Tracing and Sampling for Real-Time partially simulated Avionics Systems

May 3, 2013

Raphaël Beamonte Michel Dagenais

Distributed Open Reliable Systems Analysis Lab
Computer and Software Engineering Department

Introduction

- Tracing:
  - Study runtime behavior
  - Can be used to measure latency = fundamental for RT debug
- Tracer requirements:
  - Low-overhead
  - Consistant maximum latency
- Contribution:
  - Methodology and tool to measure real-time latencies (npt)
  - Usage of npt to measure LTTng-UST latency
  - Improvements to the real-time behavior of LTTng
  - Improvements to npt to add other tracer analysis

Table of Contents

1. Research Context
   - Real-Time Operating Systems
   - The Linux Tracing Toolkit next-generation, LTTng
2. Test environment
   - Presentation of the test environment
   - System verification
3. Baseline results
   - The Non-Preempt Test tool
   - Latency results
4. Improving LTTng added latency
   - Identify the source of the latency
   - Latency results and comparison
   - Tracepoint maximum frequency

Real-Time Operating Systems

Why using the Linux kernel ?

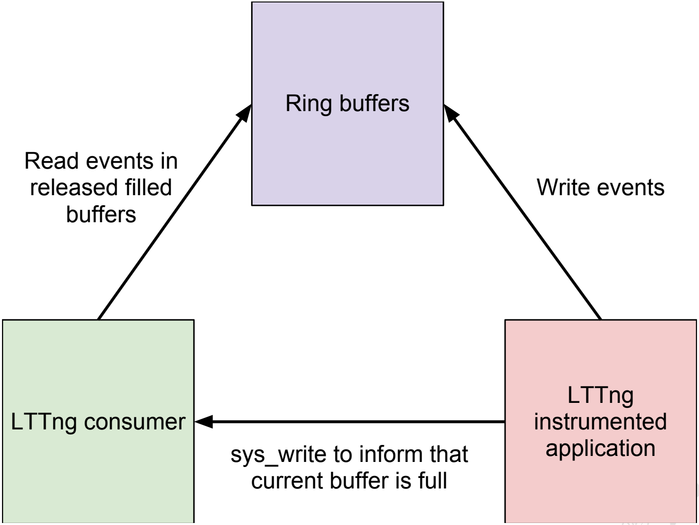
- Able to do Soft Real Time, can reach Hard Real Time :
  - BIOS configuration: would you use hyperthreading ?
  - Kernel configuration: PREEMPT_RT patch, which is more and more integrated to the standard kernel
  - Software configuration: interrupts redirection, cpu shielding...
- The power of the community

The Linux Tracing Toolkit next-generation, LTTng

Why LTTng is pertinent for RT applications ?

- Both userspace and kernel tracers (same clock source)
- Statically compiled tracepoints
- External process to consume events
- Arbitrary event types (Common Trace Format)
- Per-CPU ring buffers
- Important tracing variables protected by RCU

How LTTng-UST consumer works (simplified version)


Ring buffers
Read events in released filled buffers
Write events
LTTng consumer
sys_write to inform that current buffer is full
LTTng instrumented application

Test environment

Hardware:

CPU	Intel® Core™ i7 CPU 920 2.67 GHz
RAM	3 × 2 GiB DDR3 at 1 067 MHz
Motherboard	Intel DX58SO

Kernels:

Standard	debian Linux kernel 3.2.0-4-amd64 package version 3.2.32-1
RT	debian Linux kernel 3.2.0-4-rt-amd64 package version 3.2.32-1

System verification

hwlatdetect (`hwlat_detector`): no hardware latency detected during one hour.

```
hwlatdetect:  test duration 3600 seconds
   parameters:
        Latency threshold: 10us
        Sample window:     1000000us
        Sample width:      500000us
     Non-sampling period:  500000us
        Output File:       None

Starting test
test finished
Max Latency: 0us
Samples recorded: 0
Samples exceeding threshold: 0
```

Why npt ?

