Workload Models for System-Level Timing Analysis: Expressiveness vs. Analysis Efficiency

Nan Guan, Martin Stigge and Wang Yi
Uppsala University, Sweden
Uppsala University, China
Complex Real-Time Systems

Input Stream

BUS

I/O

DSP

I/O

ECU

FPGA

I/O

Input Stream
Complex Real-Time Systems
Timing Analysis

• What’s the maximal delay at each component?
• What’s the maximal end-to-end delay?

... ...
Timing Analysis

- **Task-Level Timing Analysis**
  - Find out the resource requirement for each task
  - WCET estimation
- **System-Level Timing Analysis**
  - According to the WCET, budgeting the “CPU resource” for each task
  - Schedulability Analysis (Response Time Analysis)
Workload models

- **“Operational” Models**: release patterns/executions
  - Periodic/sporadic models [Liu & Layland, 1973]
  - Tree/Graph-based models [Baruah, Stigge et al. RTSS 2013]

- **“Denotational” Models**: resource requirement over time
  - Demand (Request) Bound Functions (DBF) [Baruah et al, 1990s]
  - Real-Time Calculus (with origin from Network calculus) [Thiele/Samarjit 2000 … Guan et al RTSS 2013]
Expressiveness vs. Analysis Efficiency

Feasibility test

difficult -> efficient

Expressiveness

low -> high

Strongly (co)NP-hard
Pseudo-Polynomial
Expressiveness vs. Analysis Efficiency vs. Analysis Precision

- Expressiveness:
  - High
  - Low

- Analysis Efficiency:
  - Difficult
  - Efficient

- Analysis Precision:
  - Approximate
  - Exact

Feasibility test:
- Strongly (co)NP-hard
- Pseudo-Polynomial
OUTLINE

• An Overview on RT workload models
  – from the simplest to the most complex
  – from Liu & Layland classic periodic model to timed automata

• Recent Results
  – the graph-based model [RTAS 2011, RTSS 2013, Stigge & Yi]
    • Combinatorial Abstraction for Schedulability Analysis
Modeling a system for analysis

- System = a set of Tasks


task 1

Task 2

Task n

- Tasks releasing jobs, \( J = (e, d) \) -- basic unit of workload
  - Release time \( r \)
  - Worst-case execution time \( e \)
  - Deadline \( d \)

- Job release patterns:
  - periodicity, branching structures, loop ... ...

Feasibility/Schedulability: Can we check s.t. all jobs meet their deadlines?
Typical task code structure:

```plaintext
... 
loop 

    // Execute some function for, e.g., 
    // up to 11ms 
    // (obtained via WCET analysis)

    delay until Previous_Period + 50ms; 
end loop; 
...
```

**Periodic Task**

- Execution time $e = 11ms$ of each job
- Period $p = 50ms$
- Implicit deadline $d = p$ for each job
The Liu and Layland Task Model

- Each \textit{periodic} task defined via 2 numbers:
  - Job WCET $e$
  - Minimum inter-release delay $p$ (implicit deadline)

- Task set $\tau = \{ \tau_1, \ldots, \tau_n \}$ with $\tau_i = (e_i, p_i)$

- How to \textit{schedule} these tasks?
  - Static/fixed priority scheduling (e.g. RM)
  - Dynamic priority scheduling (e.g. EDF)

- How to \textit{analyze} schedulability?
The Liu and Layland Task Model: Schedulability

- **Static priority scheduling**
  - Response time analysis for all tasks
    \[
    R_i = e_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{p_j} \right\rceil \cdot e_j
    \]
  - Compute iteratively; compare with \( p_i \)
  - **Precise** test (sufficient and necessary)
  - **Utilization** test
    - Define \( U_i := e_i/p_i \) as utilization of task \( \tau_i \)
    - \( \tau \) schedulable if \( \sum_i U_i \leq n(2^{1/n} - 1) \)
    - Only sufficient test

- **Dynamic priority scheduling**
  - Just focus on EDF, it’s optimal
  - Simple and precise test: \( \sum_i U_i \leq 1 \)
Hierarchy of Models

- Feasibility test
  - difficult
  - efficient

- Expressiveness
  - high
  - low

- Strongly (co)NP-hard
- Pseudo-Polynomial

L&L
Hierarchy of Models (upto RTSS 2011)
The Recurring Branching (RB) Task Model (cont.)

