Nested Virtualization Friendly KVM

Sheng Yang, Qing He, Eddie Dong
Virtualization vs. Nested Virtualization

- Single-Layer Virtualization
- Multi-Layer (Nested) Virtualization

![Diagram showing the differences between single-layer and multi-layer (nested) virtualization]
Challenge of Nested Virtualization

• Ideal virtualization model:
  – The Virtual Platform is exactly the same as the real hardware platform, except for timing/performance.
  – However, commercial VMM typically presents only a subset of hardware features in the virtual platform
    • Enough to accommodate commercial OS
    • But can’t run the VMM inside → No nested virtualization
      – KVM/Xen/Vmware/Hyper-V are all examples

• Challenges of nested virtualization:
  – Present full underlying hardware features to the virtual platform efficiently, such as VMX, EPT.
Nested Virtualization: Virtual VMX

- Virtual VMX

- Significant virtualization overhead was observed due to shadow page fault in L1 VMM
  - Kernel build in L2 guest is only 1/3 of L1 guest
Nested Virtualization: Virtual EPT

- **Shadow-like virtual EPT**
  - Write-protection guest EPT table
    - Update sEPT when gEPT changes
  - Directly invept of guest
  - May suffer from global lock

- **VTLB-like virtual EPT**
  - No write-protection to gEPT
  - Trap-and-emulate guest INVEPT
    - Updating sEPT when cached mappings may (?) be changed
  - Better SMP scalability (Preferred)

Prefer VTLB-like virtual EPT for better scalability!
Performance Challenges

- L1 VMM VMCS register access is trapped-and-emulated by L0 VMM
  - An L1 VM exit may trigger tens of VMCS access, which is trapped-and-emulated by L0 VMM
  - Emulation of INVEPT is extremely expensive
    - The entire sEPT has to be re-generated 😊

- Reducing the frequency of L1 VM exit is key
  - Virtual EPT significantly improves performance
  - Virtual VT-d etc.
  - Nested virtualization friendly guest
Optimizations

- Minimize the frequency of L1 VM exit
  - Build as possible as static guest EPT table
  - Mitigate the host swap activity in L1 VMM
  - Cross-layer I/O para-virtualization
- Accelerate handling of virtual VM exit
  - Minimize privilege resource access per virtual VM exit
    - Such as VMCS access
  - Avoid unnecessary INVEPT
  - Choose efficient operands
Pre-build vs. On-demand EPT

- On-demand build of EPT hurts nested virtualization
  - KVM sets up EPT table on demand so far
  - Page age checking of LRU zaps EPT entry

A command line option for static EPT?
Mitigate the Host Swap Activity

- Virtual host swap is expensive in L1 VMM
  - It may generate up to ~4K/s EPT table modification
  - Emulation of INVEPT has to zap and rebuilt the entire shadow EPT table in vTLB-like virtual EPT
    - L0 VMM may defer part of the shadow EPT rebuilt effort

Retain host swap in L0 VMM rather than L1 VMM by presenting enough pseudo memory to L1 guest
Cross-Layer I/O Paravirtualization

- Backend service from L1 may trigger tremendous VM exit to L0
- Can L0 directly service L2 I/O?
  - Network is stateless
  - Cooperation between L1/L2 BE

Give some data here: How L1 BE overhead is?
Accelerate Handling of Virtual VM exit

• # of privilege resource (VMCS) access in virtual VM exit handler (top 3)

Extending cache_reg to efficiently reduce average VMCS access #!
Avoid Unnecessary INVEPT

- Emulation of INVEPT in vTLB-like virtual EPT implementation has to remove the entire sEPT table
  - Extreme heavy cost 😊

![INVEPT During Qemu Build](chart)

<table>
<thead>
<tr>
<th>Frequency (Hz)</th>
<th>Linux (KVM) version</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.6.25</td>
</tr>
<tr>
<td>50</td>
<td>2.6.31</td>
</tr>
<tr>
<td>40</td>
<td>2.6.32</td>
</tr>
<tr>
<td>30</td>
<td>2.6.33</td>
</tr>
<tr>
<td>20</td>
<td>2.6.34</td>
</tr>
</tbody>
</table>

- `kvm_mmu_flush_tlb`
- `CR3 opt.`
- `CR0.ts opt.`
Efficient Operands in VMCS Access

- Register operands can be easily emulated by L0 VMM, while memory operand is expansive
  - Access of L1 memory needs additional map and un-map in L0 VMM

So far KVM uses register operand for VMCS read/write, keep the good behavior 😊
Performance Status

KB Time (Seconds)

2.6.31  2.6.32  2.6.33  2.6.34  2.6.35-rc5  w/ vmcs caching