- What we have with known tools:
  - `cyclictest`: runs periodic tasks and calculates discrepancy between desired and real period
  - `preempt-test`: verify if higher priority tasks can preempt lower ones

- What we want:
  - A high-priority process that should not stop
  - No latency during the run of this process (no preemption)
  - Ability to add tracepoints easily
  - Ability to add new tracepoints context tests

How npt works ?

- Sets CPU affinity
- Sets RT priority
- Locks process memory into RAM to disable swapping
- Disables local IRQs

- Non-stop loops to calculate statistics with `rdtsc`

- Re-enables local IRQs
- Prints computed statistics

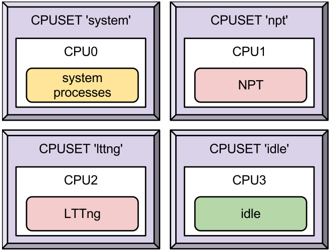
Algorithm of npt's main loop

```
$i \leftarrow 0$
$t_0 \leftarrow$ **read** *rdtsc*
$t_1 \leftarrow t_0$

**while** $i \leq loops\_to\_do$ **do**
    $i \leftarrow i + 1$
    $duration \leftarrow (t_0 - t_1) \times cpuPeriod$

    CalculateStatistics($duration$)
    $t_1 \leftarrow t_0$
    $t_0 \leftarrow$ **read** *rdtsc*
**end while**
```

Algorithm of npt's main loop

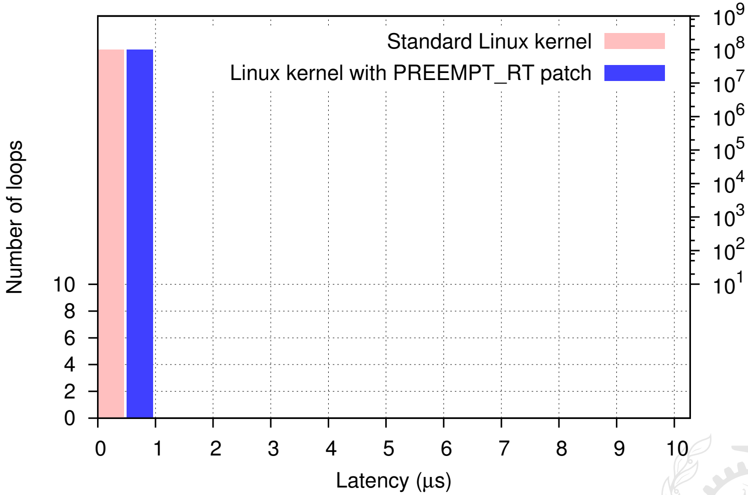
```
$i \leftarrow 0$
$t_0 \leftarrow$ **read** *rdtsc*
$t_1 \leftarrow t_0$
**tracepoint** *nptstart*
**while** $i \leq loops\_to\_do$ **do**
    $i \leftarrow i + 1$
    $duration \leftarrow (t_0 - t_1) \times cpuPeriod$
    **tracepoint** *nptloop*    ▷ Every loop or frequency dependent
    CalculateStatistics($duration$)
    $t_1 \leftarrow t_0$
    $t_0 \leftarrow$ **read** *rdtsc*
**end while**
**tracepoint** *nptstop*
```

