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

Progress Report Meeting
May 5, 2016

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

Variables

Counters variables incremented each time *something* happens (i.e. number of system calls)

Timers variables following the duration of *something* (i.e. cpu usage)

State system free variables based on userspace events (i.e. deadline)

Algorithms

Two approaches

- When the constraint is absolute *(counter* $==0$, *timer* $==100\%, \ldots)$
  - Every occurrence is a problem: classify by level of responsibility
- When the constraint is relative
  - How can you find the problem ?
  - $\Rightarrow$ **Compare valid and invalid instances**

Algorithms

Data extraction: what should we compare?

Counters the events that increment the counter (i.e. `syscall_entry_*` for system calls)

Timers the occurrences and durations of periods incrementing the timer (i.e. `sched_switch` to unschedule then to schedule back for cpu usage)

State system free kind of everything that happens in the period that influences the variable (i.e. the state of the process for a dealine)

Algorithms

Weighting of valid instances

The **weight** is 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 $W_{r\,i}$:

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

With:

- O_i the number of occurrences of the valid list i
- d_i the distance between this list and the invalid one
- $\sum_{d_j \le d_i} O_j$ 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

Case studies

Examples of results: Too low priority (jackd)

- Run a real-time application (`JACK2`, the audio server) with a low real-time priority (-1), and pinned on a given CPU (2)
- Force its preemption by running another application (cpuburn) on the same CPU, with a higher priority

```
$ taskset -c 0 jackd -P 1 -v -d alsa -H
...
creating alsa driver ... hw:0|hw:0|1024|2|48000|0|0|
    hwmon|swmeter|-|32bit
configuring for 48000Hz, period = 1024 frames (21.3 ms),
     buffer = 2 periods
...
Jack: alsa_pcm: xrun of at least 353.963 msecs
```

Case studies: Too low priority (jackd)

Invalid interval set

```
IntervalSet(1): [
    [BLOCKED on poll, 3, [4.1705323E7, 4.1705323E7]]
    [RUNNING, 14, [238189.0, 238189.0]]
    [SYSCALL futex, 1, [10094.0, 10094.0]]
    [Interrupted by IRQ29 (snd_hda_intel), 1, [977.0, 977.0]]
    [WAKING, 3, [5.85536417E8, 5.85536417E8]]
    [SYSCALL poll, 6, [58995.0, 58995.0]]
    [SYSCALL ioctl, 5, [103256.0, 103256.0]]
    [SYSCALL write, 5, [31727.0, 31727.0]]
]
```

Case studies: Too low priority (jackd)

Valid interval sets weighting against invalid one

Valid list	Distance	Occurrences	Weight
IntervalSet(105): [[BLOCKED on poll, 2, [2.0703525E7, 2.2803038E7]] [RUNNING, 3, [31941.0, 111629.0]] [WAKING, 2, [10907.0, 85443.0]] [SYSCALL poll, 4, [34946.0, 90288.0]]]	30	105	93.33%
IntervalSet(2): [[BLOCKED on poll, 2, [2.0767886E7, 2.17529E7]] [RUNNING, 3, [61252.0, 87261.0]] [WAKING, 2, [15856.0, 23237.0]] [SYSCALL poll, 5, [43815.0, 76711.0]] [Interrupted by HRTIMER, 1, [5848.0, 15496.0]]]	31	2	13.28%
IntervalSet(4): [[BLOCKED on poll, 2, [2.0757138E7, 2.1794678E7]] [RUNNING, 4, [46599.0, 115511.0]] [WAKING, 2, [13054.0, 24173.0]] [SYSCALL poll, 4, [32867.0, 61412.0]] [Interrupted by HRTIMER, 1, [7908.0, 12598.0]]]	33	4	9.62%

Case studies: Too low priority (jackd)

Computed interval set difference *(using only weights >= 50%)*

```
Diff: [
    [RUNNING, 11, [126560.0, 206248.0]]
    [BLOCKED on poll, 1, [1.8902285E7, 2.1001798E7]]
    [WAKING, 1, [5.85450974E8, 5.8552551E8]]
    [SYSCALL futex, 1, [10094.0, 10094.0]]
    [SYSCALL ioctl, 5, [103256.0, 103256.0]]
    [SYSCALL poll, 2, [0.0, 24049.0]]
    [SYSCALL write, 5, [31727.0, 31727.0]]
    [Interrupted by IRQ29 (snd_hda_intel), 1, [977.0, 977.0]]
]
```

Case studies: Too low priority (jackd)

Computed responsibility (state analysis)

State	Responsibility for added time
WAKING	96.65%
BLOCKED on poll	3.29%
RUNNING	0.03%
SYSCALL ioctl	0.02%
SYSCALL write	0.01%
SYSCALL poll	0.00%
SYSCALL futex	0.00%
Interrupted by IRQ29 (snd_hda_intel)	0.00%

