Testing Deadline Misses for Real-Time Systems Using Constraint Optimization Techniques

Stefano Di Alesio$^{1,2}$
Arnaud Gotlieb$^1$
Shiva Nejati$^1$
Lionel Briand$^{1,2}$

$^1$Simula Research Laboratory
$^2$University of Luxembourg

SweConsNet 2012
14/05/2012
We present a technique to use Constraint Optimization to test deadline misses for RTES

Performance Requirements (PRs) vs. Real Time Embedded Systems (RTES)

Using Constraint Programming for Verification and Validation of RTES

Using Constraint Programming for Verification and Validation of RTES

Evaluation, Experience and Current Work
RTES are typically safety-critical, and thus bound to meet strict Performance Requirements.
Performance Requirements are the most difficult requirements to verify

They depend on the environment the software interacts with (hw devices)

They depend on the computing platform on which the software runs

They constraint the entire system’s behavior and thus can’t be checked locally
RTES have concurrent interdependent tasks which have to finish before their deadlines

Each task has a deadline (i.e., latest finishing time) w.r.t. its arrival time

Some task properties depend on the environment, some are design choices

Each task can trigger other tasks, and can share computational resources with other tasks
Particular sequences of arrival times of tasks can determine deadline miss scenarios

\[ j_0, j_1, j_2 \text{ arrive at } at_0, at_1, at_2 \text{ and must finish before } dl_0, dl_1, dl_2 \]

\[ j_1 \text{ can miss its deadline } dl_1 \text{ depending on when } at_2 \text{ occurs!} \]
We are looking for sequences of arrival times maximizing the likelihood of deadline misses.

Arrival times for tasks in a RTES depend on the environment:

\[ a_1 = 1, \quad a_2 = 3, \quad a_3 = 3, \quad a_4 = 7 \]

Arrival times can be tuned during software testing:

\[ a_1 = 1, \quad a_2 = 3, \quad a_3 = 4, \quad a_4 = 7 \]

A sequence of arrival times identified by our approach as likely to lead to a deadline miss defines a Stress Test Case.
This problem has been well studied, but each existing approach has its weaknesses

<table>
<thead>
<tr>
<th></th>
<th>Schedulability Theory</th>
<th>Model Checking</th>
<th>Genetic Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basis</strong></td>
<td>Mathematical Theory</td>
<td>System Modeling</td>
<td>System Modeling</td>
</tr>
<tr>
<td><strong>Background</strong></td>
<td>WCET, Queuing Theory, etc.</td>
<td>Fixed-point Computation</td>
<td>Meta-Heuristic Search</td>
</tr>
<tr>
<td><strong>Weaknesses</strong></td>
<td>Assumptions, Multi-Core</td>
<td>Complex Modeling</td>
<td>Non-Exhaustive Search</td>
</tr>
</tbody>
</table>

We model the RTES Design, Platform and PRs through a Integer Linear Program solved by CPLEX

The System is modeled through constants, variables and constraints

Performance Requirements are modeled as objective functions

The System Design & Platform

Performance Requirements

RTES

Optimization Engine

Stress Test Cases

Stress Test Cases are the optimal solutions for the constraint program

Stefano Di Alesio - 9/20
Our approach tries to mitigate some weaknesses found in related works

<table>
<thead>
<tr>
<th></th>
<th>Schedulability Theory</th>
<th>Model Checking</th>
<th>Genetic Algorithms</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basis</strong></td>
<td>Mathematical Theory</td>
<td>System Modeling</td>
<td>System Modeling</td>
<td>System Modeling</td>
</tr>
<tr>
<td><strong>Background</strong></td>
<td>WCET, Queuing Theory, etc.</td>
<td>Fixed-point Computation</td>
<td>Meta-Heuristic Search</td>
<td>Mathematical Optimization</td>
</tr>
<tr>
<td><strong>Weaknesses</strong></td>
<td>Assumptions, Multi-Core</td>
<td>Complex Modeling</td>
<td>Non-Exhaustive Search</td>
<td>Performance / Scalability (?)</td>
</tr>
</tbody>
</table>

Tasks and Platform design properties are modeled through constants.

**Assumption 1:** The scheduler checks tasks for preemptions at regular intervals ($tq$)

```plaintext
// T: Observation interval (range of time quanta)
int tq = ...;
range T = 0..tq-1;

// c: Number of Processor Cores
int c = ...;

// n: Number of tasks
int n = ...;
range J = 0..n-1;

tuple TaskExecution {
    int task;
    int execution;
}

int priority[J] = ...;
int task_deadline[J] = ...;
int max_interarrival_time[J] = ...;
int min_duration[J] = ...;
int max_duration[J] = ...;
int triggers[J, J] = ...;
int dependent[J, J] = ...;
```

**Assumption 2:** The context switching time between tasks is negligible w.r.t. $tq$
Tasks and Platform real time properties are modeled through variables

\begin{align*}
\text{efe}(a) & \overset{\text{def}}{=} \text{earliest time when } a \text{ could start if an unlimited number of cores was available} \\
\text{active}(a, t) & \overset{\text{def}}{=} \begin{cases} 
1 & \text{if } a \text{ is executing at time } t \\
0 & \text{otherwise} \end{cases} \text{ [1]}
\end{align*}

<table>
<thead>
<tr>
<th>Time quanta</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
</table>
| Task executions | \begin{bmatrix} 
a_0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
a_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
a_2 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
a_3 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
\end{bmatrix} |

