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/

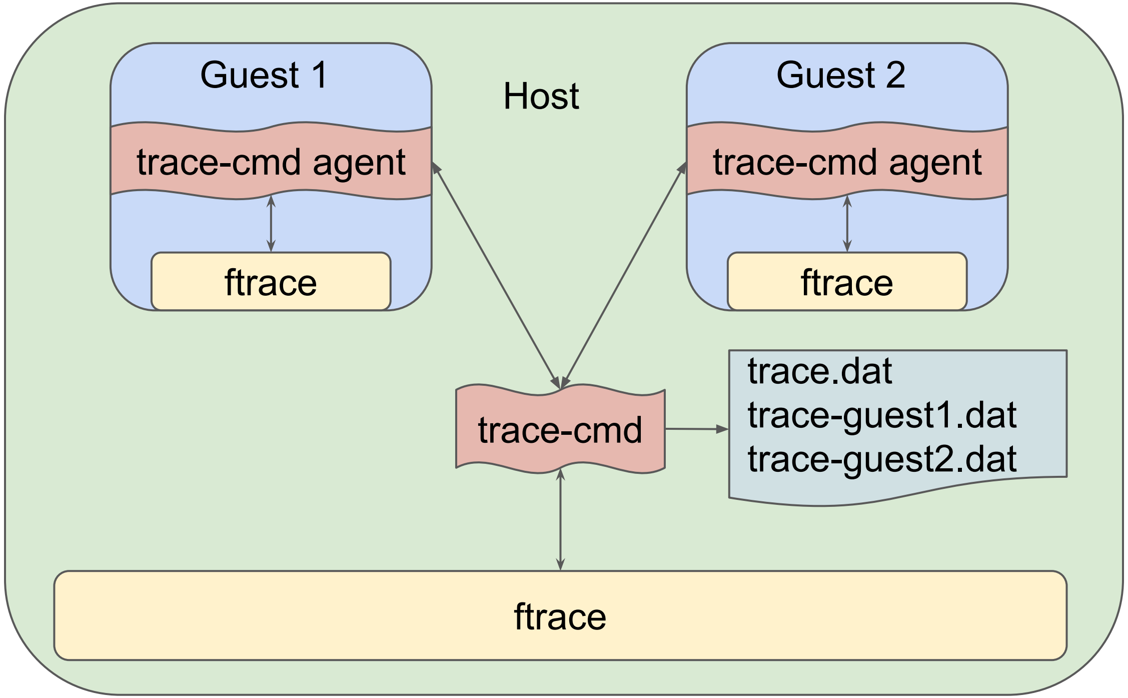
Tzvetomir Stoyanov
Software engineer in the Open Source Technology Center, VMware/Bulgaria working on the Linux Kernel tracing infrastructure.
- e-mail: tz.stoyanov@gmail.com
- github: https://github.com/tzstoyanov

Ftrace

```
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 5)               |  wake_up_q() {
 5)               |    try_to_wake_up() {
 5)   0.148 us    |      _raw_spin_lock_irqsave();
 5)               |      select_task_rq_fair() {
 5)   0.167 us    |        available_idle_cpu();
 5)   0.415 us    |        update_cfs_rq_h_load();
 5)               |        select_idle_sibling() {
 5)   0.216 us    |          available_idle_cpu();
 5)   0.519 us    |        }
 5)   1.931 us    |      }
 5)   0.151 us    |      _raw_spin_lock();
 5)               |      update_rq_clock() {
 5)   0.193 us    |        update_irq_load_avg();
 5)   0.529 us    |      }
```

