Progress Report Meeting, December 2016

EfficiOS Projects Status Update and Roadmap

jdesfossez@efficios.com
alexandre.montplaisir@efficios.com
jonathan.rajotte-julien@efficios.com
Content

• New LTTng Features (2016),
• Project updates for 2016:
  – LTTng,
  – LTTng-Analyses,
  – Trace Compass,
  – Latency Tracker,
  – Babeltrace,
  – Common Trace Format (CTF) 2.0,
  – Barectf
New LTTng Features (2016)

*LTTng is a low-overhead Linux kernel and user-space tracer*

- Speeding up LTTng-UST (user-space tracer) on ARM32,
- Performance Monitoring Unit counters improvements,
- Linux kernel scheduler thread priority instrumentation for LTTng kernel tracer.
• Speed ups resulting from LTTng-UST profiling
• Propose new kernel system call: restartable sequences (rseq)
  – Expose CPU number through thread-local storage variable rather than system call on ARM32,
  – Expose Restartable Sequences ABI to speed up per-cpu atomic operations. Allows implementing atomic operations on per-cpu data as standard non-atomic operations,
  – Presented at Linux Plumbers Conference Referee Track: http://www.linuxplumbersconf.net/2016/ocw/proposals/3873
Speeding up LTTng-UST

Benchmarks on Cubietruck ARM Cortex A7 @ 1GHz

LTTng stable-2.8 (baseline), minus clock_gettime system call: 2288 ns/event
Adding speed up commit resulting from profiling: 1624 ns/event 1.40:1
Adding use of restartable sequences: 1261 ns/event 1.81:1

- Also speed up LTTng-UST on i7-5600U @2.60GHz x86-64 from ~150ns to 90ns/event,
- Relevant improvements also implemented into LTTng modules kernel tracer.
LTTng Performance Monitoring Unit Counters

- Added support for Performance Monitoring Unit counters for reader from user-space on ARM32,
  - Architectural limitation: requires a system call to read the counter value on ARM32.
- Added custom counter support on all architectures for LTTng-UST and LTTng modules:
  - Specify counter by raw value, associate name from user interface,
  - Useful for architectures with custom-made PMU counters.
Scheduler Thread Priority Instrumentation

- Linux kernel Tracepoints currently expose the “prio” value,
  - Internal scheduler value, should not have been exposed to user-space,
  - Does not convey deadline scheduler information,
  - Missing information at priority changes, only known on the next sched_switch event.
Scheduler Thread Priority Instrumentation

- New instrumentation proposed:
  - Expose Real-Time, Fair, and Deadline schedulers task state:
    - Scheduling policy,
    - Nice value, real-time priority,
    - Deadline scheduler: runtime, deadline, period,
    - Top waiter (priority inheritance).
  - Add missing instrumentation,
- Received feedback from scheduler maintainers, working on updated version.
LTTng Project Update (H2-2016)

• LTTng 2.9 (29-11-2016):
  – Discard mode buffers now available with snapshot tracing (single-shot),
  – New `lttng regenerate statedump` command,
    • Use-case: trigger state dump before taking flight recorder snapshot,
  – Allow override of trace name, path, destination URL when loading a session configuration.
• Titan LTTng-UST CTF logger plugin.
The LTTng analyses are a set of various executable analyses to extract and visualize monitoring data and metrics from LTTng kernel traces on the command line. It models some kernel subsystems to track their state:

- Latency statistics and distributions (IO, Scheduling, IRQ),
- System call statistics,
- IRQ handler duration,
- Top resource users (CPU, memory, ...).
LTTng-Analyses Project Update (H2-2016)

- Added support for nested period analyses: log, frequency distribution, statistics, top,
- Now uses stream intersection mode by default,
LTTng analyses

```
[ljgalar@XThink ~]$ lttng cputop /home/ljgalar/lttng-traces/lttng-analysis-20987-20160820-235617
Checking the trace for lost events...
Processing the trace: 100% ...
```

