Large-scale performance monitoring framework for cloud monitoring

Run-Time Latency Detection in Production

Julien Desfossez
Michel Dagenais

Décembre 2014
École Polytechnique de Montreal

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

Latency tracker previous state

- Prototype working and stable
- Needed more testing use-cases
- Performance measurements was in progress
- Hashtable needed scaling optimization

Usage

```
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).

New feature: check current state

- It is now possible to query the current state of a request without removing the key :

```
event = latency_tracker_get_event(tracker, key);

latency_tracker_put_event(event);
```

- Stateful tracing
- Refcount-based ownership

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
- IRQ latency
- System call latency
  - Delay between the entry and exit of a system call
- Offcpu latency
  - How long a process has been scheduled out

System call latency

- Developed in collaboration with François Doray

```
on syscall_entry:
  latency_tracker_event_in(current_pid);
on syscall_exit:
  latency_tracker_event_out(current_pid);
```

System call latency

- Developed in collaboration with François Doray

```
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

Off-cpu profiling

```
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

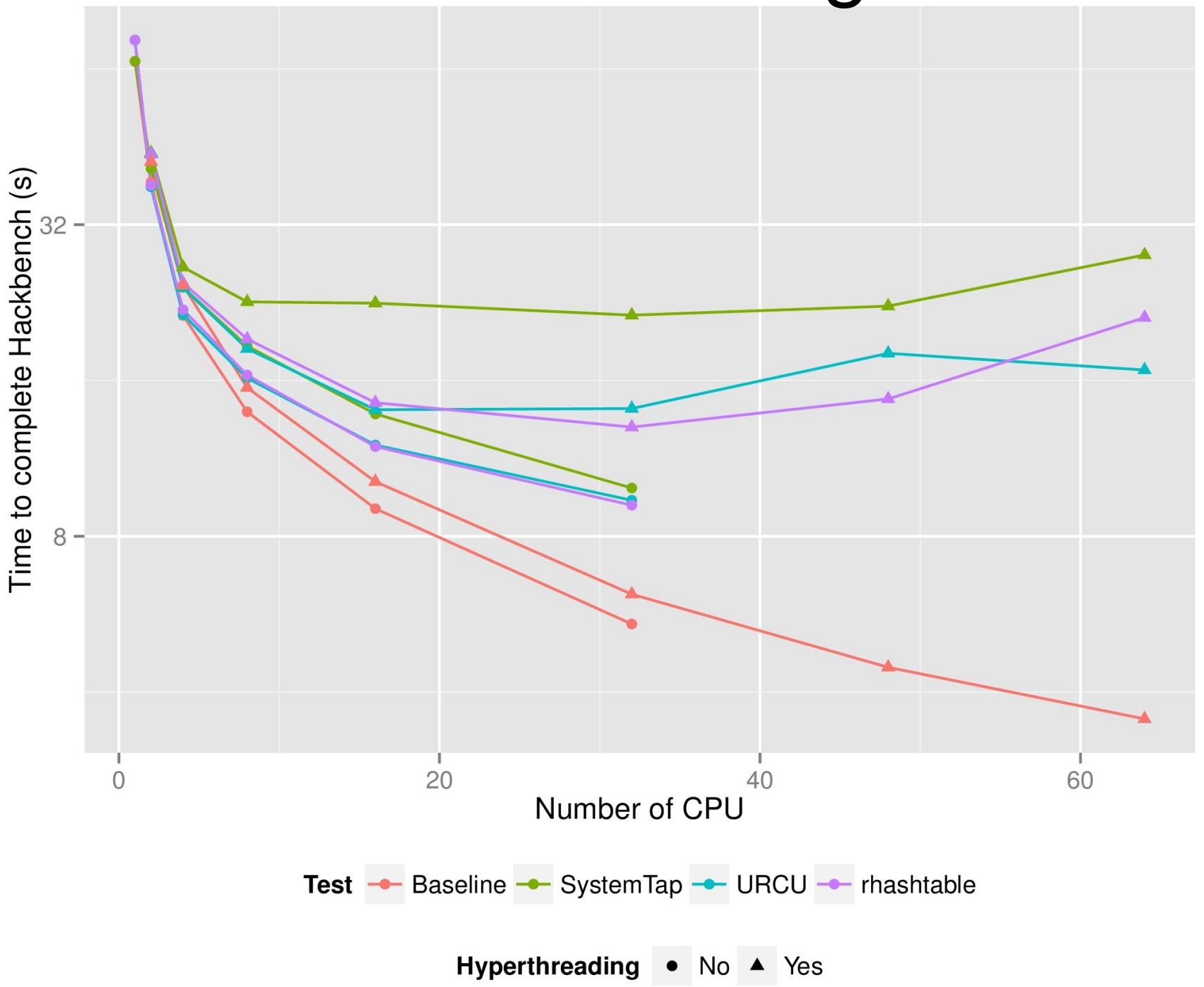
```
offcpu_sched_wakeup:
  waker_comm=swapper/3 (0),
  wakee_comm=qemu-system-x86 (7726),
  wakee_offcpu_delay=10000018451,
  waker_stack=
    ttwu_do_wakeup
ttwu_do_activate.constprop.74
    try_to_wake_up
    wake_up_process
    hrtimer_wakeup
    __run_hrtimer
    hrtimer_interrupt
  local_apic_timer_interrupt
    smp_apic_timer_interrupt
    apic_timer_interrupt
```

```
offcpu_sched_switch:
  comm=qemu-system-x86,
  pid=7726,
  delay=10000140896,
  stack=
    schedule
    futex_wait_queue_me
    futex_wait
    do_futex
    SyS_futex
    system_call_fastpath
```

Performance improvements

- Controlled memory allocation
- Lock-less free-list
- Out-of-context reallocation of memory if needed/enabled
- Now using userspace-rcu hashtable for lock-less insert and lookup (ported to the kernel by Mathieu Desnoyers: KURCU ?)
- Custom call_rcu thread to avoid the variable side-effects of the built-in one

CPU scaling


Time to complete Hackbench (s)
32
8
0
20
40
60
Number of CPU
Test
Baseline
SystemTap
URCU
rhashtable
Hyperthreading
No
Yes

Overhead on sysbench oltp (MySQL)

Test	Average	Overhead
Baseline	63.26s	
LTTng sched	63.65s	0.61%
LTTng syscalls	64.95s	2.66%
Latency_tracker	**65.36s**	**3.31%**
Latencytop	66.24s	4.70%
LTTng all	70.24s	11%

Future Work

- Keep internal state of the current latency profile (last minute, last 5 minutes, last hours)
- Extract aggregated information about latencies as histograms
- Compare evolutions of latencies and major changes
- Analyse large data set of high-latency events to help create and understand latency profiles

Install it

```
apt-get install git gcc make
linux-headers-generic

git clone
https://github.com/jdesfossez/late
ncy_tracker.git

cd latency_tracker

make
```

Questions ?