Minimum responsibility for a case to be considered: 64.83%

Case studies: Too low priority (jackd)

Critical path analysis triggered by the state analysis

Critical path state	Responsibility for added time
jackd PREEMPTED	96.51%
swapper/0 RUNNING	1.88%
swapper/0 PREEMPTED	1.55%
jackd RUNNING	0.05%
irq/29-snd_hda_ PREEMPTED	0.01%
irq/29-snd_hda_ RUNNING	0.00%

Minimum responsibility for a case to be considered: 60.79%

Case studies: Too low priority (jackd)

CPUTop analysis triggered by the critical path analysis

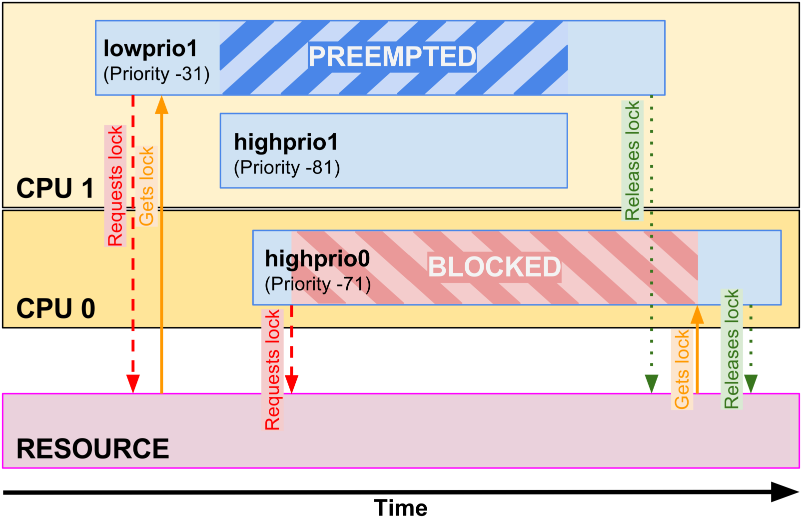
```
Analyzed CPUs: 0
Analyzed timerange: [11:54:37.711 896 155, 11:54:38.319 519 477]
```

Per-TID Usage	Process	Migrations	Priorities
96.34%	cpuburn (11371)	0	[-61]
0.29%	bash (11375)	1	[20]
0.13%	bash (11376)	2	[20]
0.11%	bash (11377)	0	[20]
0.10%	/usr/bin/x-term (3154)	2	[]

```
Priorities of the PREEMPTED process during that interval: -2
```

Case studies

Examples of results: In-kernel wake lock priority inversion



Case studies: In-kernel wake lock priority inversion

Computed responsibility (state analysis)

State	Responsibility for added time
BLOCKED on open	100.00%
RUNNING	0.00%
SYSCALL open	0.00%
SYSCALL gettid	0.00%

```
Minimum responsibility for a case to be considered: 56.70%
```

Case studies: In-kernel wake lock priority inversion

Priority inversion analysis triggered by the state analysis

```
Duration of active 'sched_pi_setprio'...
... in valid instances (maximum): 170.909us
... in invalid instances (average): 329.534us

Active 92.81% more time in invalid instances than in valid instances.

Verdict: Very high probability of a priority inversion
```

Critical path analysis triggered by the state analysis

Critical path state	Responsibility for added time
lowprio1 PREEMPTED	100.00%

```
Minimum responsibility for a case to be considered: 100.00%
```

Case studies: In-kernel wake lock priority inversion

CPUTop analysis triggered by the critical path analysis

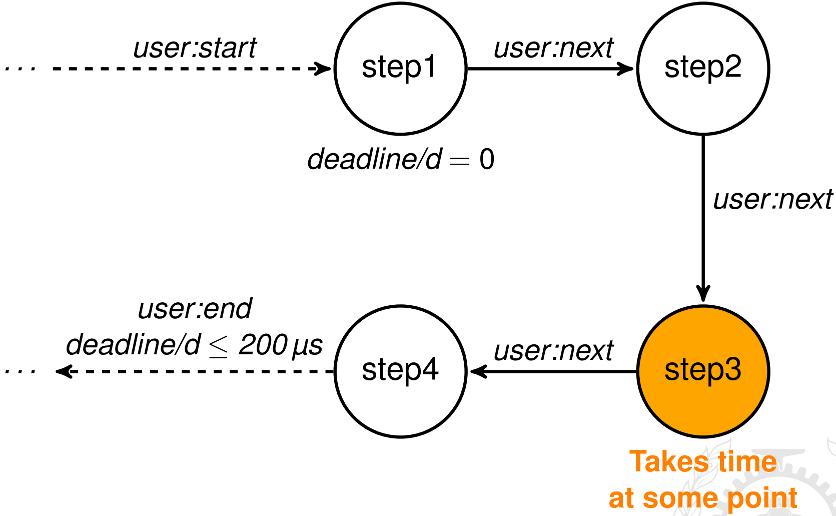
```
Analyzed CPUs: 1
Analyzed timerange: [21:40:35.867 825 012, 21:40:36.033 045 371]
```