- 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

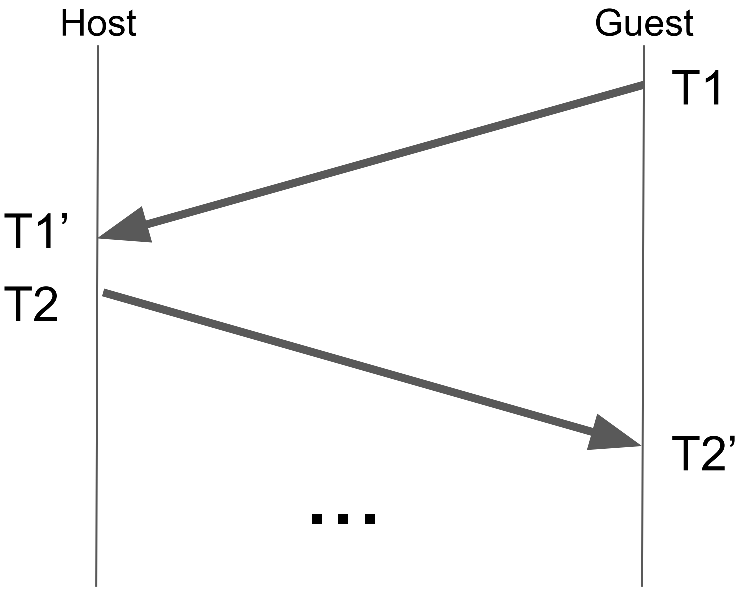
- ➔ Fast transfer of huge tracing data between guest and host
- ➔ Time stamps synchronisation

