Host-Assisted Virtual Machine Tracing and Analysis

Abderrahmane Benbachir
Michel Dagenais

Dec 7, 2017

École Polytechnique de Montréal
Laboratoire DORSAL
Introduction

Hypertracing

Hypercall

Boot-up Analysis

Shared Memory

Virtualization Awareness

Future Work

Questions
Monitoring Tools

Tracer
- callback, serialize, write
  - Lttng
  - Perf
  - Ftrace
  - Strace

Aggregator
- callback, compute, update
  - Dtrace
  - eBPF
  - SystemTap

Offloader
- Sync
- Async
What is hypertracing?

**Hypertracing** = Offloading traces from guest to host (or vice versa)
How to do hypertracing?

Communication Channels

- Shared Memory
- Network TCP/IP
- Virtio Virtio-serial

Host

- probes
- buffer
- sync
- async

Qemu

Virtual Machine

- offloader
- kernel

tracer → data → consumer

Hypercall
Hypercall Channel
Hypercall in a nutshell

Virtual Machine

Qemu (layer not involved)

Host Hypervisor

Hardware

L_0

L_1

Context Switch overhead

kvm_exit

kvm_entry

Hypervisor overhead
Which tracer to use?

Host Tracers Overhead (hypercall)

The diagram shows the overhead in nanoseconds for different tracers with the baseline indicated.

- Ftrace: 19%
- Lttng 2.10: 21%
- Lttng 2.7: 31%
- Perf: 41%
Event Aggregation

Tracing hypercalls on host using Lttng

```plaintext
kvm_x86_exit: { cpu_id = 0 }, { exit_reason = 18, guest_rip = 94139651574911, isa = 1... }

kvm_x86_hypercall: { cpu_id = 0 }, { nr = 1000, a0=0xffffffff89c2ce60, a1 = 0, a2 = 0, a3 = 0 }

kvm_x86_entry: { cpu_id = 0 }, { vcpu_id = 2 }

hypergraph_host: { nr = 1000, a0 = 0xffffffff89c2ce60, a1 = 0, a2 = 0, a3 =0, vcpu_id = 2, guest_rip = 94139651574911, exit_overhead = 85 }
```
Nested VMs
Nested Virtualization

Remove Exit Multiplication

- Qemu (not involved)
- Host Hypervisor
- Hardware

Single level | Two levels | Three levels
Exit-handling code in the hypervisor is slower when run in L1 or L2 than the same code running in L0. Transition between L1, L2, ... Ln involve an exit to L0 and then an entry.

Hardware: Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz (x86_64)
EPT-on-EPT-on-EPT
Micro-Benchmarks (worst-case)

<table>
<thead>
<tr>
<th>Nested overhead</th>
<th>L1</th>
<th>L2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lttng</td>
<td>18%</td>
<td>206.4 ns</td>
</tr>
<tr>
<td>Ftrace</td>
<td>x2</td>
<td>243.1 ns</td>
</tr>
<tr>
<td>Hypertracing + compress (2)</td>
<td>12.6%</td>
<td>314 ns</td>
</tr>
<tr>
<td>Perf</td>
<td>15%</td>
<td>478.1 ns</td>
</tr>
<tr>
<td>Hypertracing</td>
<td>3.6%</td>
<td>812 ns</td>
</tr>
</tbody>
</table>

Cost of:
- Guest Tracing
- Host Tracing
- Hypercall
- Syscall Probe
- Time+Compression
- Write to disk

System call
Event Compression

**sched_switch** example

```c
sched_switch { prev_prio = 0, prev_tid = 0, next_tid = 10, prev_state = 0, next_prio = 0, prev_comm = "swapper/1", next_comm = "migration/1" } ...
sched_switch { prev_prio = 0, prev_tid = 10, next_tid = 11, prev_state = 1, next_prio = 0, prev_comm = "migration/1", next_comm = "ksoftirqd/1" } ...
sched_switch { prev_prio = 0, prev_tid = 11, next_tid = 12, prev_state = 1, next_prio = 0, prev_comm = "ksoftirqd/1", next_comm = "kworker/1:0" } ...
sched_switch { prev_prio = 0, prev_tid = 12, next_tid = 0, prev_state = 1, next_prio = 0, prev_comm = "kworker/1:0", next_comm = "swapper/1" } ...
sched_switch { prev_prio = 0, prev_tid = 0, next_tid = 10, prev_state = 0, next_prio = 0, prev_comm = "swapper/1", next_comm = "migration/1" }
```

---

**Diagram Description**

- **Guest** and **Host** timelines are shown.
- Time delta and hypercall annotations are integrated.