- Introduces *branching* structures
- A *tree* for each task
  - Vertices $v$: job types with WCET and deadline $\langle e(v), d(v) \rangle$
  - Edges $(u, v)$: minimum inter-release delays $p(u, v)$
  - General period parameter $P$

![Diagram of RB task model]

Period $P = 57$
Restrictions of DAG/RRT model

- Tasks are still *recurrent*
  - Always revisit source $J_1$
  - *No cycles allowed!*

Consequences:

- No *local loops*
  - Not *compositional* (for modes etc.)
Restrictions of DAG/RRT model

- Tasks are still *recurrent*
  - Always revisit source $J_1$
  - *No cycles allowed!*

**Consequences:**

- No *local loops*

  - Not *compositional* (for modes etc.)
Restrictions of DAG/RRT model

- Tasks are still *recurrent*
  - Always revisit source $J_1$
  - *No cycles allowed!*

- Consequences:
  - No *local loops*
  - Not *compositional* (for modes etc.)
The Digraph Real-Time (DRT) Task Model

- Branching, cycles (loops), ...
- *Directed graph* for each task
  - Vertices $v$: job types with WCET and deadline $\langle e(v), d(v) \rangle$
  - Edges $(u, v)$: minimum inter-release delays $p(u, v)$
DRT: Semantics

Path $\pi = (v_4)$
Path $\pi = (v_4, v_2)$
DRT: Semantics

Path $\pi = (v_4, v_2, v_3)$
Hierarchy of Models

- DRT: branching, loops, ...
- RB: branching
- GMF: different job types
- sporadic: explicit deadline
- L&L: implicit deadline

Feasibility test:
- arbitrary graph
- tree
- cycle graph
- three integers
- two integers

Expressiveness:
- difficult
- high
- efficient
- low
How to check the feasibility?
The Demand Bound Function [Baruah et al]

- General tool/technique for EDF schedulability analysis: $dbf(t)$
- Intuition:
  - Given a time interval length $t$
  - $dbf(t)$ bounds the demand for processor time within any $t$ interval

$$dbf(t) = 8 + 8$$
Example: L&L tasks \((e_i, p_i)\)

\[
dbf_{\tau_i}(t) = \frac{t}{p_i} \cdot e_i
\]
Example: Sporadic tasks: \((ei, di, pi)\)

\[
dbf_{\tau_i}(t) = \sum_{T_i \in \tau} ei \cdot \max \left\{ 0, \left\lfloor \frac{t - di}{pi} \right\rfloor + 1 \right\}
\]
Feasibility Test Using dbf()
Feasibility Test Using $\text{dbf}()$

**Theorem**

* A task system $\tau$ is preemptive uniprocessor feasible iff:

\[ \forall t \geq 0 : \sum_{T \in \tau} \text{dbf}_T(t) \leq t \]

**Challenges:**

1. How to calculate $\text{dbf}_T()$?
2. How to check existence of a violating $t$?

There exists a **Bound** for systems where the worst-case utilization $C$ (long-term rate) is less than 1.
Calculating the Bound

\( dbf(t) \) is bounded by \( C_{\text{max}} + t \cdot c \)

- **Linear bound for** \( dbf(t) \)
  - Slope: Less than 1
  - Intersection with \( t \) gives bound \( D \)
  - Check only up to \( D \)

\( C_{\text{max}} = \) sum of WCETs for all jobs
\( c = \) “the worst-case utilization”
(note: \( c < 1 \))
Calculating the Bound

Linear bound for $dbf(t)$
- Slope: Less than 1
- Intersection with $t$ gives bound $D$
- Check only up to $D$
Calculating demand pairs for graph models

Recall Task Model:

For each path, we have:
1. An execution demand $e$
2. A deadline $d$ for this demand

Call $\langle e, d \rangle$ a demand pair

Execution demand:
$$e = 1 + 5 + 2 = 8$$

Deadline:
$$d = 11 + 10 + 10 = 31$$

Demand pair: $\langle 8, 31 \rangle$
Path Abstraction: \textit{Demand Triples}

- Extend $\langle e, d \rangle$ with end vertex $v$
- Call $\langle e, d, v \rangle$ a \textit{demand triple}
- Allows \textit{extensions} to create new triples

Demand triple: $\langle 8, 31, J_5 \rangle$
Path Abstraction: Demand Triples

- Extend $\langle e, d \rangle$ with end vertex $v$
- Call $\langle e, d, v \rangle$ a demand triple
- Allows extensions to create new triples

Demand triple: $\langle 8, 31, J_5 \rangle$
New demand triple: $\langle 10, 41, J_1 \rangle$

