LTTng: from Low-Level Tracing to High-Level Analyses
Content

- LTTng
- Babeltrace
- Common Trace Format
- LTTng analyses
- Latency tracker
- Histogram generator
- TraceCompass
LTTng

- Performs user-space and kernel tracing,
- Kernel tracing performed by out-of-tree module
  - **No kernel patching required**
  - Supports kernel from 2.6.38 to 4.2+
- Enables seamless analysis of correlated kernel and user-space data.
- Flexible and fast.
What's new in LTTng?

- LTTng 2.6 (01/2015)
  - Java Log4j support,
  - Kernel tracer per system call tracing,
  - Kernel tracer system call content (input/output) fetching,
  - Tracing NMI handlers (with Linux 3.17 or better).
  - LTTng MI (Machine Interface),
What's new in LTTng?

- LTTng 2.7 (currently in RC)
  - Persistent memory UST ring buffer
    - pramfs (out of tree), or
    - DAX (Linux 4.0) and pmem driver (upcoming Linux 4.1)
    - Either BIOS does not reset memory on soft reboot, or use kexec(8)
    - Allows recovering user-space traced when system crash with new lttng-crash tool.
What's new in LTTng?

- LTTng 2.7 (currently in RC)
  - LTTng filtering for kernel domain,
  - Per-process user-space and kernel tracing,
    - Select a set of PIDs
  - Wildcards for kernel tracepoints,
  - LTTng modules clock plugin support,
  - LTTng UST clock and getcpu plugin support,
  - LTTng Python logger support.
Babeltrace

• Babeltrace 1.x
  – CTF reader
  – Merge CTF traces by timestamp,
  – Supports live LTTng tracing,
  – C, C++, Python APIs.

• Babeltrace 2.0 (approx. 10/2015)
  – Plugin system overhaul,
  – Intermediate Representation,
  – Event filtering.
Common Trace Format (CTF)

- Currently working on CTF 2.0
- Goal: transition from own metadata grammar (TSDL) to JSON.
- Will be easier to extend, and easier to parse by alternative CTF reader implementations.
- Specification of CTF 1.8 available at http://diamon.org/ctf
LTTSng Analyses

• Set of Python scripts providing summarized trace information,
• Each analysis typically classified as:
  – Top N
  – Statistics table (avg., std. dev, min, max)
  – Frequency histogram
• Available at https://github.com/lttng/lttng-analyses
Available Analyses

- CPU usage for the whole system
- CPU usage per-process
- Process CPU migration count
- Memory usage per-process (as seen by the kernel)
- Memory usage system-wide (as seen by the kernel)
- I/O usage (syscalls, disk, network)
- I/O operations log (with latency and usage)
- I/O latency statistics (open, read, write, sync operations)
- I/O latency frequency distribution
- Interrupt handler duration statistics (count, min, max, average stdev)
- Interrupt handler duration top
- Interrupt handler duration log
- Interrupt handler duration frequency distribution
- SoftIRQ handler latency statistics
- Syscalls usage statistics
LTTng Analyses (live demo)
Latency-tracker

- Kernel module to track down latency problems at run-time
- Simple API that can be called from anywhere in the kernel (tracepoints, kprobes, netfilter hooks, hardcoded in other module or the kernel tree source code)
- Keep track of entry/exit events and calls a callback if the delay between the two events is higher than a threshold
tracker = latency_tracker_create();

latency_tracker_event_in(tracker, key, threshold, timeout, callback);

....

latency_tracker_event_out(tracker, key);

If the delay between the event_in and event_out for the same key is higher than “threshold”, the callback function is called.