1. Well-Formedness: relations directly following from variables definitions

The Platform Scheduler behavior is modeled through 5 sets of constraints

/* I. Well-formedness constraints */
forall(a in A) {

    wf3: eligible_for_execution[a] <= start[a];
    wf4: start[a] <= end[a];

    if(prevc(A, a).task == a.task)
        wf6: eligible_for_execution[a] == maxl(arrival_time[a],
            end[prevc(A, a)]);
    else
        wf7: eligible_for_execution[a] == arrival_time[a];

    wf8: duration[a] == sum(t in T) active[a, t];

    forall(t in T) {
        wf9: t == start[a] => active[a, t] == 1;
        wf10: t == end[a] - 1 => active[a, t] == 1;
        wf11: t <= start[a] - 1 =>
            active[a, t] == 0;
        wf12: t >= end[a] => active[a, t] == 0;
    }
}
The Platform Scheduler behavior is modeled through 5 sets of constraints

2. Temporal Ordering: executions, triggering and resource usage relations

```plaintext
/* II. Temporal Ordering constraints */
for all (a in A) {
    for all (a1 in A : a1.task == a.task &&
                a1.execution == a.execution - 1)
        to1: start[a] >= end[a1];
    for all (a1 in A : triggers[a.task, a1.task] == 1)
        to2: end[a] == arrival_time[a1];
    for all (a1 in A : dependent[a.task, a1.task] == 1) {
        to3: start[a] != start[a1];
        to4: start[a] <= start[a1] - 1 =>
            start[a] >= end[a];
    }
}
```

3. Multicore: computing capacity of the platform

```plaintext
/* III. Multi-core Constraint */
for all (t in T)
    mc: sum(a in A) active[a, t] <= c;
```
The Platform Scheduler behavior is modeled through 5 sets of constraints

4. Preemptive Scheduling: priority-driven preemptive scheduling behavior

```c
/* IV. Preemptive Scheduling Constraints */
forall(t in T, a0 in A, a1 in A)
  ps2: (active[a0, t] == 0 &&
        active[a1, t] == 1 &&
        sum(a2 in A) active[a2, t] == c &&
        eligible_for_execution[a0] <= t &&
        end[a0] >= t+1)
    =>
        (priority[a1.task] >=
         priority[a0.task]);
```

5. Good CPU Usage: scheduler’s CPU Usage optimizations

```c
/* V. Good CPU Usage Constraints */
forall(a in A, t in T)
  gcu1: (sum(a1 in A) active[a1, t]<=c-1)
      =>
        (active[a, t] == 1 ||
         eligible_for_execution[a]>= t+1 ||
         end[a] <= t);

forall(a0 in A, a1 in A, t in T : t < tq-1)
  gcu2: (active[a0, t] == 1 &&
          active[a0, t+1] == 0)
      =>
        (end[a0] == t+1 ||
         (active[a1, t+1] == 1 =>
          priority[a1.task] >=
          priority[a0.task]+1));
```
The Performance Requirement is modeled as an objective function to maximize

The objective function is a counter for deadline misses

```plaintext
tf13: deadline_miss[a] == end[a] - task_execution_deadline[a];
maximize
sum(a in A)(max(0, min(1, deadline_miss[a])));
```

Main limitation: each deadline miss is given the same weight in the sum

```
f \equiv \sum_i \max(0, \min(1, e(a_i) - dl(a_i)))
```

Potential alternative [1]: non-linear objective function to weight deadlines

```
\tilde{f} \equiv \sum_i 2^{e(a_i) - dl(a_i)}
```

Correctness was evaluated analyzing the results computed starting from a set of toy examples:

<table>
<thead>
<tr>
<th></th>
<th>$j_0$</th>
<th>$j_1$</th>
<th>$j_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>exec($j$)</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$p(j)$</td>
<td>100</td>
<td>101</td>
<td>102</td>
</tr>
<tr>
<td>$dl(j)$</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>max$_{ia}(j)$</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>min$_{dr}(j)$</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>max$_{dr}(j)$</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

In this case, we found a solution where both executions of task $j_0$ miss their deadline.

arrival_time (at): [0 3 2 4 0 3]
duration (dr): [3 3 2 2 3 3]
eligible_for_execution (efe): [0 7 2 4 0 3]
start (s): [0 7 2 4 0 3]
end (e): [7 10 4 6 3 6]
task_execution_deadline (edl): [3 6 4 6 3 6]
deadline_miss: [4 4 0 0 0 0]
active: 
[[1 1 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 1 1 1]
 [0 0 1 1 0 0 0 0 0 0]
 [0 0 0 0 1 1 0 0 0 0]
 [1 1 1 0 0 0 0 0 0 0]
 [0 0 0 1 1 1 0 0 0 0]]
Performance was evaluated by measuring solving times with increasing input size.

We evaluated Performance by increasing $n$ and $tq$.

It took a significant amount of time to find all optimal solutions.

Most optimal solutions were found shortly after the search started, even if the search took a much more time to terminate.
Our current work relies on improving the approach scalability with respect to $n$ and $tq$

**Problem:** it’s hard to compute the active matrix ($2^{n \times \text{exec}(j_n) \times tq}$ possible values)

**Idea:** we don’t really need the whole matrix, but just to know where the 1’s are!
In summary, Constraint Optimization is a promising approach to derive Stress Test Cases for RTES

System Platform, Tasks and PRs are modeled in a Constraint Program

Solving the CP finds tunable values more likely to stress test the system

Significant advantages over other approaches encourage future work

Questions?