Nested virtualization: shadow turtles

Orit Wasserman
Red Hat

KVM forum 2013
Agenda

• Nested virtualization (the turtles project) overview
• What's new in nested VMX?
• What is VMCS?
• VMCS and nested virtualization
• Shadow VMCS implementation

Karen Noel

Karen Noel
What is nested virtualization?

• L0 – hypervisor running directly on the hardware (host hypervisor)
• L1 – guest hypervisor or nested hypervisor
• L2 – nested guest
• VMCS\textsubscript{x}→\textsubscript{y} – VMCS used by L\textsubscript{x} to run L\textsubscript{y} (VMCS\textsubscript{x}y for short)
• The scope of the talk is limited to Intel x86 architecture.
What is nested virtualization?

• Running multiple **unmodified** hypervisors, with their associated **unmodified** guest VM's simultaneously on the x86 hardware

• x86 supports a single level of virtualization

• Does not support nesting in hardware (mainframe does)
Why?

• Operating system may have built in hypervisors (Windows 2008R2/2012R2, Linux/KVM)
• To be able to run another hypervisor in the cloud
• Security (e.g. Hypervisor level toolkit)
• Co-design of x86 hardware and system software
• Testing, demonstrating and debugging hypervisors
• Live migration of hypervisors
How?

- L0 multiplexes the hardware between L1 and L2, running both as guests of L0 – without either being aware of it.
How?

1. L0 runs L1 with VMCS0 → 1
2. L1 prepares VMCS1 → 2 and executes vmlaunch
3. vmlaunch traps to L0
4. L0 merges VMCS 0 → 1 with VMCS 1 → 2 into VMCS0 → 2
5. L0 launches L2
6. L2 causes a trap
7. L0 handles trap itself or forwards it to L1
8. ... 
9. Eventually, L0 resumes L2
10. Repeat
Nested VMX cost

- To handle a single L2 exit, L1 does many things: read and write the VMCS, disable interrupts, etc.
- Those operations can trap, leading to exit multiplication.
- Exit multiplication: a single L2 exit can cause 40-50 L1 exits!
- Optimize: execute a single exit faster and reduce frequency of exits.
Whats new in nested VMX?

• Many bug fixes :)

• Lots of new tests in kvm-unit-tests (Arthur Chunqi Li)

• Nested EPT (Nadav Har'El/Gleb Natapov) – Gleb will talk about it next, don't miss it!
“Unrestricted guest” support for nested VMX

- "Unrestricted Guest" feature was added to the VMX specification in Intel Westmere and onward
- It allows kvm guests to run real mode and unpaged mode code natively under VMX mode when EPT is turned on
- With the unrestricted guest there is no need to emulate the guest real mode code in the vm86 container or in the emulator
- The guest big real mode code runs like native
- By Jan Kiszka
VMX Preemption timer for nested VMX

• Enable setting a timer for the VM executing. When the timer expires there will be a vmexit
• The timer is set for the VM time slice
• Used to improve virtual machine scheduling because VM won't need to exit on every timer interrupt (fewer exits)
• By Arthur Chunqi Li
What is VMCS - Virtual Machine Control Structure

• Each vCPU has a structure/block for storing it's state and information needed for running it.
• VMCS is stored in on-chip-memory
• Special VMX instructions to access it: VMREAD and VMWRITE
• It is divided into 4 sections:
  • Guest state
  • Host state
  • Control – fields to control VMExit/VMEntry behavior
  • Read Only – Usually contain VMExit information
What is VMCS - Virtual Machine Control Structure cont

• Special encoding that can move fields between processor versions

<table>
<thead>
<tr>
<th>31</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>reserved (=0)</td>
<td>W</td>
<td>0</td>
<td>T</td>
<td>INDEX</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend:
W (width of field): 00=16-bit, 01=64-bit, 10=32-bit, 11=natural-width
T (Type of field): 00=control, 01=read-only, 10=guest-state, 11=host-state
A (Access-type): 0= full, 1=high