The timeout parameter allows to launch the callback if the event_out takes too long to arrive (off-CPU profiling).
Implemented use-cases

- Block layer latency
  - Delay between block request issue and complete
- Wake-up latency
  - Delay between sched_wakeup and sched_switch
- Network latency (prototype)
- IRQ handler latency (prototype)
- System call latency
  - Delay between the entry and exit of a system call
- Offcpu latency
  - How long a process has been scheduled out and why did it get woken up
Example: system call latency

- Developed in collaboration with François Doray

```c
on syscall_entry:
    latency_tracker_event_in(current_pid);

on syscall_exit:
    latency_tracker_event_out(current_pid);

on sched_switch:
    event = latency_tracker_get_event(next_pid);
    if event && ((now - event->start) > threshold):
        dump_stack(next_pid);
```
System call latency example

syscall_latency_stack: comm=sync, pid=32224

81136.460929
schedule
schedule_timeout
wait_for_completion
sync_inodes_sb
sync_inodes_one_sb
iterate_supers
sys_sync
tracesys

81136.461482
_cond_resched
sync_inodes_sb
sync_inodes_one_sb
iterate_supers
sys_sync
tracesys

81136.467357
_cond_resched
mempool_alloc
__split_and_process_bio
dm_request
generic_make_request
submit_bio
submit_bio_wait
blkdev_issue_flush
ext4_sync_fs
sync_fs_one_sb

81136.470176
schedule
schedule_timeout
wait_for_completion
submit_bio_wait
blkdev_issue_flush
ext4_sync_fs
sync_fs_one_sb
iterate_supers
sys_sync
tracesys

Dynamically change the threshold:
# echo 1000000 > /sys/module/latency_tracker_syscalls/parameters/usec_threshold

EfficiOS
Off-cpu profiling

```python
on sched_switch(prev, next):
    latency_tracker_event_in(prev, cb)
    latency_tracker_event_out(next)

cb():
    dump_stack(pid)

on sched_wakeup(pid):
    event = latency_tracker_get_event(pid)
    if event && ((now - event->start) > threshold):
        dump_stack(current)
```
Off-cpu profiling example

offcpu_sched_wakeup:
  waker_comm=swapper/3 (0),
  wakee_comm=qemu-system-x86 (7726),
  wakee_offcpu_delay=1000018451,
  waker_stack=
    ttwu_do_wakeup
  ttwu_do_activate.constprop.74
    try_to_wake_up
    wake_up_process
    hrtimer_wakeup
    __run_hrtimer
    hrtimer_interrupt
  local_apic_timer_interrupt
  smp_apic_timer_interrupt
  apic_timer_interrupt

offcpu_sched_switch:
  comm=qemu-system-x86,
  pid=7726,
  delay=1000140896,
  stack=
    schedule
    futex_wait_queue_me
    futex_wait
    do_futex
    SyS_futex
    system_call_fastpath
Runtime latency distributions

- For system calls, file system, I/O scheduler and block requests
- Show the distribution of request latencies
- Clearly see in one screen the latencies of all disk I/O at various level
- Available at https://github.com/jdesfossez/latency_tracker
- Video demo (demo-latency_tracker.ogv)
## Overhead on sysbench oltp (MySQL)

<table>
<thead>
<tr>
<th>Test</th>
<th>Average</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>63.26s</td>
<td></td>
</tr>
<tr>
<td>LTTng sched</td>
<td>63.65s</td>
<td>0.61%</td>
</tr>
<tr>
<td>LTTng syscalls</td>
<td>64.95s</td>
<td>2.66%</td>
</tr>
<tr>
<td>Latency_tracker</td>
<td>65.36s</td>
<td>3.31%</td>
</tr>
<tr>
<td>Latencytop</td>
<td>66.24s</td>
<td>4.70%</td>
</tr>
<tr>
<td>LTTng all</td>
<td>70.24s</td>
<td>11%</td>
</tr>
</tbody>
</table>
TraceCompass

• Now available as a standalone application (requires only a Java Virtual Machine)
• Available at http://tracecompass.org
• We are currently working at facilitating workflows involving frequent back-and-forth between LTTng analyses and TraceCompass,
• Can now read Perf traces converted to CTF.
TraceCompass Screenshot

EfficiOS
Questions?

EfficiOS

www.efficios.com

lttng.org

lttng-dev@lists.lttng.org

@lttng_project