Per-TID Usage	Process	Migrations	Priorities
99.18%	highprio1 (26408)	0	[-100, -81]
0.82%	lowprio1 (26407)	0	[-71]
0.00%	lttng-consumerd (26369)	0	[20]
0.00%	lttng-consumerd (26370)	0	[20]
0.00%	Cache2 I/O (3011)	0	[20]

```
Priorities of the PREEMPTED process during that interval: -71
```

Case studies

Examples of results: Bad code



Case studies: Bad code

Computed responsibility (state analysis)

State	Responsibility for added time
RUNNING	99.89%
Interrupted by HRTIMER	0.04%
Interrupted by SOFTIRQ_TIMER	0.03%
Interrupted by SOFTIRQ_RCU	0.02%
SYSCALL write	0.02%
Interrupted by SOFTIRQ_SCHED	0.00%
SYSCALL gettid	0.00%

Minimum responsibility for a case to be considered: 64.94%

Case studies: Bad code

State machine state analysis triggered by the state analysis

Model state	Responsibility for added time
step3	99.92%
step1	0.05%
step4	0.03%
step2	0.01%

Case studies

Examples of results: Preempted waker

- Instrumentation of `cyclictest` to take a snapshot when a period goes over a threshold
- The application was then started on a machine, and ran for a considerable amount of time before saving the trace
- We didn't know what happened. We used our analysis approach on it.

Case studies: Preempted waker

Computed responsibility (state analysis)

State	Responsibility for added time
BLOCKED on rt_sigtimedwait	99.95%
RUNNING	0.03%
SYSCALL getcpu	0.01%
SYSCALL clock_gettime	0.01%

Minimum responsibility for a case to be considered: 56.68%

Case studies: Preempted waker

Critical path analysis triggered by the state analysis

Critical path state	Responsibility for added time
ktimersoftd/3 PREEMPTED	99.68%
ktimersoftd/3 RUNNING	0.24%
cyclictest PREEMPTED	0.08%

Minimum responsibility for a case to be considered: 52.77%

Case studies: Preempted waker

CPUTop analysis triggered by the critical path analysis

```
Analyzed CPUs: 3
Analyzed timerange: [15:55:53.528 529 561, 15:55:53.536 012 105]
```

Per-TID Usage	Process	Migrations	Priorities
99.73%	irq/154-hpd (162)	0	[-51]
0.26%	irq/25-54200000 (163)	0	[-51]
0.01%	ktimersoftd/3 (32)	0	[-2]
0.00%	irq/22-Tegra PC (160)	0	[-51]
0.00%	irq/388-eth0 (611)	0	[-51]

```
Priorities of the PREEMPTED process during that interval: -2
```

Case studies

Last case: Frequency scaling

- Context: trying to generate a case for our analysis
- More exactly... to break an application workflow
- But... the "should be valid" and "should be invalid" case were having similar durations
- These durations were between 6 ms to 12 ms for each instance
- After a few number of tries... We got it! We forgot to disable frequency scaling.
- ... We could have saved some time.

Frequency scaling analysis

CPU frequency for valid instances:

CPU Frequency	Percent of time
2661.00 MHz	83.09%
2660.00 MHz	11.85%
2527.00 MHz	5.06%

Average frequency: 2654.10 MHz

CPU frequency for invalid instances:

CPU Frequency	Percent of time
1596.00 MHz	39.77%
2527.00 MHz	24.97%
2661.00 MHz	18.36%
Unknown	8.55%
2660.00 MHz	8.36%

Average frequency: 2161.23 MHz

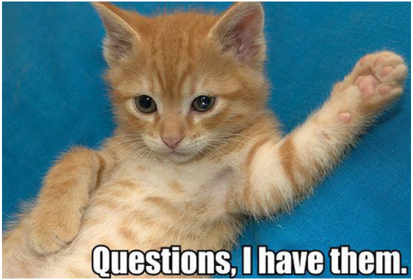
22.81% higher average frequency in valid than in invalid instances.

Verdict: Probability of a frequency scaling problem

Conclusion

- New approach using constraints to automatically detect problems using traces
- Algorithms to do an analysis of the constraints violations
- Presentation of the analysis results of multiple and common real-time problems
- Future work:
  - Write the paper that comes with that!
  - Track 3 of the Ph.D. $\Rightarrow$ from trace to model-based constraints

Thank you.
Any question?



raphael.beamonte@polymtl.ca

Slides:
`secretaire.dorsal.polymtl.ca/∼rbeamonte/dorsal-pm-may2016.pdf`