Instruction caching for bhyve

Mihai Carabas, Neel Natu
{mihai,neel}@freebsd.org

AsiaBSDCon 2015
Tokyo University of Science
Tokyo, Japan
March 12 – 15, 2015
Who we are?

- Mihai Carabas
  - PhD Student and Teaching Assistant at the University POLITEHNICA of Bucharest, Romania
  - DragonFly BSD (SMT aware scheduler - 2012 / Intel EPT for vkernels - 2013)
  - FreeBSD - bhyve (instruction caching - 2014 / coordinating students in bhyve projects - current)
Who we are?

▶ Mihai Carabas
  ▶ PhD Student and Teaching Assistant at the University POLITEHNICA of Bucharest, Romania
  ▶ DragonFly BSD (SMT aware scheduler - 2012 / Intel EPT for vkernels - 2013)
  ▶ FreeBSD - bhyve (instruction caching - 2014 / coordinating students in bhyve projects - current)

▶ Neel Natu
  ▶ principal contributor for the bhyve project (together with Peter Grehan)
  ▶ started as a FreeBSD/mips committer
Context

- Hardware Assisted Virtualization
  - a new CPU privilege level
  - memory virtualization (EPT / NPT)
Context

- Hardware Assisted Virtualization
  - a new CPU privilege level
  - memory virtualization (EPT / NPT)

- What about controlling the APIC from the VM?
  - each control register access traps in the hypervisor
  - the hypervisor needs to emulate that access
Steps for handling a trap in the hypervisor

- Fetch the instruction
  - manually walking the Guest OS page table to find the physical address
  - map the address in the hypervisor address space and copy the instruction

- Decode the instruction
  - variable length instructions for x86 platforms

- Emulate the instruction
  - execute the instruction in the name of the VM

Any solution to jump over some of them?
Steps for handling a trap in the hypervisor

- Fetch the instruction
  - manually walking the Guest OS page table to find the physical address
  - map the address in the hypervisor address space and copy the instruction

- Decode the instruction
  - variable length instructions for x86 platforms
Steps for handling a trap in the hypervisor

- Fetch the instruction
  - manually walking the Guest OS page table to find the physical address
  - map the address in the hypervisor address space and copy the instruction

- Decode the instruction
  - variable length instructions for x86 platforms

- Emulate the instruction
  - execute the instruction in the name of the VM
Steps for handling a trap in the hypervisor

- Fetch the instruction
  - manually walking the Guest OS page table to find the physical address
  - map the address in the hypervisor address space and copy the instruction

- Decode the instruction
  - variable length instructions for x86 platforms

- Emulate the instruction
  - execute the instruction in the name of the VM

- Any solution to jump over some of them?
Identify an instruction for caching

- Cached object: struct vie

- Unique identifier (key)
Identify an instruction for caching

- **Cached object:** struct vie
- **Unique identifier (key)**
  - VM ID: struct vm *
  - instruction address (RIP)
  - pointer to the page table (CR3)
- **Stored in struct vie_cached**
Integrating caching mechanism in the emulation code

- New interface provided by vmm_instruction_cache.h
Integrating caching mechanism in the emulation code

- New interface provided by vmm_instruction_cache.h

- vm_inst_cache_add
  - adds the instruction to the cache
  - mark as read-only the pages related to the instruction
Integrating caching mechanism in the emulation code

- New interface provided by vmm_instruction_cache.h

- vm_inst_cache_add
  - adds the instruction to the cache
  - mark as read-only the pages related to the instruction

- vm_inst_cache_delete
  - removes an instruction from cache
  - solves the write page fault
Integrating caching mechanism in the emulation code

- New interface provided by vmm_instruction_cache.h
  - vm_inst_cache_add
    - adds the instruction to the cache
    - mark as read-only the pages related to the instruction
  - vm_inst_cache_delete
    - removes an instruction from cache
    - solves the write page fault
  - vm_inst_cache_lookup
Caching flow

vm_handle_inst_emul
Caching flow

vm_handle_inst_emul

vm_inst_cache_lookup
Caching flow

- vm_handle_inst_emul
- vm_inst_cache_lookup
- vmm_fetch_instruction
- vmm_decode_instruction

Not found
Caching flow

1. `vm_handle_inst_emul`
2. `vm_inst_cache_lookup`
3. `vmm_fetch_instruction`
4. `vmm_decode_instruction`
5. `vm_inst_cache_add`
6. `struct vie`
   (the decoded instruction)
7. **Not found**

流程图表示了缓存过程，当未找到指令时，会继续执行流程。
Caching flow

vm_handle_inst_emul

Not found

vm_inst_cache_lookup

vmm_fetch_instruction

vmm_decode_instruction

vm_inst_cache_add

struct vie
(the decoded instruction)

Found
Cache invalidation flow

- Page Fault
  - vm_handle.paging
    - vm_fault
Cache invalidation flow

- Page Fault
- vm_handle_paging
- KERN_PROTECTION_FAILURE
- vm_fault
- Is cache locked?
Cache invalidation flow

- Page Fault
  - vm_handlepaging
  - KERN_PROTECTION_FAILURE
    - vm_fault
  - Is cache locked?
    - No
    - Lock the cache
Cache invalidation flow