The test procedure

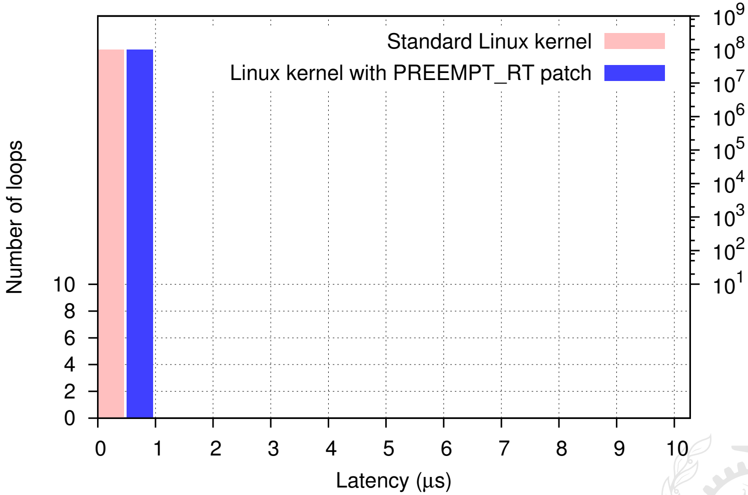


- Shield CPUs (cpusets)
- Run npt for 10^8 loops:
  - Without tracing
  - With LTTng kernel tracing alone
  - With LTTng-UST tracing alone
  - With LTTng-UST and kernel tracing
- Do it on:
  - Standard kernel
  - PREEMPT_RT patched kernel

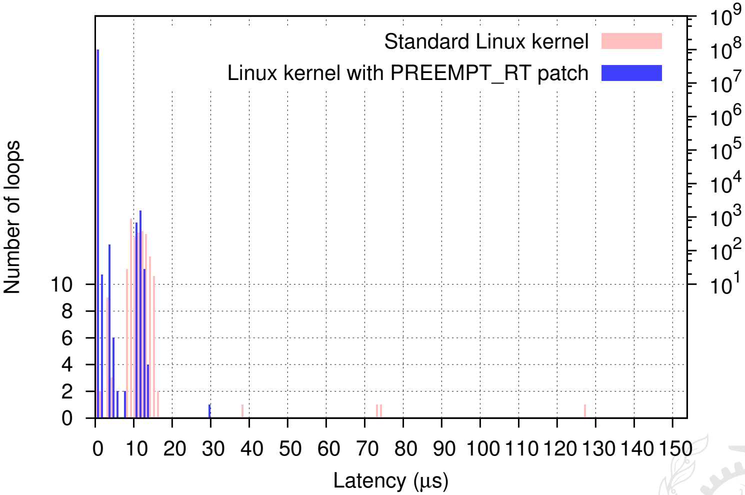
Latency results without tracing



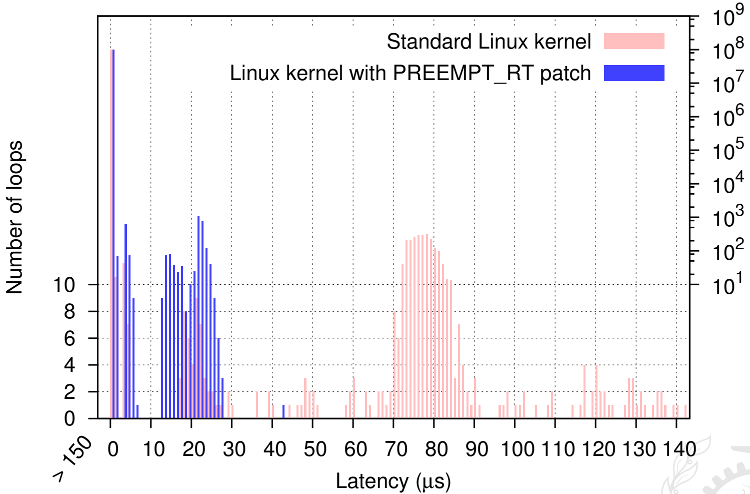
Latency results with LTTng kernel tracing