**Per-TID Usage**

```
<table>
<thead>
<tr>
<th>Process</th>
<th>Migrations</th>
<th>Priorities</th>
</tr>
</thead>
<tbody>
<tr>
<td>watch (16125)</td>
<td>0</td>
<td>[20]</td>
</tr>
<tr>
<td>Xorg (852)</td>
<td>0</td>
<td>[20]</td>
</tr>
<tr>
<td>chromium (15418)</td>
<td>0</td>
<td>[20]</td>
</tr>
<tr>
<td>lttng-sessiod (9343)</td>
<td>0</td>
<td>[20]</td>
</tr>
<tr>
<td>alsa-sink-USB A (999)</td>
<td>0</td>
<td>[-6]</td>
</tr>
<tr>
<td>watch (20771)</td>
<td>0</td>
<td>[20]</td>
</tr>
<tr>
<td>watch (20769)</td>
<td>0</td>
<td>[20]</td>
</tr>
<tr>
<td>watch (20775)</td>
<td>0</td>
<td>[20]</td>
</tr>
<tr>
<td>watch (20777)</td>
<td>0</td>
<td>[20]</td>
</tr>
<tr>
<td>pulseaudio (990)</td>
<td>0</td>
<td>[9]</td>
</tr>
</tbody>
</table>
```

**Per-CPU Usage**

```
<table>
<thead>
<tr>
<th>CPU</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 0</td>
<td>3.77%</td>
</tr>
<tr>
<td>CPU 1</td>
<td>4.44%</td>
</tr>
<tr>
<td>CPU 2</td>
<td>6.42%</td>
</tr>
<tr>
<td>CPU 3</td>
<td>3.44%</td>
</tr>
</tbody>
</table>
```