Trace data transfer

Channel	Throughput
FIFOs	1000 MB/s
vsockets	900 MB/s
TCP/IP sockets	275 MB/s

Measured on a laptop with Intel i5 CPU with 8 cores and and 16G RAM

Time stamps synchronisation - PTP



Clock offset

(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

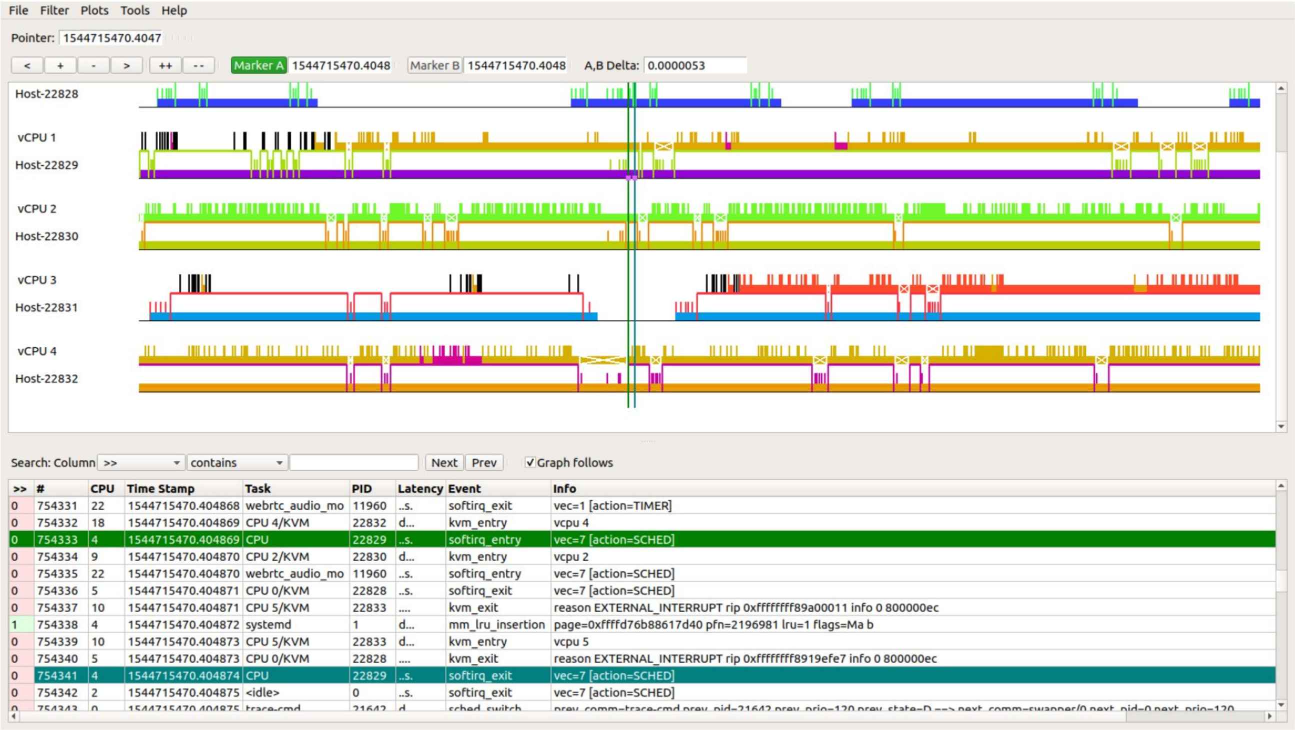
Time stamps synchronisation - KVM

/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

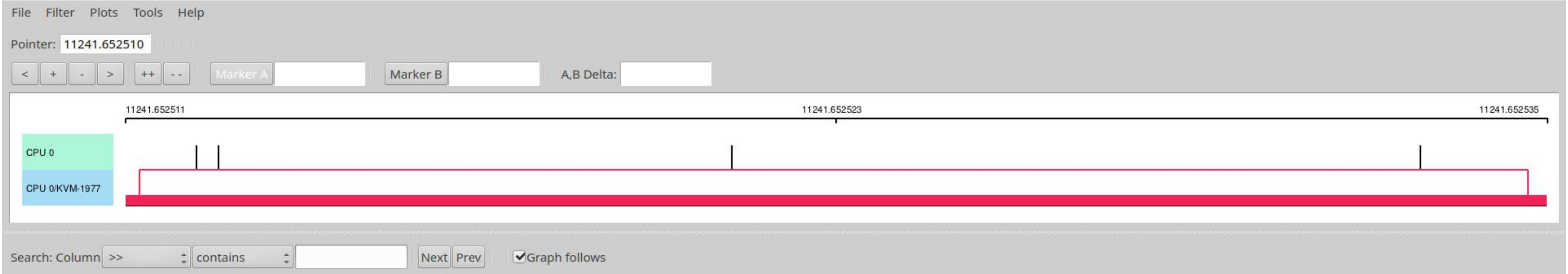
Two crucial events**:**

- **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?

