Runtime verification of real-time applications using trace data and model requirements

Progress Report Meeting
December 10, 2015

Raphaël BEAMONTE   Michel DAGENAIS

Distributed Open Reliable Systems Analysis Lab
Computer and Software Engineering Department
École Polytechnique de Montréal

Introduction

- Low-overhead tracing is available
- But trace analysis requires users to have kernel knowledge
- So what about automating the analysis ?
  - CAE suggested to verify applications' execution using specifications
  - Ericsson is working towards programming at model level
  - **Why couldn't we do both?**
- $\Rightarrow$ model-based constraints

Table of Contents

Analyzed case

Occasional missing of deadlines

Problem

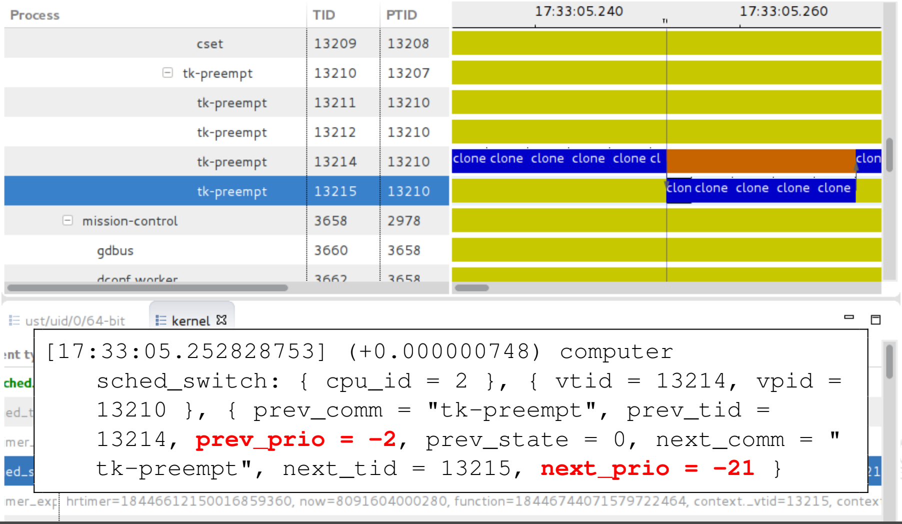
In a task that appears a lot of times, some deadlines are missed occasionnaly.

Analysis

What happened on the kernel side when the deadlines were missed ?

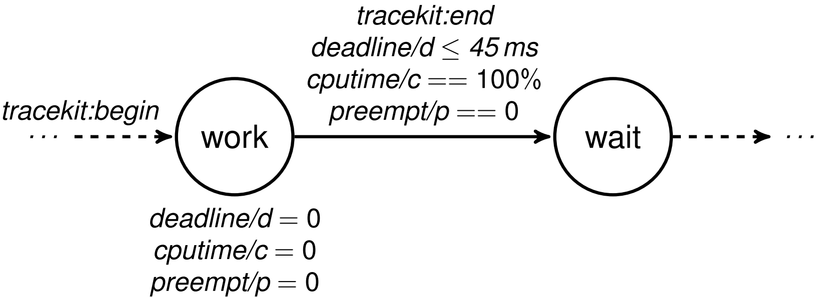
Analyzed case

Occasional missing of deadlines



Analyzed case

State machine representation



State machine representation of `tk-preempt`'s work using constraints to check if our process spent at most 45 ms working, used 100 % of the CPU time and was not preempted during its critical real-time task

Analyzed case

State Chart XML representation

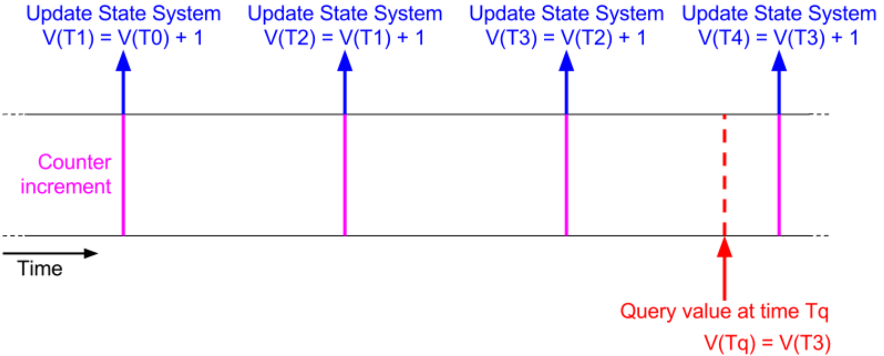
```
<?xml version="1.0" encoding="UTF-8"?>
<scxml xmlns="http://www.w3.org/2005/07/scxml" version="1.0">
    <initial>
        <transition event="tracekit:begin" target="work"/>
    </initial>

    <state id="work">
        <onentry>
            <assign location="deadline/d" expr="0"/>
            <assign location="preempt/p" expr="0"/>
            <assign location="cputime/c" expr="0"/>
        </onentry>

        <transition event="tracekit:end" target="wait"
                cond="deadline/d &lt;= 45ms; preempt/p == 0;
                    cputime/c == 100%"/>
    </state>

    <state id="wait">
    </state>
</scxml>
```