Total CPU Usage: 4.52%
## LTTng analyses

```plaintext
[ljgalar@Think ~/LinuxCon2016/investigation/simple-trace]$ lttng-syscallstats kernel/
Checking the trace for lost events...
Processing the trace: 100% [aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa]
Per-TID syscall statistics (usec)

<table>
<thead>
<tr>
<th>Watch (16125, TID: 16125)</th>
<th>Count</th>
<th>Min</th>
<th>Average</th>
<th>Max</th>
<th>Stddev</th>
<th>Return values</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>44207</td>
<td>0.138</td>
<td>0.797</td>
<td>4251.704</td>
<td>50.316</td>
<td>'success': 4408</td>
</tr>
<tr>
<td>rt_sigaction</td>
<td>14</td>
<td>0.247</td>
<td>0.489</td>
<td>0.924</td>
<td>0.247</td>
<td>'success': 15</td>
</tr>
<tr>
<td>close</td>
<td>14</td>
<td>0.39</td>
<td>0.61</td>
<td>1.058</td>
<td>0.193</td>
<td>'success': 15</td>
</tr>
<tr>
<td>poll</td>
<td>14</td>
<td>0.536</td>
<td>1.613</td>
<td>3.075</td>
<td>1.036</td>
<td>'success': 15</td>
</tr>
<tr>
<td>pipe</td>
<td>7</td>
<td>5.466</td>
<td>6.542</td>
<td>7.626</td>
<td>0.885</td>
<td>'success': 8</td>
</tr>
<tr>
<td>wait</td>
<td>7</td>
<td>3.255</td>
<td>3.83</td>
<td>5.179</td>
<td>0.542</td>
<td>'success': 8</td>
</tr>
<tr>
<td>fcntl</td>
<td>7</td>
<td>0.519</td>
<td>1.045</td>
<td>1.255</td>
<td>0.25</td>
<td>'success': 8</td>
</tr>
<tr>
<td>clone</td>
<td>7</td>
<td>64.16</td>
<td>69.005</td>
<td>74.077</td>
<td>3.14</td>
<td>'success': 8</td>
</tr>
<tr>
<td>newfstat</td>
<td>7</td>
<td>0.772</td>
<td>0.878</td>
<td>1.195</td>
<td>0.151</td>
<td>'success': 8</td>
</tr>
<tr>
<td>newutent</td>
<td>7</td>
<td>8.248</td>
<td>9.101</td>
<td>10.215</td>
<td>0.67</td>
<td>'success': 8</td>
</tr>
<tr>
<td>rename</td>
<td>7</td>
<td>1.044</td>
<td>1.23</td>
<td>1.517</td>
<td>0.174</td>
<td>'success': 8</td>
</tr>
<tr>
<td>namenode</td>
<td>6</td>
<td>100096.694</td>
<td>100102.787</td>
<td>100107.054</td>
<td>4.424</td>
<td>'success': 7</td>
</tr>
<tr>
<td>Total</td>
<td>44207</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lttng-consumerd (? , TID: 9423)</th>
<th>Count</th>
<th>Min</th>
<th>Average</th>
<th>Max</th>
<th>Stddev</th>
<th>Return values</th>
</tr>
</thead>
<tbody>
<tr>
<td>iocctl</td>
<td>325</td>
<td>0.177</td>
<td>1.234</td>
<td>55.117</td>
<td>3.355</td>
<td>'success': 325</td>
</tr>
<tr>
<td>splice</td>
<td>162</td>
<td>1.982</td>
<td>3.313</td>
<td>12.323</td>
<td>2.013</td>
<td>'success': 163</td>
</tr>
<tr>
<td>sync_file_range</td>
<td>162</td>
<td>8.040</td>
<td>30.1</td>
<td>512.694</td>
<td>42.733</td>
<td>'success': 163</td>
</tr>
<tr>
<td>fadvise6</td>
<td>81</td>
<td>1.889</td>
<td>2.676</td>
<td>24.711</td>
<td>2.783</td>
<td>'success': 82</td>
</tr>
<tr>
<td>Total</td>
<td>730</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>thread-ml (? , TID: 7926)</th>
<th>Count</th>
<th>Min</th>
<th>Average</th>
<th>Max</th>
<th>Stddev</th>
<th>Return values</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>162</td>
<td>0.24</td>
<td>1.971</td>
<td>17.799</td>
<td>2.599</td>
<td>'success': 113</td>
</tr>
<tr>
<td>write</td>
<td>97</td>
<td>0.456</td>
<td>2.508</td>
<td>9.372</td>
<td>1.444</td>
<td>'success': 98</td>
</tr>
<tr>
<td>fcntl</td>
<td>93</td>
<td>0.178</td>
<td>0.989</td>
<td>1.799</td>
<td>0.433</td>
<td>'success': 94</td>
</tr>
<tr>
<td>poll</td>
<td>49</td>
<td>186.732</td>
<td>1449.45</td>
<td>26299.653</td>
<td>5428.461</td>
<td>'success': 550</td>
</tr>
<tr>
<td>select</td>
<td>31</td>
<td>3.987</td>
<td>4.924</td>
<td>8.53</td>
<td>0.572</td>
<td>'success': 35</td>
</tr>
<tr>
<td>iocctl</td>
<td>31</td>
<td>0.781</td>
<td>1.1</td>
<td>3.642</td>
<td>0.49</td>
<td>'success': 32</td>
</tr>
<tr>
<td>Total</td>
<td>453</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
LTtng analyses

[jgalar@XThink ~]$ lttng-iolatencyfreq kernel/
Checking the trace for lost events...
Processing the trace: 100%

I/O request latency distribution open (us)

---

EfficiOS
Custom Period

$ lttng-periodlog --period 'job_len: $evt.$name == "job_start" : $evt.$name == "job_end"
--period 'fetch(job_len): $evt.$name == "job_start" : $evt.$name == "fetch_end"
[...]
• Extract statistics, log, top, frequency distributions of period durations
• Allow to identify the longest periods (high latency)
• Keep relationship and data between child/parent period definitions
• Group statistics and frequency distributions based on payload value(s)
$ lttng-periodlog
   --period ‘[ NAME [ (PARENT) ] ] : BEGIN_EXPR [ : END_EXPR ]’
   --period-capture ‘NAME : BEGINCAPTURES [ : ENDCAPTURES ]’
   --select PERIODS
   --aggregate-by PERIOD
   --group-by FIELD
sched_switch: { cpu_id = 1 }, { prev_comm = "swapper/1", prev_tid = 0,
    prev_prio = 20, prev_state = 0,
    next_comm = "bash", next_tid = 12421, next_prio = 20 }

syscall_entry_open: { cpu_id = 1 }, { filename = "/etc/ld.so.cache",
    flags = 524288, mode = 1 }

kmem_cache_alloc: { cpu_id = 1 }, { call_site = 0xFFFFF88037BF67000,
    ptr = 0xFFFF88037BF67000, bytes_req = 4096, bytes_alloc = 4096,