**Bitwidths**:
- **prio** -> 8 bits
- **state** -> 8 bits
- **pid_max** -> 15 bits
- **time_delta** -> 32 bits

---
Event Compression

Offloading Latency of a Hypercall When Enabling Event Compression
Boot-up Analysis
Boot-up phases

Kernel boot-up

very early

early

main

run_init_process()

pure_initcall()

trace_init()

boot scripts/systemd

Userland boot-up

External modules

Forking processes
Boot-up Sequence

- **POLYTECHNIQUE MONTREAL**
- **Abderrahmane Benbachir (Abder)**

**start**

- **kernel**

**console**

**security**

- **run_init_process()**
  - 18 ms
  - 1 sec
  - 900 ms
  - 700 ms

**rootfs**

**device**

**late**

**core**

**postcore**

**arch**

**subsys**

**fs**

**Ftrace**

**Lttng**

**Userland Bootup**

**External Modules**

**trace_init()**

**105 million events**

=> **9.5 GB**

**No tracer**

- We did a patch for that:
  - ftrace: very early function tracing
Boot-up level tracing overhead

<table>
<thead>
<tr>
<th>Bootup level</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch</td>
<td>0.017 %</td>
</tr>
<tr>
<td>postcore_sync</td>
<td>0.033 %</td>
</tr>
<tr>
<td>postcore</td>
<td>0.038 %</td>
</tr>
<tr>
<td>subsys_sync</td>
<td>0.041 %</td>
</tr>
<tr>
<td>core</td>
<td>0.05 %</td>
</tr>
<tr>
<td>core_sync</td>
<td>0.052 %</td>
</tr>
<tr>
<td>pure</td>
<td>0.067 %</td>
</tr>
<tr>
<td>device_sync</td>
<td>0.087 %</td>
</tr>
<tr>
<td>rootfs</td>
<td>0.155 %</td>
</tr>
<tr>
<td>arch_sync</td>
<td>0.432 %</td>
</tr>
<tr>
<td>late</td>
<td>2.247 %</td>
</tr>
<tr>
<td>early</td>
<td>2.396 %</td>
</tr>
<tr>
<td>subsys</td>
<td>3.618 %</td>
</tr>
<tr>
<td>fs</td>
<td>7.142 %</td>
</tr>
<tr>
<td>device</td>
<td>10.415 %</td>
</tr>
<tr>
<td>fs_sync</td>
<td>40.76 %</td>
</tr>
</tbody>
</table>
Boot levels: Marker event

Resources

Console

Boot levels

Control Flow

Statistics

Resources  oneos-bootup  Merge  guest-kernel

CPU Usage
Dynamic Analysis

Top 17 most called functions

- note_page: 2.014%
- acpi_ps_get_opcode_info: 2.268%
- acpi_ut_create_update_state_and_push: 2.705%
- acpi_ut_create_update_state: 2.705%
- acpi_ut_push_generic_state: 2.961%
- acpi_ut_pop_generic_state: 2.967%
- acpi_ut_create_generic_state: 2.968%
- acpi_ut_delete_generic_state: 2.968%
- ACPI_os_release_object: 3.513%
- kmem_cache_free: 3.549%
- kmem_cache_alloc: 3.622%
- _cond_resched: 3.967%
- ACPI_ut_update_ref_count: 8.858%
- ACPI_os_release_lock: 8.858%
- ACPI_os_acquire_lock: 8.858%
- _raw_spin_unlock_irqrestore: 9.031%
- _raw_spin_lock_irqsave: 9.033%