Determine dbf iteratively:
1. Demand triples for 0-paths
2. Extend demand triples
3. Take maximum
Iterative Procedure

- Start with just the vertices (0-paths)
- Then, at each iteration:
  - *Extend* current demand triples
  - Optimization: Discard non-critical triples on the way
- Finally, *dbf* is maximum
Iterative Procedure

- Start with just the vertices (0-paths)
- Then, at each iteration:
  1. \textit{Extend} current demand triples
  2. Optimization: Discard non-critical triples on the way
- Finally: $\text{dbf}_T$ is maximum
Iterative Procedure

- Create all demand triples up to $D$:
  1. Start with all 0-paths, i.e., $\langle e(v), d(v), v \rangle$ for all vertices $v$
  2. Pick some stored demand triple $\langle e, d, u \rangle$
  3. **Create new demand triple:**
     - Choose successor vertex $v$ of $u$
     - $e' = e + e(v)$
     - $d' = d - d(u) + p(u, v) + d(v)$
     - $\langle e', d', v \rangle$ is new demand triple!
  4. Store $\langle e', d', v \rangle$ if
     - not stored yet, and
     - $d' \leq D$
  5. Repeat from 2 until no change

- **Efficient** procedure!
  - Note: Actual paths never stored
  - Optimizations: Discard non-critical triples along the way
DRT Schedulability: Summary

- Schedulability test for *EDF*, based on dbf
- First, compute *utilization* for all tasks
  - Based on most dense cycles in graphs
- Derive *bound D*
- Compute dbf(t) for all \( t \leq D \)
  - Uses iterative procedure with *demand triples*
  - Path abstraction to reduce complexity
- If \( t \leq D \) with dbf(t) > t found: \( \tau \) unschedulable
- Else: \( \tau \) schedulable
• How about “synchronization among tasks”? 
• How about “general constraints on job releases”? 
Task Automata [Fersman/Yi et al, TACAS 2002/TIMES tool]

- Task releases modeled by *timed automata*
- Advantage: Very *expressive*
- Disadvantage: Schedulability test *expensive* (or even impossible)
States/Configurations of automata

A state is a triple: \((m, u, q)\)

- Location (node)
- Clock assignment (valuation)
- Job queue

Feasibility/schedulability is a reachability problem for timed automata

Code the problem and use UPPAAL ...
Hierarchy of Models

Timed automata

- DRT: branching, loops, ...
- RB: branching
- GMF: different job types
- sporadic: explicit deadline
- L&L: implicit deadline

Feasibility test:
- Arbitrary graph
- Tree
- Cycle graph
- Three integers
- Two integers

Expressiveness:
- Difficult (Strongly (co)NP-hard, Pseudo-Polynomial)
- Efficient (low)

High
• How about “static priority scheduling”? 
Hierarchy of Models

- **Static-priority Schedulability**

  - **Easy**
    - **Task automata**
    - **arbitrary graph**
    - **branching, loops, ...**
  - **Efficient**
    - **RB**
    - **branching**
    - **tree**
  - **Strongly NP hard**
    - **GMF**
    - **different job types**
    - **cycle graph**
  - **L&L**
    - **sporadic**
    - **explicit deadline**
    - **three integers**
  - **L&L**
    - **implicit deadline**
    - **two integers**

- **Expressiveness**
  - **difficult**
  - **high**
## Summary

<table>
<thead>
<tr>
<th>Models</th>
<th>Analysis Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Feasibility -- EDF-schedulability</td>
</tr>
<tr>
<td>Timed Automata</td>
<td>Strongly NP-hard</td>
</tr>
<tr>
<td>General graphs (Di-graph)</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>Trees/DAGs (RRT)</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>Cyclic graphs (GMF)</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>Sporadic, D&lt;T</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>Sporadic, D=T (Layland &amp; Liu)</td>
<td>Linear</td>
</tr>
</tbody>
</table>

[Stigge/Wang, ECRTS 2012]
• Static priority schedulability is **Strongly NP-hard** for all except L&L (and sporadic) model

• What to do?
OUTLINE

• An Overview on RT workload models
  – from the simplest to the most complex
  – from Liu & Layland classic periodic model to timed automata

• Recent Results
  – the graph-based model [RTAS 2011, RTSS 2013, Stigge & Yi]
  • Combinatorial Abstraction Refinement for Schedulability Analysis
Problem setting

Is $C$ lowest-priority feasible?
Problem setting

Is C lowest-priority feasible?

Scheduling window of C
Problem setting

Is $C$ lowest-priority feasible?