    gfp_flags = 208 }

kmem_cache_free: { cpu_id = 1 }, { call_site = 0xFFFFF88037BF67000 }

syscall_exit_open: { cpu_id = 1 }, { ret = 3 }

sched_switch: { cpu_id = 1 }, { prev_comm = "lttng", prev_tid = 12421,
    prev_prio = 20, prev_state = 1, next_comm = "swapper/1",
    next_tid = 0, next_prio = 20 }

Delay between the 2 sched_switch: 3.6ms
Files opened: 22
Events generated during that period: 1570
$ ./lttng-periodlog /path/to/trace
   --period 'switch :
       $evt.$name == "sched_switch" :
       $evt.$name == "sched_switch" &&
       $begin.$evt.next_tid == $evt.prev_tid &&
       $begin.$evt.cpu_id == $evt.cpu_id' \ 
   --period 'open(switch) :
       $evt.$name == "syscall_entry_open" &&
       $parent.$begin.$evt.cpu_id == $evt.cpu_id :
       $evt.$name == "syscall_exit_open" &&
       $begin.$evt.cpu_id == $evt.cpu_id' \ 
   --period 'alloc(open) :
       $evt.$name == "kmem_cache_alloc" &&
       $parent.$begin.$evt.cpu_id == $evt.cpu_id :
       $evt.$name == "kmem_cache_free" && $evt.ptr == $begin.$evt.ptr' \ 
   --period-captures 'switch : comm = $evt.next_comm, cpu = $evt.cpu_id, tid = $evt.next_tid' \ 
   --period-captures 'open : filename = $evt.filename : fd = $evt.ret' \ 
   --period-captures 'alloc : ptr = $evt.ptr' \ 
   --select "open alloc" \
Trace Compass Project Update (H2-2016)

- *Eclipse Trace Compass provides views, graphs, metrics, and more to help extract useful information from traces.*

- Speed up single-stepping of kernel events when following one thread,
- Integration of LTTng-Analyses machine interface,
- Pin & New View features (proposed upstream),
- Control Flow View dynamic filter on active threads with CPU set selection (proposed upstream),
- Stream intersection mode,
- Trace cut feature.
LTTng analyses - Trace Compass Integration

- Invoke custom analyses
- LAMI 1.0
  - Open Specification
  - JSON based
Scheduling Latencies

EffectOS
The Latency Tracker is a kernel module performing statistical latency trend aggregation, and identification of outliers. It can trigger user-configurable actions such as recording a flight recorder snapshot when outliers are detected,

- Track work begin/end with identifiers from instrumented user-space,
- Time-to-first-byte tracker.
Available Latency Trackers

- Block layer: from block request issue to completion,
- Network: from socket buffer receive to consumption by user-space,
- Wake-up: from each thread wake-up to next scheduling of that thread,
- Off-cpu: from each thread preemption/blocking to next execution of that thread,
- IRQ handler: from irq handler entry to exit,
- System call: from system call entry to exit,
- Time-to-first-byte: from accept system call return to write system call family entry on the same inode,
- Online critical path analysis: from interrupt servicing to completion of task.
The Babeltrace project provides a library, Python bindings, as well as a command-line tool to view and convert traces. It is a reference implementation of the Common Trace Format (CTF).

- Babeltrace 1.4 (06-2016)
  - Mapping events to C/C++ source code (DWARF debug info, ELF),
  - Stream intersection mode (for LTTng snapshots),
  - Lost packet reporting.
- Babeltrace 1.5 (12-2016)
  - Expose APIs required by Perf to CTF converter.
Babeltrace Project Update (2017)

- Babeltrace 2.0 planned in January 2017
  - Trace Intermediate Representation,
  - Modular source/filter/sink architecture,
  - Plugin architecture,
  - C/C++/Python APIs,
  - Allows analyses to read live traces,
  - CTF 1.8 source/sink (reader/writer),
  - Trace cut feature,
  - Multi-clock support (e.g. Epoch time and BFN clock).
Babeltrace Project Update (2017)

- Babeltrace 2.1 (2017)
  - Event filtering,
  - CTF 2.0.
The Common Trace Format (CTF) is a binary trace format designed to be very fast to write without compromising great flexibility. It allows traces to be natively generated by any C/C++ application or system, as well as by bare-metal (hardware) components.

- Main change in CTF 2 is to move from custom metadata language to JSON, for flexibility and extensibility purposes,
- CTF 2 proposal document sent for comments on lttng-dev and diamon-discuss mailing lists.
barectf is a command-line generator of ANSI C tracers which output Common Trace Format packets natively.

- Demo of instrumented Parallella bare-metal application, with flight recorder snapshots, and custom Trace Compass view.
Links

LTTng: http://lttng.org

LTTng analyses scripts: https://github.com/lttng/lttng-analyses

Latency tracker: https://github.com/efficios/latency-tracker

barectf: https://github.com/efficios/barectf

TraceCompass: http://tracecompass.org/

Babeltrace http://diamon.org/babeltrace

Common Trace Format http://diamon.org/ctf
Trace Compass Demos

- Dynamic filters
  - Active Threads
  - Per CPU filtering
- Pin & Clone of views
- Trace cutting
Questions ?

EfficiOS

www.efficios.com

lttng.org

lttng-dev@lists.lttng.org

@lttng_project