Latency results with LTTng-UST tracing



Latency results with LTTng-UST and kernel tracing



Identify the source of the latency

Problem

Latency added by the LTTng-UST tracing synchronization

Proposed solution

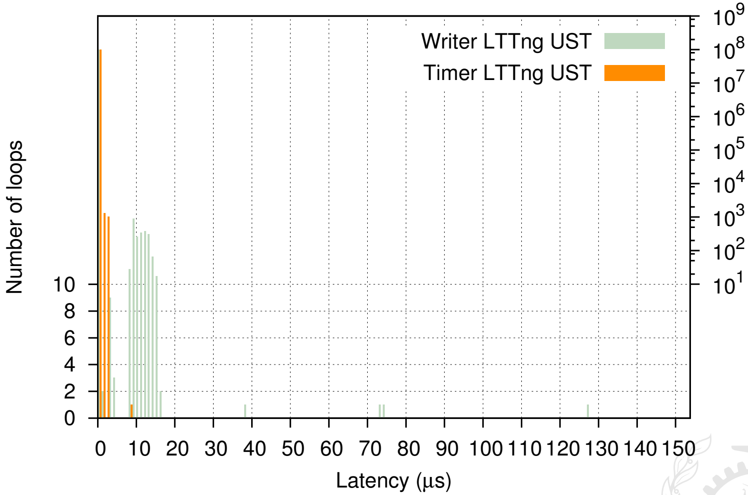
Removing synchronization between instrumented application and LTTng consumer:

- The consumer will now poll to verify if a buffer is full
- Permanent polling (100% CPU use) and `usleep`-timed polling = same performances (CPU shielding)