Scheduling window of $C$
Problem setting

Is C lowest-priority feasible?

Problem: Combinatorial Explosion!
Every Request Function corresponds to an execution path in the graph.
Request Functions: Schedulability Test

Lemma

A vertex $v$ is \textit{lp-feasible} if for all \textit{combinations} of request functions $r_f^{(T)}$ of higher priority tasks:

$$\exists t \leq d(v): e(v) + \sum_{T \in \tau} r_f^{(T)}(t) \leq t.$$  \hfill (1)

Problem: \textit{Combinatorial Explosion!}
Every Request Function corresponds to an execution path in the graph.
Every Request Function corresponds to an execution path in the graph.
Overapproximation: \textbf{mrf} (maximum of rf’s)

Every Request Function corresponds to an execution path in the graph
Overapproximation: \textit{mrf}

- Trick: Define an overapproximation
- Let \( \text{mrf}^{(T)}(t) \) be \textit{maximum} of \textit{all} \( \text{rf}^{(T)}(t) \) for a task \( T \).
- New test:
  \[ \exists t \leq d(v) : e(v) + \sum_{T \in \mathcal{T}} \text{mrf}^{(T)}(t) \leq t. \]
- \textit{Efficient}: Only \textit{one} test, no combinatorial explosion
- Problem: Imprecise!
Overapproximation: \( mrf \)

- Trick: Define an overapproximation
- Let \( mrf^{(T)}(t) \) be \textit{maximum} of all \( rf^{(T)}(t) \) for a task \( T \).
- New test:
  \[
  \exists t \leq d(v) : e(v) + \sum_{T \in \tau} mrf^{(T)}(t) \leq t.
  \]
- \textit{Efficient}: Only one test, no combinatorial explosion
- Problem: Imprecise!

\textbf{How can we get efficiency \textit{and} precision?}
Define an *abstraction tree* per task:

- Leaves are concrete \( rf \)
- Each node: maximum function of child nodes
- Root is \( mrf \), maximum of all \( rf \)
Combinatorial Abstraction Refinement

New Algorithm:
- Test *one* combination of all $mrf$.
- If fp-feasible: done
- Otherwise: Replace *one* $mrf$ with all child nodes, get 2 new combinations to test
- Repeat until:
  - All combinations show fp-feasibility, or
  - A combination of leaves shows non-fp-feasibility
Evaluation: Tested vs. Total Combinations

10^5 samples of single-job tests.
- Executed tests: in 99.9% of all cases, less than 100
- Total combinations possible: up to 10^{12}
Evaluation: Runtime vs. Utilization

Comparing runtimes of
- EDF-test using dbf (pseudo-polynomial)
- SP-test based on *Combinatorial Abstraction Refinement*
Hierarchy of Models

- **Timed automata**
  - DRT: branching, loops, ...
  - RB: branching
  - GMF: different job types
  - sporadic: explicit deadline
  - L&L: implicit deadline

- Expressiveness:
  - Strongly (co)NP-hard
  - Pseudo-Polynomial

- Feasibility test:
  - arbitrary graph
  - tree
  - cycle graph
  - three integers
  - two integers

- difficulty:
  - difficult
  - efficient
## Summary

<table>
<thead>
<tr>
<th>Models</th>
<th>Analysis Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Feasibility -- EDF-schedulability</td>
</tr>
<tr>
<td>Timed Automata</td>
<td>Strongly NP-hard (or impossible)</td>
</tr>
<tr>
<td>Original RTC @ ETH</td>
<td>Strongly NP-hard</td>
</tr>
<tr>
<td>General graphs (Di-graph)</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>Trees/DAGs (RRT)</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>Cyclic graphs (GMF)</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>Finitary RTC</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>Sporadic, D&lt;T</td>
<td>Pseudo-P</td>
</tr>
<tr>
<td>Sporadic, D=T (Layland &amp; Liu)</td>
<td>Linear</td>
</tr>
</tbody>
</table>
Conclusion

• Analysis is often easier for models with no “urgency/synchrony” e.g.
  – job releases with minimal separation distance, no upper bound
  – no synchronization between tasks
• Analysis related static priority is “more difficult” than feasibility (EDF-schedulability)
  – It is strongly coNP-hard for all models more expressive than GMF
• Efficient techniques exist:
  – Combinatorial Abstraction [RTSS 2013, Stigge/Yi]
  – Finitary RTC [RTSS 2013, Guan/Yi]
• Current/future work: making a tool integrating all these!