Internal state system

- Used to store the state of different metrics per PID through time
- Three categories of variables:
  - Independent from the current state (state system free)
  - Counters
  - Timers
- Variables are categorized according to the number of queries needed to get their value at a given time: 0, 1 or 2

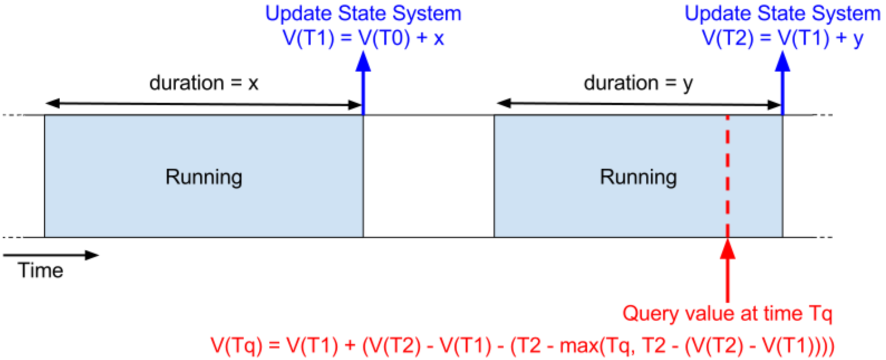
Internal state system

Counters



Internal state system

Timers



Constraint verification

How it is done

- Done when following a transition
- Consider T_{init} is the timestamp of the variable initialization and T_{trans} the timestamp of the event setting off the transition
- Consider $V(T_x)$ the value of the variable in the state system at timestamp T_x
- The variable value for constraint verification will be $V(T_{trans}) - V(T_{init})$

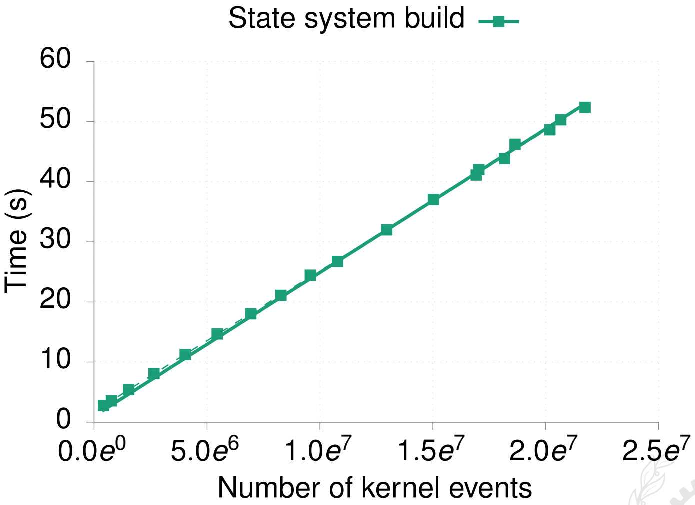
Constraint verification

Constraints status: VALID vs. INVALID vs. UNCERTAIN

```
Received tracekit:begin at 18:27:53.173 147 428
    Entering state: Work
    Variables:
        - deadline/d = 0
        - cputime/c = 0
        - preempt/p = 0
Received tracekit:end at 18:27:53.218 552 778
    Entering state: Wait
    Constraints:
        - deadline/d <= 45ms [INVALID] value: 45.4054ms
        - cputime/c == 100% [INVALID] value: 99.9806%
        - preempt/p == 0 [INVALID] value: 1
```