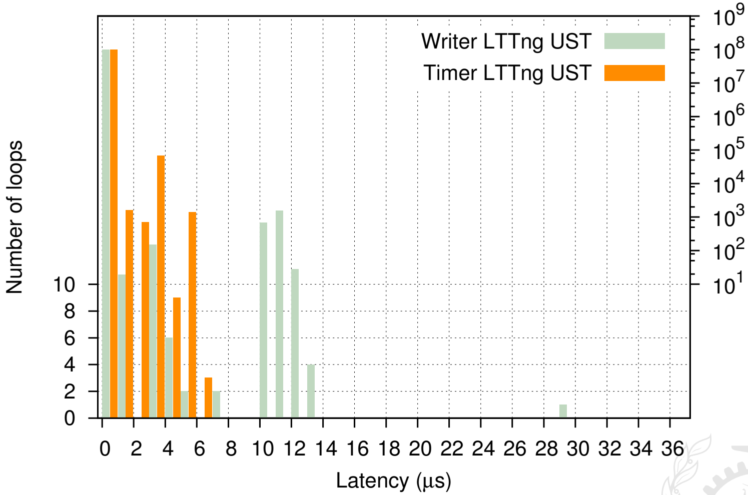
Removing LTTng-UST `getcpu` system call

=> Included in LTTng 2.2 as a new `read-timer` option

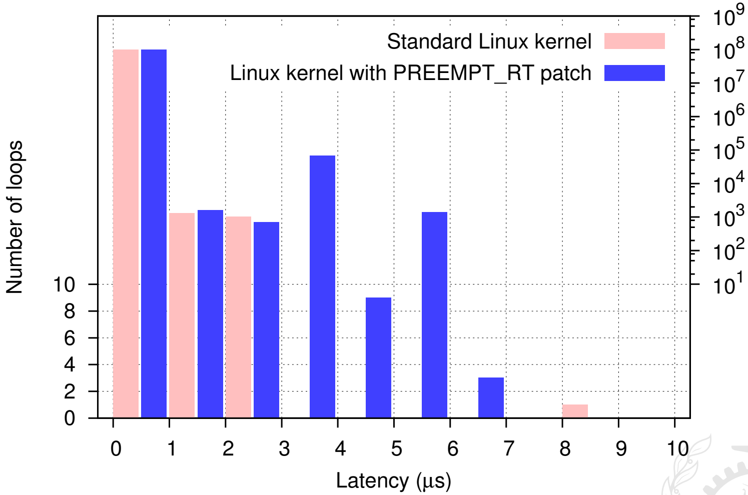
Latency results on the standard kernel



Latency results on the RT kernel



Latency results with modified LTTng-UST

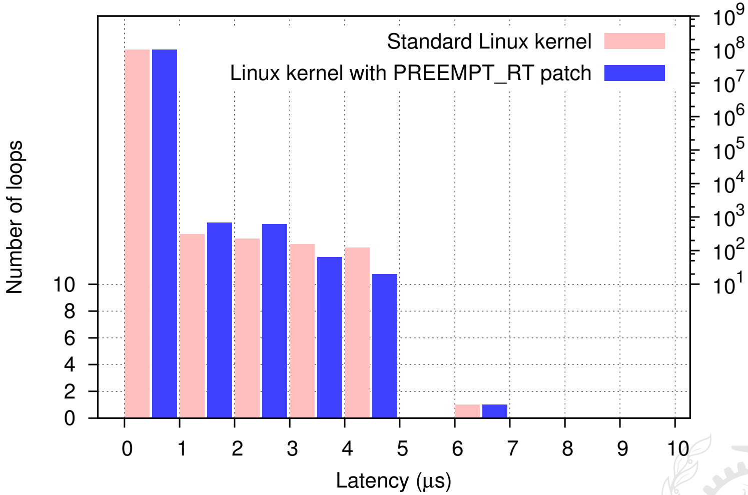


Numeric comparison

Statistics per loops, in nanoseconds, generated by npt on both standard and RT kernels for both the writer and timer versions of LTTng-UST

	Latencies in *ns*			
Kernel	standard		RT	
LTTng 2.2	writer	timer	writer	timer
Minimum	258	458	258	257
Mean	478	538	484	362
Maximum	127 780	8 258	29 999	6 934
Variance	12.071	3.394	3.545	2.002
Deviation	109.869	58.255	59.536	44.742

Latency results with timer LTTng-UST and kernel tracing



Tracing kernel with fast UST events creation

```
...
[warning] Tracer discarded 14893 events between [23:13:14.281175
[warning] Tracer discarded 144973 events between [23:13:14.34974
[warning] Tracer discarded 39160 events between [23:13:14.382843
[warning] Tracer discarded 169643 events between [23:13:14.44459
[warning] Tracer discarded 105019 events between [23:13:14.49214
[warning] Tracer discarded 290003 events between [23:13:14.58030
[warning] Tracer discarded 191738 events between [23:13:14.64690
[warning] Tracer discarded 244662 events between [23:13:14.72511
[warning] Tracer discarded 144658 events between [23:13:14.78175
[warning] Tracer discarded 240612 events between [23:13:14.85946
[warning] Tracer discarded 180970 events between [23:13:14.92408
[warning] Tracer discarded 249067 events between [23:13:15.00367
[warning] Tracer discarded 202268 events between [23:13:15.07256
...
```

Tracing kernel with fast UST events creation

Real-time tracing: what are the “real world” situations ?

- Verifying the run of an application: UST trace only; not any drops
- Understanding the problems of an application: UST and kernel traces; UST drops when we have more data than the buffer size in stressing situations

When can we use LTTng if we want to cross-trace kernel and UST during intensive UST tracepoints creation ?

The npt's tracepoint maximum frequency option

Implementing a new tracepoint maximum frequency option:

- Allow to identify the maximum frequency of tracepoints we can use per second without any drops of events
- For our configuration (32 subbuffers of 1 MB for UST, 32 subbuffers of 4 MB for the kernel trace, `-k -a` with lttng-modules 2.1):
  - ~1 800 000 events per second for the standard kernel
  - ~2 200 000 events per second available for the PREEMPT_RT kernel

Conclusion

- Non-Preempt Test tool
- Effects of LTTng tracing on both standard and RT kernels
- Modified LTTng according to our observations and integrate the changes in the main branch
- Latency is currently as low as 8 μs on standard kernel and 6 μs on the PREEMPT_RT patched one
- Future Work
  - Identifying new real-time tracing usecases to add to npt (i.e. many events during a period of time, no events during another period, and switch between these two periods)
  - Clarify the LTTng real-time limits

Thank you. Any question ?

LTTng `www.lttng.org`
mailing list: `lttng-dev@lttng.org`

npt: `git.dorsal.polymtl.ca/?p=npt.git`

Slides: `www.dorsal.polymtl.ca/~rbeamonte/`
`dorsal-pm-may2013.pdf`