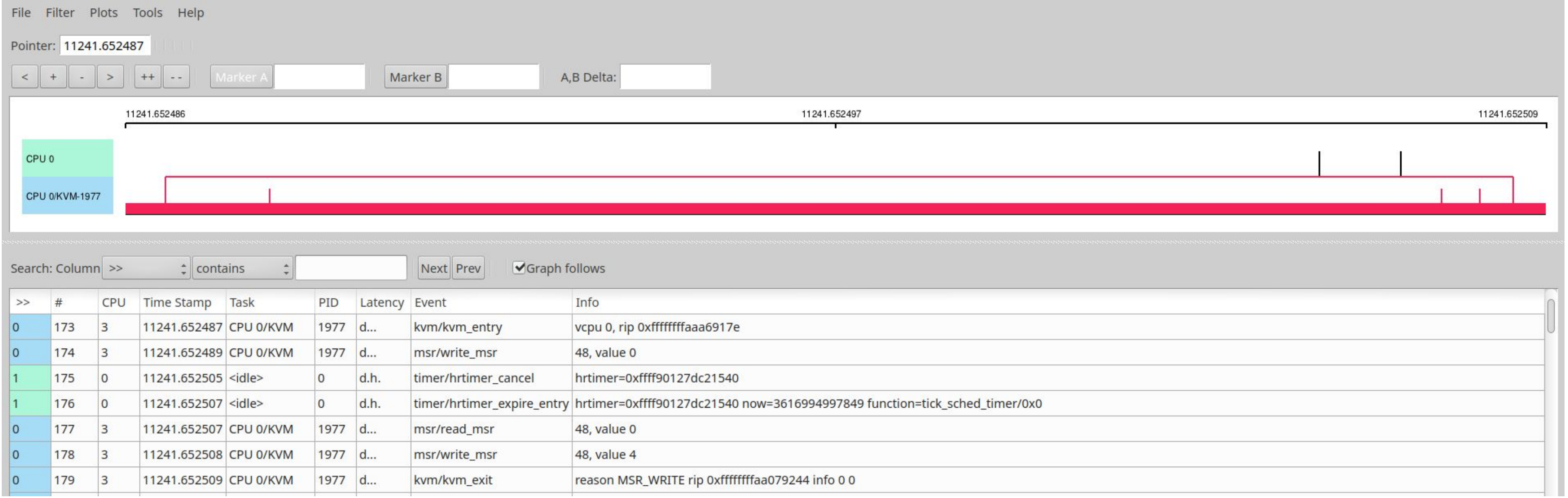


>>	#	CPU	Time Stamp	Task	PID	Latency	Event	Info
0	182	3	11241.652511	CPU 0/KVM	1977	d...	kvm/kvm_entry	vcpu 0, rip 0xffffffffaa079246
1	184	0	11241.652512	<idle>	0	d.h.	timer/hrtimer_expire_exit	hrtimer=0xffff90127dc21540
1	185	0	11241.652513	<idle>	0	d.h.	timer/hrtimer_start	hrtimer=0xffff90127dc21540 function=tick_sched_timer/0x0 expires=3616998901864 softexpires=3616998901864
1	186	0	11241.652521	<idle>	0	d.h.	msr/write_msr	6e0, value 9e5fd35bc24
1	187	0	11241.652533	<idle>	0	....	power/cpu_idle	state=4294967295 cpu_id=0
0	190	3	11241.652535	CPU 0/KVM	1977	d...	kvm/kvm_exit	reason HLT rip 0xffffffffaaa6917d info 0 0

Validation: what we expect?

>>	#	CPU	Time Stamp	Task	PID	Latency	Event	Info
0	190	3	11241.652535	CPU 0/KVM	1977	d...	kvm/kvm_exit	reason HLT rip 0xffffffffaaa6917d info 0 0
0	191	3	11241.652536	CPU 0/KVM	1977	d...	kvm/kvm_wait_lapic_expire	vcpu 0: delta 280220 (late)
0	192	3	11241.652537	CPU 0/KVM	1977	....	kvm/kvm_eoi	apicid 0 vector 236
0	193	3	11241.652537	CPU 0/KVM	1977	....	kvm/kvm_pv_eoi	apicid 0 vector 236
0	194	3	11241.652539	CPU 0/KVM	1977	d...	timer/hrtimer_start	hrtimer=0xffff949286151210 function=apic_timer_fn/0x0 expires=3718274710335 softexpires=3718274710335
1	195	0	11241.652540	<idle>	0	d...	timer/tick_stop	success=1 dependency=NONE
1	196	0	11241.652540	<idle>	0	d...	timer/hrtimer_cancel	hrtimer=0xffff90127dc21540
1	197	0	11241.652541	<idle>	0	d...	timer/hrtimer_start	hrtimer=0xffff90127dc21540 function=tick_sched_timer/0x0 expires=3617026901864 softexpires=3617026901864
0	199	3	11241.652541	CPU 0/KVM	1977	....	kvm/kvm_hv_timer_state	vcpu_id 0 hv_timer 0
1	200	0	11241.652542	<idle>	0	d...	power/cpu_idle	state=1 cpu_id=0

Validation: what we expect?



>>	#	CPU	Time Stamp	Task	PID	Latency	Event	Info
0	173	3	11241.652487	CPU 0/KVM	1977	d...	kvm/kvm_entry	vcpu 0, rip 0xffffffffaaa6917e
0	174	3	11241.652489	CPU 0/KVM	1977	d...	msr/write_msr	48, value 0
1	175	0	11241.652505	<idle>	0	d.h.	timer/hrtimer_cancel	hrtimer=0xffff90127dc21540
1	176	0	11241.652507	<idle>	0	d.h.	timer/hrtimer_expire_entry	hrtimer=0xffff90127dc21540 now=3616994997849 function=tick_sched_timer/0x0
0	177	3	11241.652507	CPU 0/KVM	1977	d...	msr/read_msr	48, value 0
0	178	3	11241.652508	CPU 0/KVM	1977	d...	msr/write_msr	48, value 4
0	179	3	11241.652509	CPU 0/KVM	1977	d...	kvm/kvm_exit	reason MSR_WRITE rip 0xffffffffaa079244 info 0 0

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

```
H:    CPU 0/KVM-1977   [000]   2360.150329: kvm_entry:             vcpu 0, rip 0xffffffffaa079246
G:       <idle>-0       [000]   2360.150331: hrtimer_start:         hrtimer=0xffff90127dc21540 ...
H:    CPU 0/KVM-1977   [000]   2360.150333: kvm_exit:              vcpu 0 reason MSR_WRITE ...
H:    CPU 0/KVM-1977   [000]   2360.150333: kvm_hv_timer_state:     vcpu_id 0 hv_timer 1
H:    CPU 0/KVM-1977   [000]   2360.150333: kvm_msr:               msr_write 6e0 = 0x61d24e6ecd4
H:    CPU 0/KVM-1977   [000]   2360.150333: kvm_entry:             vcpu 0, rip 0xffffffffaa079246
G:       <idle>-0       [000]   2360.150334: write_msr:             6e0, value 61d24e6ecd4
```