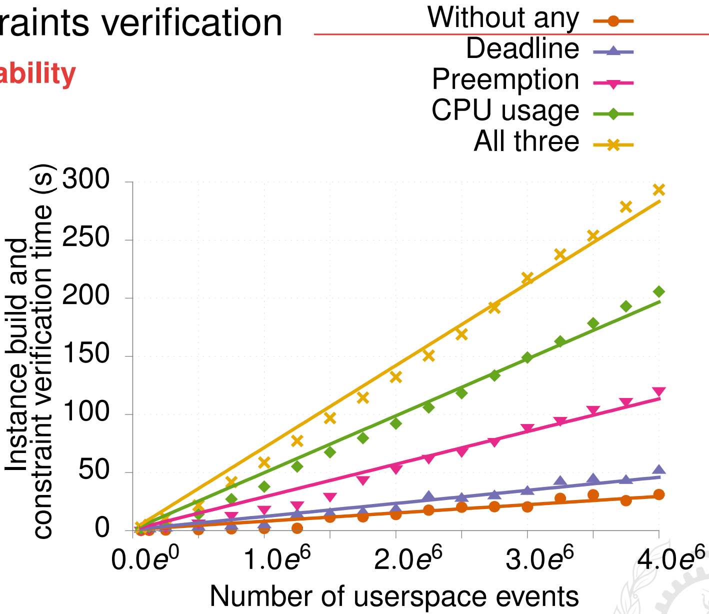
Constraints verification

Scalability



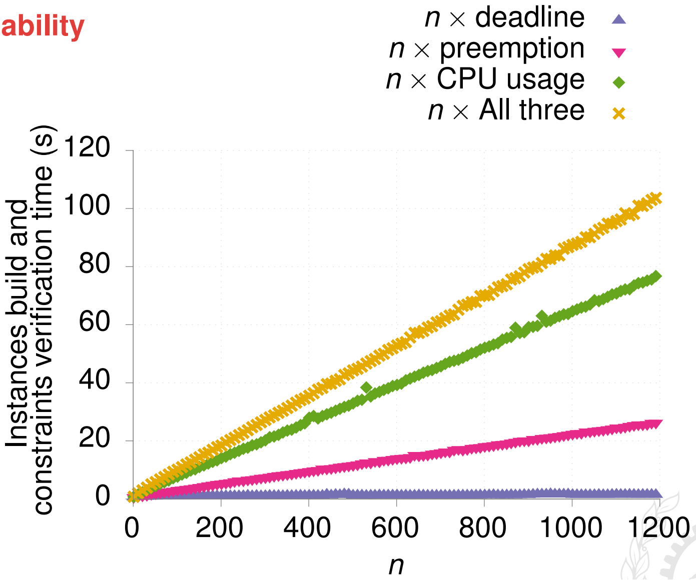
Constraints verification

Scalability



Constraints verification

Scalability



Constraints analysis

- Now we know where are the problems. But we'd like to know why there was a problem!
- Step 1: Data extraction
  - Depends on the category of variable
  - Aims to extract relevant data for next step
- Step 2: Data organization
  - Depends on the category of variable
  - Aims to identify the normal and abnormal behaviors and regroup them
- Step 3: Algorithm application
  - Different approaches for absolute (e.g. $== 0$) and relative (e.g. < 2) constraints
  - Same algorithm used no matter the category of variable

Algorithms

Absolute

- Each element is part of the problem: each occurrence adds to the responsibility of that element

```
1 different constraint(s) with invalid status on 31021 different instances
Constraint preempt/p == 0 (wait) has been invalid 2036 times

Process should not have been preempted.
Processes identified as potential cause of constraint violation:
   -------------------------------------------------------------------
   | Task        | TID   | Occurrences | Responsibility | Max. Prio. |
   |=================================================================|
   | tk-preempt  | 13215 |        1905 |       93.5658% |        -21 |
   | watchdog/2  |    32 |         144 |        7.0727% |       -100 |
   -------------------------------------------------------------------
   Total preemption occurrences identified: 2049

   In 100.0% of cases, the process which preempted had a higher priority.
```

Algorithms

Relative

- Amongst the elements, some are part of the problem, some not.
- We need to identify what should have been instead.

```
1 different constraint(s) with invalid status on 31021 different instances
Constraint preempt/p < 2 (wait) has been invalid 13 times

Invalid preempting tasks list [tk-preempt 13215, watchdog/2 32] has been encountered
     13 times
Preempting tasks lists identified as potential expected list instead of this invalid
     one:
   ---------------------------------------------------------------
   | Preempting tasks list | Distance | Occurrences | Weight     |
   |=============================================================|
   | [tk-preempt 13215]    |        1 |        1892 |   90.0956% |
   | []                    |        2 |       28984 |   86.8090% |
   | [watchdog/2 32]       |        1 |         131 |   46.5711% |
   ---------------------------------------------------------------
```