1. Page Fault
2. vm_handle_paging
3. KERN_PROTECTION_FAILURE
4. vm_fault
5. Is cache locked?
   - Yes
   - No
     - Lock the cache
     - inst_cache_delete
6. Again
Cache invalidation flow

1. Page Fault
2. vm_handle_paging
3. KERN_PROTECTION_FAILURE
4. vm_fault
5. Is cache locked?
6. No
7. Lock the cache
8. inst_cache_delete
9. Again
10. KERN_SUCCESS?
Cache invalidation flow

- Page Fault
  - vm_handle_paging
    - KERN_PROTECTION.FAILURE
      - vm_fault
        - Is cache locked?
          - No
            - Lock the cache
            - inst_cache_delete
          - Yes
            - KERN_SUCCESS?
              - Yes
                - SUCCESS
              - No
                - Again

Cache invalidation flow

1. Page Fault
2. `vm_handle_paging`
3. `vm_fault`
4. Check if cache is locked.
   - **No**:
     - Lock the cache
     - `inst_cache_delete`
     - **Again**
   - **Yes**:
     - `KERN_SUCCESS`?
       - **No**:
         - `EFAULT`
       - **Yes**:
         - `SUCCESS`
Cache invalidation flow

Page Fault

vm_handle_paging

KERN_PROTECTION_FAILURE

vm_fault

Is cache locked?

Yes

Unlock the cache

No

Lock the cache

inst_cache_delete

KERN_SUCCESS?

No

efaault

Yes

success

Again
Efficiency evaluation

- Micro-benchmarking
  - kernel module accessing the LAPIC ID in a tight loop
  - measure the average access time
  - 10500 ticks without instruction caching
  - 6700 ticks with it (30% improvement)
Efficiency evaluation

- Micro-benchmarking
  - kernel module accessing the LAPIC ID in a tight loop
  - measure the average access time
  - 10500 ticks without instruction caching
  - 6700 ticks with it (30% improvement)

- Real world workloads
  - simple loop running in user space and make buildworld in VM
  - measure the time that needs to finish the workload (time command)
  - measure the cache efficiency (hits, misses) (VMM_STAT_* custom counters)
Real world cache efficiency

**Table: CPU intensive bash script**

<table>
<thead>
<tr>
<th>Number of instruction cache</th>
<th>vCPU0</th>
<th>vCPU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>hits</td>
<td>699.519</td>
<td>840.485</td>
</tr>
<tr>
<td>insertions</td>
<td>10.395</td>
<td>5.743</td>
</tr>
<tr>
<td>evictions[0]</td>
<td>7.139</td>
<td>8.926</td>
</tr>
<tr>
<td>evictions[1]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>evictions[2]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>evictions[3]</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table: make buildworld -j2**

<table>
<thead>
<tr>
<th>Number of instruction cache</th>
<th>vCPU0</th>
<th>vCPU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>hits</td>
<td>19.204.630</td>
<td>12.930.500</td>
</tr>
<tr>
<td>insertions</td>
<td>8.688.733</td>
<td>9.051.295</td>
</tr>
<tr>
<td>evictions[0]</td>
<td>8.563.694</td>
<td>9.173.381</td>
</tr>
<tr>
<td>evictions[1]</td>
<td>1.131</td>
<td>1.457</td>
</tr>
<tr>
<td>evictions[2]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>evictions[3]</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Speed-up for running time

**Table: CPU intensive bash script**

<table>
<thead>
<tr>
<th>hw.vmm.instruction_cache</th>
<th>time spent in execution (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>225</td>
</tr>
<tr>
<td>0</td>
<td>230</td>
</tr>
</tbody>
</table>

**Table: make buildworld -j2**

<table>
<thead>
<tr>
<th>hw.vmm.instruction_cache</th>
<th>time spent in execution (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13900</td>
</tr>
<tr>
<td>0</td>
<td>13938</td>
</tr>
</tbody>
</table>
Related work

- KVM driver isn’t using any caching technique
- there exists something in the fetch part (pre-fetch the instructions bytes in advanced)
Related work

- KVM driver isn’t using any caching technique
- there exists something in the fetch part (pre-fetch the instructions bytes in advanced)
- KVM community opinion as stated in a KVM-Intel presentation from 2012
  - they want to rely on the hardware only
  - all the interrupt handling in hardware (virtualize the APIC without VM exists)
  - a VM exit is too expensive

instruction emulation will still be used for other devices (e.g. HPET, AHCI)
Related work

- KVM driver isn’t using any caching technique
- there exists something in the fetch part (pre-fetch the instructions bytes in advanced)
- KVM community opinion as stated in a KVM-Intel presentation from 2012
  - they want to rely on the hardware only
  - all the interrupt handling in hardware (virtualize the APIC without VM exists)
  - a VM exit is too expensive
- instruction emulation will still be used for other devices models (e.g. HPET, AHCI)
Conclusions

- Cache the emulated instructions in order to decrease the time spent in the hypervisor
- Handled corner cases like contention on the VM page table without using a big lock
- Theoretical good results (e.g. 30% improvement of the average access time)
- Didn’t find a real world workload to benefit from this mechanism

Thank you for your attention!

ask questions