Frequency

0e+00, 1e+06, 2e+06, 3e+06, 4e+06, 5e+06
Dynamic Analysis

Filters

- *acpi*: 60.16%
- _raw_spin_*: 18.34%
- *kmem_cache*: 7.19%
- _cond_resched*: 3.97%
- note_page: 2.01%
- *slab*: 0.97%
- *console*: 0.61%
- *fb*: 0.29%
- *mulex*: 0.2 %
## Boot-up Tracing

<table>
<thead>
<tr>
<th>Host Tracing</th>
<th>Guest</th>
<th>Optimization &amp; Configuration</th>
<th>Events</th>
<th>Trace Size</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tracer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Baseline</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>868.52 ms</td>
</tr>
<tr>
<td></td>
<td>Function graph</td>
<td>Function entries &amp; exits*</td>
<td>168 M</td>
<td>Buffer size</td>
<td>22 secs</td>
</tr>
<tr>
<td></td>
<td>Hypergraph</td>
<td>Function entries &amp; exits</td>
<td>-</td>
<td>-</td>
<td>1 min 08s</td>
</tr>
</tbody>
</table>

### Hypergraph

<table>
<thead>
<tr>
<th>Function tracing</th>
<th>Optimization &amp; Configuration</th>
<th>Events</th>
<th>Trace Size</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypergraph</td>
<td>Function entries &amp; exits</td>
<td>322 M</td>
<td>9.4 GB</td>
<td>1 min 20s</td>
</tr>
<tr>
<td></td>
<td>Event Aggregation (Host opt.)</td>
<td>105 M</td>
<td>6.7 GB</td>
<td>57 secs</td>
</tr>
<tr>
<td></td>
<td>Event Compression (Guest opt.)</td>
<td>163 M</td>
<td>4.8 GB</td>
<td>40 secs</td>
</tr>
<tr>
<td></td>
<td>Aggr + Comp</td>
<td>52 M</td>
<td>3.4 GB</td>
<td>30 secs</td>
</tr>
<tr>
<td></td>
<td>Aggr + Comp + Filtering (Guest opt.)</td>
<td>3.2 M</td>
<td>207 MB</td>
<td>2.6 secs</td>
</tr>
</tbody>
</table>

### Lttng

<table>
<thead>
<tr>
<th>Function tracing</th>
<th>Optimization &amp; Configuration</th>
<th>Events</th>
<th>Trace Size</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypergraph + Bootlevel</td>
<td>Aggr + Comp + Filtering (Guest opt.)</td>
<td>early 20 K</td>
<td>1.4 MB</td>
<td>93 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pure  5 K</td>
<td>0.3 MB</td>
<td>2.6 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>core   8 K</td>
<td>0.5 MB</td>
<td>4 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>postcore 5 K</td>
<td>0.4 MB</td>
<td>2.7 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>arch   32 K</td>
<td>2.1 MB</td>
<td>17 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>subsys 115 K</td>
<td>7.5 MB</td>
<td>143 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fs     2.5 M</td>
<td>155 MB</td>
<td>1871 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rootfs 10 K</td>
<td>0.7 MB</td>
<td>6 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>device 412 K</td>
<td>26 MB</td>
<td>410 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>late   258 K</td>
<td>16 MB</td>
<td>87 ms</td>
</tr>
</tbody>
</table>

**Friendly advice:**
Always use function tracing with filters. Enable it for specific cases.

66% overhead
Shared Memory Channel
Zero Copy Shared Memory

Virtual Machine

- Guest kernel
- Qemu

App

Guest virtual address space

Consumer

virtual host address space

mmap

Host

physical address space

Zero copy using mmap
Virtualization awareness

Guest

Host

Shared ring buffer

kvm_exit

kvm_entry

kvm_exit

kvm_entry

Synchronization ?

Work in progress ...
Future Work

Compare Shared M. vs Hypercall vs Virtio
Shutdown analysis
Virtualization awareness
Feedbacks & Questions

abderrahmane.benbachir@polymtl.ca
https://github.com/abenbachir
References

Hypercall Implementation: https://gist.github.com/abenbachir/344822b5ba9fc5ac384cdec3f087e018

QEMU Hypertrace Patches: http://patchwork.ozlabs.org/project/qemu-devel/list/?state=&q=Hypertrace&archive


Callstack xml analysis: https://gist.github.com/abenbachir/e813790f183945b6f74dc74ecce57c75