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:**

  (typicall) VMExit to the host, to let other tasks/VMs run

```
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

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

- idle=*halt*

```
G:          <idle>-0     [000]17183092310903: cpu_idle:             state=1 cpu_id=0
H:       CPU 0/KVM-7361  [003]17183092315864: kvm_exit:             vcpu 0 reason HLT rip 0xffffffffabda5e07 ...
```

- idle=*poll*

```
G:          <idle>-0     [000]11350708411803: cpu_idle:             state=0 cpu_id=0
                                   .
                                   .
                                   .
H:       CPU 0/KVM-1594  [000]11350713125170: kvm_exit:             vcpu 0 reason INTR rip 0xffffffff9f8792b6 ...
```

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
```

```
################### GLOBAL STATS

Number of events: 119675
Host events inside kvm_entry/kvm_exit block: 4199
Guest 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
```

```
# ./checker trace.dat trace-tumbleweed.dat -s <filename>
```

```
# 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
        HTL 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
        HTL 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

PTP vs KVM: validation

Invalid guest events

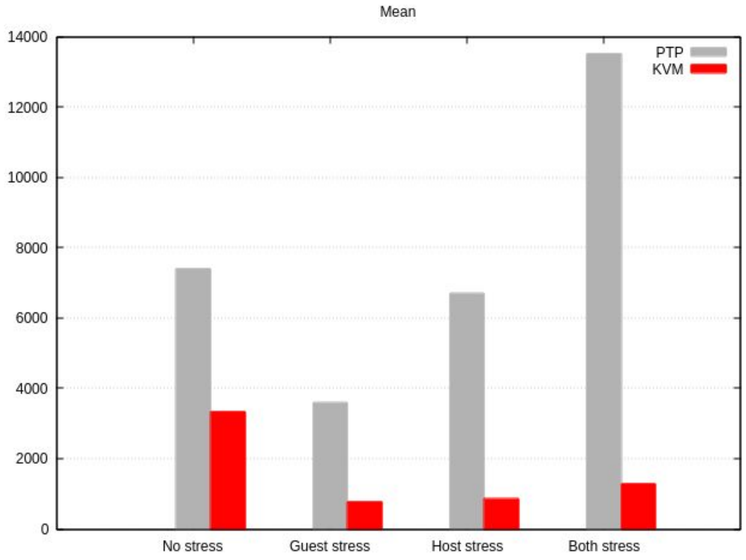
		Algorithm	
		PTP	KVM
Scenarios	No stress	0.136 %	0
	Guest stress	0.167 %	0
	Host stress	0.191 %	0
	Both stress	0.811 %	0

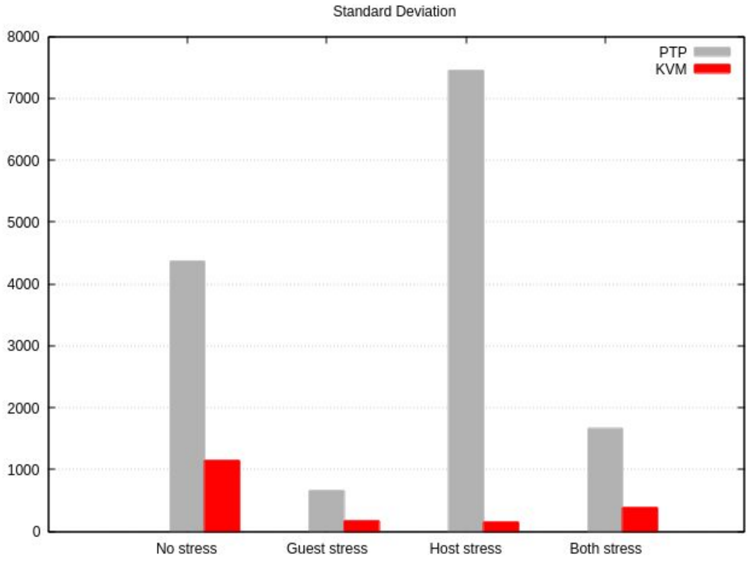
PTP vs KVM: evaluation

		Timer events		Halt events	
		Mean	Standard Deviation	Mean	Standard Deviation
PTP	No stress	7.40 μs	4.36 μs	4.43 μs	2.71 μs
	Guest stress	3.59 μs	0.65 μs	N/A	N/A
	Host stress	6.70 μs	7.45 μs	4.50 μs	1.91 μs
	Both stress	13.52 μs	1.66 μs	N/A	N/A
KVM	No stress	3.33 μs	1.14 μs	2.40 μs	0.59 μs
	Guest stress	0.78 μs	0.16 μs	N/A	N/A
	Host stress	0.85 μs	0.15 μs	0.67 μs	0.16 μs
	Both stress	1.28 μs	0.38 μs	N/A	N/A

PTP vs KVM: not a fair battle...

- Performance

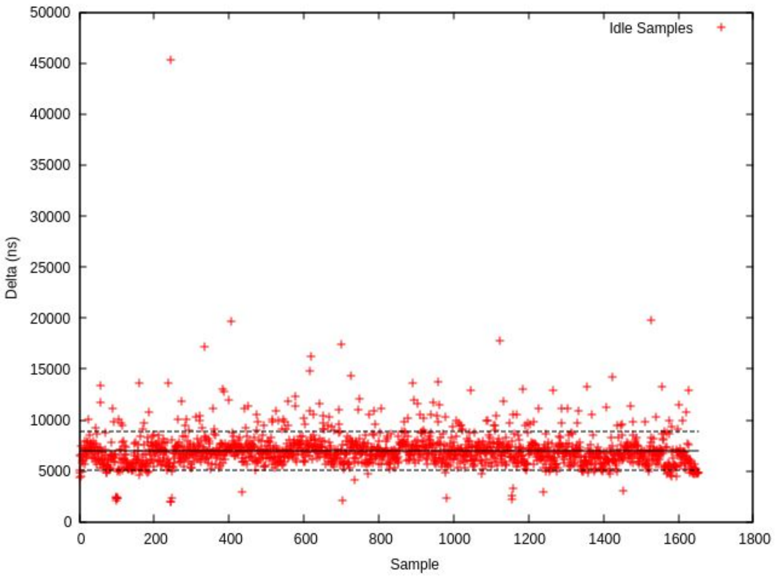




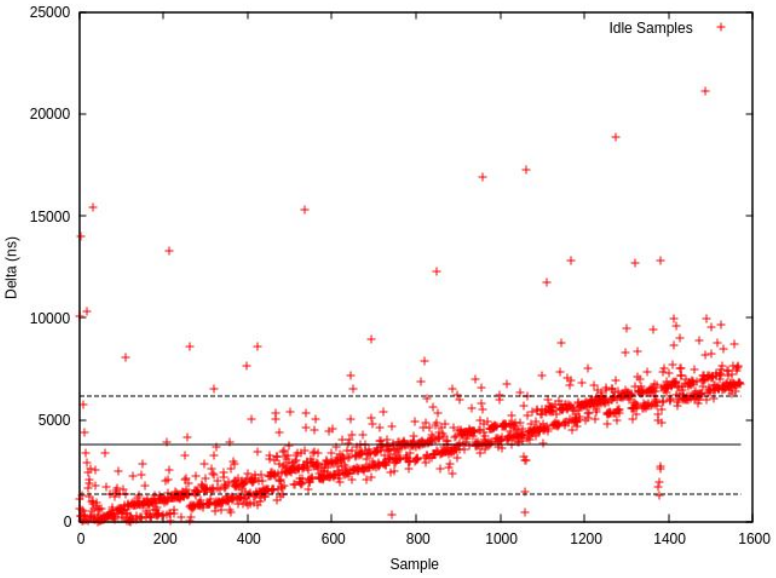
PTP vs KVM: not a fair battle...

- Stability

KVM



PTP



Conclusion

- Host-Guest tracing is possible, if we can synchronize the traces
- trace-cmd supports multiple synchronization mechanisms:

PTP	KVM
Complex to implement	Very simple implementation
Not accurate enough	Very accurate
Hypervisor agnostic	Relies on debugfs entries

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) <dfaggioli@suse.com>

Thanks for your attention!