Virtual machine CPU monitoring with Kernel Tracing

Mohamad Gebai
Michel Dagenais

15 May, 2014
École Polytechnique de Montreal
Content

- General objectives
- Current approaches
- Kernel tracing
- Trace synchronization
- Virtual Machine Analysis
- Execution flow recovery
General objectives

- Getting the state of a virtual machine at a certain point in time
- Quantifying the overhead added by virtualization
- Track the execution of processes inside a VM
- Aggregate information from host and guests
- Monitoring multiple VMs on a single host OS
- Finding performance setbacks due to resource sharing among VMs
Current approaches

- Top
  - Steal time: percentage of vCPU preemption for the last second

Does not reflect the effective load on the host
  - 0% for idle VMs even if the physical CPU is busy
  - Not enough information
Current approaches

- Perf kvm
- Information about VM exits, performance counters

<table>
<thead>
<tr>
<th>VM-EXIT</th>
<th>Samples</th>
<th>Samples%</th>
<th>Time%</th>
<th>Min Time</th>
<th>Max Time</th>
<th>Avg time</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_INSTRUCTION</td>
<td>145</td>
<td>70.05%</td>
<td>0.26%</td>
<td>2us</td>
<td>1478us</td>
<td>14.04us</td>
</tr>
<tr>
<td>EPT_MISCONFIG</td>
<td>25</td>
<td>12.08%</td>
<td>0.01%</td>
<td>1us</td>
<td>6us</td>
<td>3.79us</td>
</tr>
<tr>
<td>APIC_ACCESS</td>
<td>21</td>
<td>10.14%</td>
<td>0.02%</td>
<td>2us</td>
<td>13us</td>
<td>6.12us</td>
</tr>
<tr>
<td>HLT</td>
<td>8</td>
<td>3.86%</td>
<td>99.71%</td>
<td>2420us</td>
<td>248023us</td>
<td>99141.25us</td>
</tr>
<tr>
<td>VMCALL</td>
<td>5</td>
<td>2.42%</td>
<td>0.00%</td>
<td>0us</td>
<td>1us</td>
<td>1.20us</td>
</tr>
<tr>
<td>EXCEPTION_NMI</td>
<td>2</td>
<td>0.97%</td>
<td>0.00%</td>
<td>0us</td>
<td>1us</td>
<td>1.31us</td>
</tr>
<tr>
<td>EXTERNAL_INTERRUPT</td>
<td>1</td>
<td>0.48%</td>
<td>0.00%</td>
<td>32us</td>
<td>32us</td>
<td>32.37us</td>
</tr>
</tbody>
</table>

Total Samples:207, Total events handled time:795429.44us.

- No information from inside the VM
- No information about VM interactions
Kernel tracing

- Trace scheduling events
  - `sched_switch` for context switches
  - `sched_migrate_task` for thread migration between CPUs (optional)
  - `sched_process_fork, sched_process_exit`
- Trace VMENTRY and VMEXIT on the hypervisor (hardware virtualization)
  - `kvm_entry`
  - `kvm_exit`
Tracing virtual machines

- Each VM is a process
- Each vCPU is 1 thread
  - Per-thread state can be rebuilt
- A vCPU can be in VMX root mode or VMX non-root mode
- A vCPU can be preempted on the host
- The VM can't know when it is preempted or in VMX root mode
- Processes in the VM seem to take more time
- Trace host and guests simultaneously
Trace synchronization

- Time difference between host and an idle VM
Trace synchronization

- Time difference between host and an active VM
Trace synchronization

- Based on the fully incremental convex hull synchronization algorithm
- 1-to-1 relation required between events from guest and host
- Tracepoint is added to the guest kernel
- Executed on the system timer interrupt softirq
- Triggers a hypercall which is traced on the host
- Resistant to vCPU migrations and time drifts
Trace synchronization

- Kernel module added to LTTng as an addon
- In the guest:
  - Trigger a hypercall (event a)
- On the host:
  - Acknowledge the hypercall (event b)
  - Give control back to the guest (event c)
- In the guest:
  - Acknowledge the control (event d)
Trace synchronization

- Host and guest threads, as seen before.

- ..and after synchronization
Trace synchronization

- Time difference between host and VM after synchronization
**TMF Virtual Machine View**

- Shows the state of each vCPU of a VM
- Aggregation of traces from the host and the guests

<table>
<thead>
<tr>
<th>VM qemu:Debian</th>
<th>VM qemu:Ubuntu</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCPU 0</td>
<td>VCPU 0</td>
</tr>
<tr>
<td>VCPU 1</td>
<td>VCPU 1</td>
</tr>
</tbody>
</table>

- 2 VM:
  - Debian and Ubuntu
  - vCPU 0 and vCPU 1 are complementary; fighting over the same pCPU
TMF Virtual Machine View

- Detailed information of execution inside the VM
- Process burnP6 (TID 2635) is deprived from the pCPU while the CPU time is still accounted for
**TMF Virtual Machine View**

- Shows latency introduced by the hypervisor (i.e., emulation in KVM) to the nanosecond scale

<table>
<thead>
<tr>
<th>VM qemu:Debian</th>
<th>VCPU 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>watchdog/0: 10</td>
<td></td>
</tr>
<tr>
<td>kworker/0:1H: 151</td>
<td>kworker/0:1H</td>
</tr>
<tr>
<td>kworker/0:1: 23</td>
<td>kworker/0:1</td>
</tr>
<tr>
<td>burnP6: 2635</td>
<td>burnP6</td>
</tr>
<tr>
<td>sshd: 2648</td>
<td>sshd</td>
</tr>
<tr>
<td>kworker/u4:0: 6</td>
<td></td>
</tr>
<tr>
<td>migration/0: 7</td>
<td>migration/0</td>
</tr>
</tbody>
</table>

Virtual machine CPU monitoring with Kernel Tracing
Use case

- Periodic critical task
- Inexplicably takes longer on some executions
- 100% CPU usage from the guest's point of view
Use case

- VCPU is preempted on the host
- Invisible to the VM
- Duration of preemption is easily measurable
Execution flow recovery

- Build the execution flow centered around a certain task A
- List of execution intervals affecting the completion time of A
- Find the source of preemption across systems
- Example:

![Execution flow diagram]

Virtual machine CPU monitoring with Kernel Tracing
Execution flow recovery

Previous example:

Execution flow centered around task 3525:
Acknowledgements

- Ericsson
- CRSNG
- Professor Michel Dagenais
- Geneviève Bastien
- Francis Giraldeau
- DORSAL Lab