(Note: Access-type must be ‘full’ for 16-bit, 32-bit, and ‘natural’ widths)

Karen Noel
Nested VMX and VMCS accesses

- Every time the L1 hypervisor access VMCS1 → 2 it causes an exit
- Need to eliminate those exits
- One solution is to use a para-virtual nested hypervisor as is done in the turtles project
- Binary patching – Could be complicated as VMREAD and VMWRITE are short commands
Shadow VMCS

- Allow L0 hypervisor to define a shadow VMCS
- This VMCS can be accessed without a vmexit in guest mode
- Removes the extra exits penalty for nested virtualization
- Was added in the Haswell architecture

Figure 4. With Intel® VMCS Shadowing, two VMMs can be hosted on the same physical system, without the performance penalties of software-only nested solutions.
Shadow VMCS implementation (1/3)

• Shadow VMCS is processor-dependent and must be accessed by L0 or L1 using VMREAD and VMWRITE instructions only

• To avoid hardware dependencies:
  • Software defined VMCS1 → 2 format is part of L1 address space
  • Processor-specific shadow VMCS format is part of L0 address space

• L0 synchronize the shadow VMCS content with the software-controlled VMCS1 → 2 format

• Design simplifies live migration of L1, which does not depended on the shadow VMCS layout
Shadow VMCS implementation (2/3)

• Sync process:
  • Before running L2 after switching from L1 we need to update all the changes L1 did, from the shadow VMCS to VMCS1 → 2:
    • Load the shadow VMCS to the processor using VMPTRLD
    • Read each VMCS field with using VMREAD command
  • Before switching back to L1 after running L2 we need to sync from VMCS1 → 2 to the shadow VMCS:
    • Load the shadow VMCS to the processor using VMPTRLD
    • Write each VMCS field with using VMWRITE command
Shadow VMCS implementation (3/3)

- Reducing syncing cost:
  - Shadow only the necessary fields
  - Use a bitmap for fields that are shadowed for read
  - A field will be synced in the first scenario only if the bit is set
  - Use a bitmap for fields that are shadowed for write
  - A field in the second scenario will be synced only if the bit is set
  - A flag to indicate that VMCS1→2 was changed by L0. Reduce the second scenario occurrence

Karen Noel
## Results

- By Abel Gordon
- From the turtles paper (DRW stands for direct read/write):

<table>
<thead>
<tr>
<th></th>
<th>kernbench</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Host</td>
<td>Guest</td>
<td>Nested</td>
<td>Nested&lt;sub&gt;DRW&lt;/sub&gt;</td>
</tr>
<tr>
<td>Run time</td>
<td>324.3</td>
<td>355</td>
<td>406.3</td>
<td>391.5</td>
</tr>
<tr>
<td>% overhead vs. host</td>
<td>-</td>
<td>9.5</td>
<td>25.3</td>
<td>20.7</td>
</tr>
<tr>
<td>% overhead vs. guest</td>
<td>-</td>
<td>-</td>
<td>14.5</td>
<td>10.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>SPECjbb</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Host</td>
<td>Guest</td>
<td>Nested</td>
<td>Nested&lt;sub&gt;DRW&lt;/sub&gt;</td>
</tr>
<tr>
<td>Score</td>
<td>90493</td>
<td>83599</td>
<td>77065</td>
<td>78347</td>
</tr>
<tr>
<td>% degradation vs. host</td>
<td>-</td>
<td>7.6</td>
<td>14.8</td>
<td>13.4</td>
</tr>
<tr>
<td>% degradation vs. guest</td>
<td>-</td>
<td>-</td>
<td>7.8</td>
<td>6.3</td>
</tr>
</tbody>
</table>

Table: kernbench and SPECjbb results
What is still missing for nested VMX?

- Stability
- Nested VT-d to allow usage of device assignment in nested guests to improve I/O performance
- Running other hypervisors as L1 (nested hypervisor). ESX requires Acknowledge interrupt on exit.
- Live migration