Host & Guest Tracing in Virtualization: "To sync, or not to sync?"

Stefano De Venuto  
SUSE, Intern  

Tzvetomir Stoyanov  
VMware Open Source Technology Center
Who we are

**Stefano De Venuto**
Computer Science Student at University of Turin
Intern at SUSE
• e-mail: stefano.devenuto99@gmail.com
• github: https://github.com/stefanodevenuto/

**Tzvetomir Stoyanov**
Software engineer in the Open Source Technology Center,
VMware/Bulgaria working on the Linux Kernel tracing infrastructure.
• e-mail: tz.stoyanov@gmail.com
• github: https://github.com/tzstoyanov
Ftrace

- The official tracer of the Linux kernel
- Developed by Steven Rostedt more than 10 years ago
- Part of the kernel, compiled by default in most popular Linux distros.
- Allows you to look inside every corner of a live running kernel.
Host-Guest kernel tracing

Challenges

- Fast transfer of huge tracing data between guest and host
- Time stamps synchronisation
## Trace data transfer

<table>
<thead>
<tr>
<th>Channel</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFOs</td>
<td>1000 MB/s</td>
</tr>
<tr>
<td>vsockets</td>
<td>900 MB/s</td>
</tr>
<tr>
<td>TCP/IP sockets</td>
<td>275 MB/s</td>
</tr>
</tbody>
</table>

Measured on a laptop with Intel i5 CPU with 8 cores and 16G RAM
Time stamps synchronisation - PTP

- Clock offset: 
  \[
  \frac{(T1' - T1 - T2' + T2)}{2}
  \]

- Round trip time is not symmetric
- No hardware timestamping
- Up to few hundred packets are exchanged in one clock offset measurement
- ftrace is used to get the packet times
Time stamps synchronisation - KVM

/sys/kernel/debug/kvm:
- tsc-offset
- tsc-scaling-ratio
- tsc-scaling-ratio-frac-bits

guest_tsc = tsc-offset + (host_tsc * tsc-scaling-ratio) >> tsc-scaling-ratio-frac-bits
Tools & Libraries

- trace-cmd 2.9
- KernelShark 2.0
- trace libraries
  - librtaceevent
  - libtracefs
  - libtracecmd
  - libkshark
Verifying the results

- Given a combined (host + guest[s]) trace, how good did the merging and the timestamp synchronization algorithm of choice work?

- Two different aspects:
  - event stream validation
  - timestamp synchronization accuracy evaluation
Validation: kvm_entry & kvm_exit events

Two crucial events:

- **kvm_entry**: marks when a CPU starts executing instructions from the guest
- **kvm_exit**: marks when the CPU stops executing instructions from the guest

What we expect:

- **only** guest events between a kvm_entry and the next kvm_exit
- **no** guest events between a kvm_exit and the next kvm_entry
Validation: what we expect?

[Image of a graphical user interface showing a timeline and some text in a table format.]

- CPU 0
- CPU SKVM-1977

<table>
<thead>
<tr>
<th>#</th>
<th>CPU</th>
<th>Time Stamp</th>
<th>Task</th>
<th>PID</th>
<th>Latency</th>
<th>Event</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>182</td>
<td>11241.65251</td>
<td>CPU 0/KVM</td>
<td>1977</td>
<td>d...</td>
<td>kvm/kvm_entry</td>
<td>vcpu 0, rip 0xfffffffffa079246</td>
</tr>
<tr>
<td>1</td>
<td>184</td>
<td>11241.65251</td>
<td>&lt;idle&gt;</td>
<td>0</td>
<td>d.h.</td>
<td>timer/hrtimer_expire_exit</td>
<td>hrtimer=0xffffffff0127dc21540</td>
</tr>
<tr>
<td>1</td>
<td>185</td>
<td>11241.65251</td>
<td>&lt;idle&gt;</td>
<td>0</td>
<td>d.h.</td>
<td>timer/hrtimer_start</td>
<td>hrtimer=0xffffffff0127dc21540 function=tick_sched_timer/0x0 expires=3616998901864 softexpires=3616998901864</td>
</tr>
<tr>
<td>1</td>
<td>186</td>
<td>11241.65251</td>
<td>&lt;idle&gt;</td>
<td>0</td>
<td>d.h.</td>
<td>msr/write_msr</td>
<td>6e0, value 9e5fd35bc24</td>
</tr>
<tr>
<td>1</td>
<td>187</td>
<td>11241.65253</td>
<td>&lt;idle&gt;</td>
<td>0</td>
<td>...</td>
<td>power/cpu_idle</td>
<td>state=4294967295 cpu_id=0</td>
</tr>
<tr>
<td>0</td>
<td>190</td>
<td>11241.65253</td>
<td>CPU 0/KVM</td>
<td>1977</td>
<td>d...</td>
<td>kvm/kvm_exit</td>
<td>reason HLT rip 0xffffffffaaaa6917d info 0 0</td>
</tr>
</tbody>
</table>
Validation: what we expect?
Validation: what we expect?
Time-sync accuracy evaluation: an “event-based” approach

