores
- Each data point summarizes the results for 200 randomly generated task sets
- Each task set includes 10 parallel real-time tasks with periodic activations that have a directed-acyclic graph (DAG) data-flow dependency

Schedulability = ratio of systems deemed schedulable to the total evaluated systems
Future of schedule-abstraction graph
Extensions of schedule-abstraction graph

Supporting preemptive execution

Analyzing heterogeneous computing platforms

Analyzing conditional workload models

Using machine learning to boost state-space exploration

Deriving problem-specific partial-order reduction rules to prone state-space

Analyzing AI-based scheduling algorithms

System optimization (finding optimal configurations)

Root-cause analysis (what caused a certain outcome?)

Improving practicability (easier to be used in projects)

Improving applicability

Improving scalability

Supporting emerging technologies/research

Deriving workload’s and platform’s timing model

Supporting highly-dynamic workloads (non-deterministic arrival, event-chains, …)
My main research questions

How to **analyze** the **timing behavior** of a decision system in which decisions are made by **machine learning**?

Possible system’s decisions (actions) at state $v$

New system state after action $A$

resource 1: 10    30
recourse 2: 15    20

Certainly not available
Possibly available
Certainly available

Uncertainty interval:
My main research questions

How to analyze the worst-case performance of a decision-making system that uses AI?

How to design a dependable (trustable) decision-making system that uses AI?

How?

Timing-analysis framework (it is a formal verification engine dedicated to timing models and timing properties)

Extend to Analyze the worst-case timing behavior of a system scheduled by a smart scheduler

Are we ready to trust (and depend on) AI for safety-critical systems?

Why?
My current project involvements

- How to **analyze** the timing behavior of a given system?
- How to **design** and **configure** a system to have better timing predictability?
- How to **manage resources at runtime**, and **adapt** to **dynamic changes** such that the system **continuously delivers correct and timely** service?
- How to **analyze** the **worst-case performance** of a **decision-making system** that uses **AI**?
- How to design a **dependable** (trustable) **decision-making system** that uses **AI**?

**TRANSACT**
H2020 project (1.4M€, 3 PhD students)
June 2021 to 2024

**SAM-FMS**
NWO project (600K€, 2 PhD students)
Sep. 2020 to 2024

**My startup package**
2 PhD positions

**Working on a proposal**

Would like to collaborate?
Please contact me
My network

The Netherlands

Benny Akesson (TNO-ESI, UvA)
Marc Geilen (TU/e)
Jeroen Voeten (TU/e)
Geoffrey Nelissen (TU/e)

Twan Basten (TU/e)
Sander Stuijk (TU/e)
Kees Goossens (TU/e)

Europe and USA (Germany, Sweden, Italy, Portugal, Austria, and UK)

Björn Brandenburg (MPI)
Karl-Erik Arzen (Lund)
Anton Cervin (CISTER)
Eduardo Tovar (KTH)
David Broman (KTH)
Robert Davis (York)
Arne Hamann (Bosch)

Marko Bertogna (UniMore)
Gerhard Fohler (TU-KL)
Ramon Serena (TTTech)
Sanjoy Baruah (WashU)
Tam Chantem (VTech)
Ryan Gerdes (VTech)
Gedare Bloom (UCCS)