Algorithms

Relative

```
1 different constraint(s) with invalid status on 31021 different instances
Constraint cputime/c > 99% (wait) has been invalid 1905 times

Invalid interruption list {PREEMPTED, tk-preempt 13215=1} has been encountered 1884
     times
Interruption lists identified as potential expected list instead of this invalid one:
---------------------------------------------------------------
| Valid interruption list | Distance | Occurrences | Weight   |
|=============================================================|
| []                      | 1        | 28984       | 95.0000% |

Invalid interruption list {PREEMPTED, tk-preempt 13215=1, PREEMPTED, watchdog/2 32=1}
      has been encountered 21 times
Interruption lists identified as potential expected list instead of this invalid one:
-------------------------------------------------------------------------------------
| Valid interruption list                       | Distance | Occurrences | Weight   |
|===================================================================================|
| [[PREEMPTED, watchdog/2 32, [860.0, 13400.0]]]| 1        | 131         | 95.0000% |
| []                                            | 2        | 28984       | 94.5501% |
```

Algorithms

Relative

- **Distance** = edit distance (only add or delete, not replace) between valid and invalid lists of elements

$$
\begin{array}{rll}
[a, b, c]\text{to } [a, b] & (d = 1) & \text{remove } c \\
\text{to } [a] & (d = 2) & \text{remove } b \\
\text{to } [\,] & (d = 3) & \text{remove } a \\
\text{to } [e] & (d = 4) & \text{insert } e
\end{array}
$$

- **Occurrences** = number of times we encountered that valid list of elements

Algorithms

Relative

Weight = weight as computed by:

$$W_i = W_{r\,i} - P$$

With the uncertainty penalty $P = F_C \times \left(1 - \frac{1}{N_{valid}}\right)$ $(F_C = 0.1)$

And the relative weight $W_{r\,i}$:

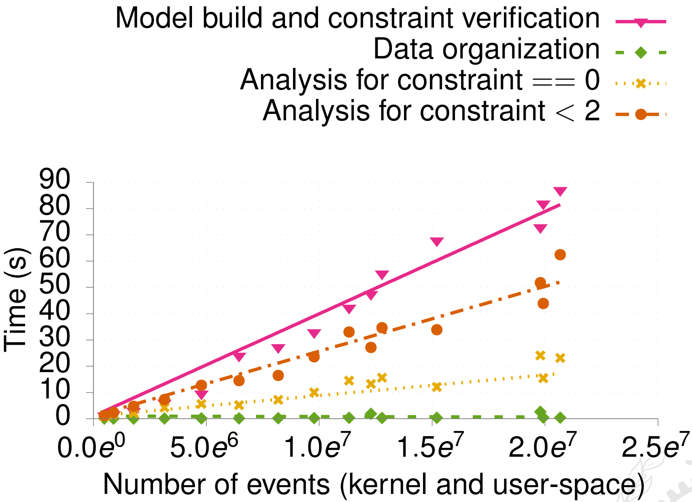
$$W_{r\,i} = \frac{O_i}{\sum_{d_j \leq d_i} O_j} \times \frac{d_i}{\max(1, s)} + \frac{s - d_i}{\max(1, s)}$$

With:

- O_i the number of occurrences of the valid list i
- d_i the distance between this list and the invalid one
- $\sum_{d_j \leq d_i} O_j$ the sum of O_j for all list j with $d_j <= d_i$
- s the size (or number of elements) of the invalid list

Constraints analysis

Scalability



Conclusion

- New approach using constraints to automatically detect problems using traces
- Overview of how we verify constraints
- Algorithms to do a more thorough analysis of the constraints violations
- Linear scaling of the approaches used
- Future work:
  - Finish the implementation of the analysis algorithm (counters and timers done, state system free variables to go)
  - Track 3 of the Ph.D. $\Rightarrow$ from trace to model-based constraints

Thank you.
Any question?

raphael.beamonte@polymtl.ca

Slides:

www.dorsal.polymtl.ca/~rbeamonte/dorsal-pm-dec2015.pdf