- **Set(s) of consequential events:**
  - event in the guest that causes one or more events on the host
  - event on the host that causes one or more events in the guest

  *G: XXX - guest event*
  *H: YYY - host event, consequence of XXX*
  *H: YYY - host event*
  *G: XXX - guest event, consequence of YYY*

- **Reproducible and stable:**
  - *always* the same events!
First set: hrtimer & MSR events

Programming a timer

- **Baremetal**
  + Write the timeout value in an MSR (for instance, `IA32_TSC_DEADLINE MSR`)
  + Will get an interrupt when timeout expires

- **Virtual Machine**
  + Write the timeout value in an MSR... Of course, it can’t be the host MSR directly!
  + VMExit with reason: `msr_write`
  + Host handles the exit (and the timer, and the real and emulated MSR write, etc)
  + ...
  + ...
First set: hrtimer & MSR events

- High Resolution Timers:
  When in TSC Deadline mode, they interact with the Deadline MSR

```
# trace-cmd record -C x86-tsc -e msr:* -e kvm:* -e timer:* -A tumbleweed:823 -e timer:* -e msr:* sleep 1
```

<table>
<thead>
<tr>
<th>Time</th>
<th>Process</th>
<th>Event</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>2360.150329</td>
<td>CPU 0/KVM-1977</td>
<td>kvm_entry:</td>
<td>vcpu 0, rip 0xfffffffffaa079246</td>
</tr>
<tr>
<td>2360.150331</td>
<td>CPU 0/KVM-1977</td>
<td>hrtimer_start:</td>
<td>hrtimer=0xffff90127dc21540 ...</td>
</tr>
<tr>
<td>2360.150333</td>
<td>CPU 0/KVM-1977</td>
<td>kvm_exit:</td>
<td>vcpu 0 reason MSR_WRITE ...</td>
</tr>
<tr>
<td>2360.150333</td>
<td>CPU 0/KVM-1977</td>
<td>kvm_hv_timer_state:</td>
<td>vcpu_id 0 hv_timer 1</td>
</tr>
<tr>
<td>2360.150333</td>
<td>CPU 0/KVM-1977</td>
<td>kvm_msr:</td>
<td>msr_write 6e0 = 0x61d24e6ecd4</td>
</tr>
<tr>
<td>2360.150333</td>
<td>CPU 0/KVM-1977</td>
<td>kvm_entry:</td>
<td>vcpu 0, rip 0xfffffffffaa079246</td>
</tr>
<tr>
<td>2360.150334</td>
<td>CPU 0/KVM-1977</td>
<td>write_msr:</td>
<td>6e0, value 61d24e6ecd4</td>
</tr>
</tbody>
</table>
Second set: idle task & HLT events

- **Idle task:**
  Special task executed when there are no other runnable tasks. Runs the **idle loop**.

- **Idle loop:**
  Conceptually (and originally), a NOP busy loop. Nowadays, optimized with idle states (**c states**).

- **Idle loop inside a VM:**
  (typically) VMExit to the host, to let other tasks/VMs run

```c
while (!need_resched()) {
    // ...
    if (cpu_idle_force_poll || tick_check_broadcast_expired()) {
        tick_nohz_idle_restart_tick();
        cpu_idle_poll();
    } else {
        cpuidle_idle_call();
    }
    arch_cpu_idle_exit();
}
```
Second set: idle task & HLT events

```bash
# trace-cmd record -C x86-tsc -e power:* -e kvm:* -A tumbleweed:823 -e power:* sleep 1
```

- **idle=halt**

<table>
<thead>
<tr>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;idle&gt;-0 00017183092310903: cpu_idle: state=1 cpu_id=0</td>
<td>CPU 0/KVM-7361 00017183092315864: kvm_exit: vcpu 0 reason HLT rip 0xffffffffa5e07 ...</td>
</tr>
</tbody>
</table>

- **idle=poll**

<table>
<thead>
<tr>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;idle&gt;-0 00011350708411803: cpu_idle: state=0 cpu_id=0</td>
<td>CPU 0/KVM-1594 0001135071125170: kvm_exit: vcpu 0 reason INTR rip 0xffffffff98792b6 ...</td>
</tr>
</tbody>
</table>
Infer information: we have sets, and then?

- We can use the timers and idle sets to measure the achieved time synchronization accuracy.

- For each set, **subtract** the timestamps of the events in it, in order to generate many **event deltas**.

