Benchmarking and comparison of kernel and userspace tracers

Mohamad Gebai
Michel Dagenais

5 May, 2016
École Polytechnique de Montreal
Content

- The tracers
- The setup
- The benchmarks
- Results
  - Kernel space
  - System calls
  - Filtering
  - Userspace
The tracers

- **Kernel tracers**
  - LTTng
  - Ftrace
  - Perf
  - eBPF (kind of)

- **Userspace tracers**
  - LTTng
  - Printf()
  - LTTng using tracef()
  - Extrae
  - Lightweight homemade basic tracer
The setup

- Intel i7 @ 3.40 GHz
- 16 GB of RAM
- Linux kernel version 4.5.0
- LTTng suite built from tip of the branch stabe-2.7
- Deactivated:
  - Hyperthreading
  - C-states
  - CPU idle
  - Intel Turbo Boost
- LTTng
  - Snapshot mode
  - Subbuffer size 32 KB
- Ftrace
  - Global clock
The benchmarks

- Objective: measure the cost of writing a tracepoint
- Kernel tracers:
  - Microbenchmark in kernel space
  - Payload: 32 bytes
- Userspace tracers:
  - Microbenchmark in userspace
  - Payload: 32 bytes
- Steady state (drop initial page faults, cache misses, initialization, etc.)
  - Do a full buffer write and start overwriting
**Kernel tracing: the outline**

- **Final numbers:**

<table>
<thead>
<tr>
<th></th>
<th>Average</th>
<th>90th percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Value (ns)</td>
<td>Overhead</td>
</tr>
<tr>
<td>None</td>
<td>17</td>
<td>0%</td>
</tr>
<tr>
<td>LTTng</td>
<td>92</td>
<td>441%</td>
</tr>
<tr>
<td>Ftrace</td>
<td>118</td>
<td>594%</td>
</tr>
<tr>
<td>Perf</td>
<td>120</td>
<td>606%</td>
</tr>
<tr>
<td>LTTng-kprobe</td>
<td>122</td>
<td>618%</td>
</tr>
<tr>
<td>eBPF</td>
<td>131</td>
<td>671%</td>
</tr>
</tbody>
</table>

![Average latency of a tracepoint](image-url)
Kernel tracing: the outline

- Final numbers:

![Diagram showing average tracepoint latency in steady vs transient states for different kernel tracers: None, LTTng, Ftrace, Perf.](image)

Benchmarking and comparison of kernel and userspace tracers
Kernel tracing: LTTng vs Ftrace

Benchmarking and comparison of kernel and userspace tracers
Kernel tracing: closer look

Benchmarking and comparison of kernel and userspace tracers
Kernel tracing: closer look
Kernel tracing: ioctl() syscall

- Final numbers:

<table>
<thead>
<tr>
<th></th>
<th>Average</th>
<th>90th percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Value (ns)</td>
<td>Overhead</td>
</tr>
<tr>
<td>None</td>
<td>99</td>
<td>0%</td>
</tr>
<tr>
<td>Ftrace</td>
<td>311</td>
<td>214%</td>
</tr>
<tr>
<td>LTTng</td>
<td>341</td>
<td>244%</td>
</tr>
<tr>
<td>Perf</td>
<td>600</td>
<td>506%</td>
</tr>
</tbody>
</table>

Average latency of a syscall with tracing

Benchmarking and comparison of kernel and userspace tracers
Kernel tracing: open() syscall

Latency of tracing syscall open() according to filename length

Filename length in bytes

Latency in ns

Benchmarking and comparison of kernel and userspace tracers
Benchmarking and comparison of kernel and userspace tracers

**Kernel tracing: filtering with LTTng**

![Graph showing average latency of a tracepoint with filtering](image)

- **Failed conditions**
- **Average latency on success**
- **No filter**

**Average latency of a tracepoint with filtering**

- **Latency in ns**
  - 0
  - 50
  - 100
  - 150
  - 200
  - 250
  - 300
  - 350
  - 400
  - 450

- **Filter bytecode size (number of conditions)**
  - 0
  - 10
  - 20
  - 30
  - 40
  - 50
  - 60

---

**Graphic Descriptions**

- The graph illustrates the average latency of a tracepoint with filtering under different conditions.
- **Failed conditions** exhibit a higher latency compared to **average latency on success**.
- The **no filter** condition shows the least latency.

**Analysis**

- Filtering with LTTng incurs latency that increases with the number of conditions.
- Filtering improves tracepoint selection but at the cost of increased latency.
Userspace tracing: the outline

- Final numbers, but...
- LTTng is signal-safe reentrant
- LTTng is scalable

<table>
<thead>
<tr>
<th></th>
<th>Average</th>
<th></th>
<th>90th percentile</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Value (ns)</td>
<td>Overhead</td>
<td>Value (ns)</td>
<td>Overhead</td>
</tr>
<tr>
<td>None</td>
<td>18</td>
<td>0%</td>
<td>17</td>
<td>0%</td>
</tr>
<tr>
<td>LW-ust</td>
<td>67</td>
<td>272%</td>
<td>64</td>
<td>276%</td>
</tr>
<tr>
<td>Extrae</td>
<td>159</td>
<td>783%</td>
<td>118</td>
<td>594%</td>
</tr>
<tr>
<td>Printf</td>
<td>245</td>
<td>1261%</td>
<td>243</td>
<td>1329%</td>
</tr>
<tr>
<td>LTTng</td>
<td>255</td>
<td>1317%</td>
<td>249</td>
<td>1365%</td>
</tr>
<tr>
<td>LTTng tracef()</td>
<td>536</td>
<td>2878%</td>
<td>523</td>
<td>2976%</td>
</tr>
</tbody>
</table>

Average latency of a tracepoint

Benchmarking and comparison
Userspace tracing: closer look
benchmarking and comparison of kernel and userspace tracers

userspace tracing: scalability
Related work

- More scalability runs
- Closer look at aggregators (eBPF, Systemtap)
Acknowledgements

- Professor Michel Dagenais
- Julien Desfossez
- Mathieu Desnoyers
- Ericsson