- Compute the **mean** and the **standard deviation** of the event deltas found, which combined will indicate the overall performance.
Let’s automate it: final tool

- `.checker <host-file> <guest-file>... [-n event_name]... [-s samples-file]`

- Multi vCPUs and multi guests
  - `-n event_name`: exclude event from validation process
  - `-s filename`: store all samples

- Implemented with `libkshark`, directly using the generated `.dat` files

- Might also be useful to re-think the placement of the tracepoints in the kernel.
Single guest

# ./checker trace.dat trace-tumbleweed.dat

# Generated samples: <filename>-timer.txt

# Timer samples: 1729 samples
3580
2860
3122

# Generated samples: <filename>-idle.txt

# Idle samples: 632 samples
1328
1057
1298
Multiple guests

# ./checker trace.dat trace-tumbleweed.dat trace-tumbleweed2.dat

####################### GLOBAL STATS

Number of events: 121480
Host events inside kvm_entry/kvm_exit block: 7247
Guest events outside kvm_entry/kvm_exit block: 0

TIMER events:  N_Samples: 64, Mean: 2773.515625 ns, Variance: 924496.750000 ns^2, SD: 961.507568 ns
HLT events:  N_Samples: 18, Mean: 1791.000000 ns, Variance: 388813.437500 ns^2, SD: 623.549072 ns

####################### PER GUEST STATS

[+] /home/stefano/tracce/2guests/trace-tumbleweed.dat

Number of events: 71
Events outside kvm_entry/kvm_exit block: 0

TIMER events:  N_Samples: 10, Mean: 2912.399902 ns, Variance: 1457231.750000 ns^2, SD: 1207.158569 ns
HLT events:  N_Samples: 5, Mean: 1227.800049 ns, Variance: 446477.187500 ns^2, SD: 668.189514 ns

[+] /home/stefano/tracce/2guests/trace-tumbleweed2.dat

Number of events: 1805
Events outside kvm_entry/kvm_exit block: 0

TIMER events:  N_Samples: 54, Mean: 2747.796387 ns, Variance: 821608.562500 ns^2, SD: 906.426270 ns
HLT events:  N_Samples: 13, Mean: 2007.615356 ns, Variance: 197714.312500 ns^2, SD: 444.650787 ns
PTP vs KVM: analysis

- Different scenarios
  - Idle system
  - Stressed system(s): `stress-ng --matrix 0`

- 30 sessions of tracing, 20 seconds long
  - ~8000 samples for timer sequence
  - ~3000 samples for idle sequences
### Invalid guest events

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PTP</td>
</tr>
<tr>
<td>No stress</td>
<td>0.136 %</td>
</tr>
<tr>
<td>Guest stress</td>
<td>0.167 %</td>
</tr>
<tr>
<td>Host stress</td>
<td>0.191 %</td>
</tr>
<tr>
<td>Both stress</td>
<td>0.811 %</td>
</tr>
<tr>
<td></td>
<td>Timer events</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
</tr>
<tr>
<td><strong>PTP</strong></td>
<td></td>
</tr>
<tr>
<td>No stress</td>
<td>7.40 μs</td>
</tr>
<tr>
<td>Guest stress</td>
<td>3.59 μs</td>
</tr>
<tr>
<td>Host stress</td>
<td>6.70 μs</td>
</tr>
<tr>
<td>Both stress</td>
<td>13.52 μs</td>
</tr>
<tr>
<td><strong>KVM</strong></td>
<td></td>
</tr>
<tr>
<td>No stress</td>
<td>3.33 μs</td>
</tr>
<tr>
<td>Guest stress</td>
<td>0.78 μs</td>
</tr>
<tr>
<td>Host stress</td>
<td>0.85 μs</td>
</tr>
<tr>
<td>Both stress</td>
<td>1.28 μs</td>
</tr>
</tbody>
</table>
PTP vs KVM: not a fair battle...

- Performance
PTP vs KVM: not a fair battle...

- Stability
• Host-Guest tracing is possible, if we can synchronize the traces

• trace-cmd supports multiple synchronization mechanisms:

<table>
<thead>
<tr>
<th></th>
<th>PTP</th>
<th>KVM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Complex to implement</td>
<td>Very simple implementation</td>
</tr>
<tr>
<td></td>
<td>Not accurate enough</td>
<td>Very accurate</td>
</tr>
<tr>
<td></td>
<td>Hypervisor agnostic</td>
<td>Relies on debugfs entries</td>
</tr>
</tbody>
</table>
Conclusion

- **KVM relies on **debugfs **entries**
  - stable enough ABI?
  - what if debugfs is compiled out (for security reasons)?

- **Feedback wanted:** how else can we read the (per-vCPU) tsc-offset, tsc-scaling-ratio, etc values:
  - a new system call?
  - ...?
Links & Acknowledgments

● Links
  ○ trace-cmd.org
  ○ kernelshark.org
  ○ Sync evaluation tool

● Acknowledgments
  ○ Yordan Karadzhov (VMware) <y.karadz@gmail.com>
  ○ Steven Rostedt (VMware) <rostedt@goodmis.org>
  ○ Dario Faggioli (SUSE) <dfaglioli@suse.com>
Thanks for your attention!