Improvement of Real-time Performance of KVM

Tomoki Sekiyama
Linux Technology Center
Yokohama Research Lab.
Hitachi, Ltd.

Copyright (c) 2012 Hitachi Ltd., Yokohama Research Lab. All rights reserved.
1. **Overview of realtime virtualization**

2. Improvement of KVM realtime performance

3. Performance evaluation

4. Current status of development
Use case of virtualization in RT systems

• Control systems for factory automation / social infrastructure
  – Require low latencies and deadline constraints
  – Not CPU intensive; typically use single CPU core only
    → To utilize many cores by consolidating multiple systems
  – Used for very long time (10+ years)
    → To preserve old software environment in new hardware

• Embedded systems / Appliances
  – Provide realtime performance AND user-friendly interface
  – Gradually port applications from legacy RTOS to Linux
    → To run RTOS guest and Linux in parallel
Use case of virtualization in RT systems

- **Enterprise systems** (e.g. Automated trading systems)
  - To preserve old software environment in new hardware
  - To deploy applications easily into cloud DCs

- **HPC** (not RT system, but has similar requirements)
  - Low latency features are required to reduce overhead by network communication among nodes
  - Virtualization technology is used in public cloud HPC environments (e.g. Amazon EC2) to realize easy deployment and easy management of computation nodes
Requirements for realtime virtualization

• Low latency
  – Respond to external events quickly

• Bare-metal performance
  – Not to slow down applications

• Preserve (at least soft) realtime quality of the guest OS
  – Blocking the guest will lose realtime performance
    → Temporal interfere from host tasks must be avoided

• Sometimes modification of the guest OS should be avoided
  – Some legacy GPOS / RTOS is difficult to modify
Why using KVM?

- non-KVM solutions
  - Some RTOS supports Linux guests
  - Tiny hypervisors just for partitioning

- KVM has ...
  - Advanced virtualization features
    - Sharing and overcommit resources
    - Support virtualization hardware (EPT, x2APIC, VT-d, ...)
    - Well-defined management / debug interfaces (e.g. libvirt)
  - Large community
    - Upstreamed in Linux kernel
    - Well tested in various environment
    - Rapid innovation
Issues in realtime virtualization

- mlock(2), SCHED_RR and exclusive cpuset for a guest can improve realtime performance
- Still some issues remain:
  - Interfere from host’s kernel thread
  - Temporal overhead by interrupt forwarding
    - Overheads in interrupt path
- Interrupt from passed-through PCI devices also takes similar path
- Especially problematic if interrupted frequently (10Gb NIC, etc.)
  - The other issues (not focused in this presentation)
    - I/O emulation in vCPU thread, locks in hypervisor...
1. Overview of realtime virtualization

2. Improvement of KVM realtime performance

3. Performance evaluation

4. Current status of development
How to improve RT performance

- **CPU isolation**
  - Partitioning CPUs for realtime guest
    → Avoid interference from kernel threads etc.

- **Direct interrupt delivery** (requires CPU isolation)
  - Eliminate the overhead of interrupt forwarding
  - for passed-through PCI devices & local APIC timer
    → Improve latencies and reduce host CPU usage

**Example: timer interrupt**

- **Guest**
  - set APIC timer
  - emulate APIC access

- **Host**
  - VM Exit
  - VM Enter

**Diagram**

- Direct Local APIC access (for timer)
- Direct interrupt delivery
- Direct EOI
CPU isolation

- Dedicate some of CPUs to the guest
  - Make the CPUs offline from Linux host
    - Only provides minimal functions to run vCPU
    - Stop host kernel threads on the CPU
  - Execute guest vCPU thread on the CPU

- Benefit of CPU isolation
  - Avoid Interference from host kernel tasks
  - Assure Bare-metal CPU performance
    - Not interrupted by other guests or processes
  - Enable guest OS to occupy some CPU facilities (local APIC, etc)
    - This is needed for direct IRQ delivery (described in next slides)
Interface to CPU isolation

1. Offline CPUs to be dedicated

```bash
# echo 0 > /sys/devices/system/cpu/cpuX/online
```

2. (in qemu) Use `ioctl(2)` to set the dedicated CPU id for each vCPU

```c
ioctl(vcpu[i], KVM_SET_SLAVE_CPU, slave_cpu_id[i]);
```

→ The specified CPU is booted with minimal function to execute VM
   (Direct interrupt delivery features are also activated)

3. (in qemu) Start vCPU by `KVM_RUN`

```c
ioctl(vcpu[i], KVM_RUN, 0);
```

→ vcpu thread is suspended while vcpu is running on the dedicated CPU
   (resumed on VM Exit that cannot handled by KVM)
Direct interrupt delivery

• Core idea
  – Exploit CPU (Intel VT-x and AMD SVM) feature to deliver interrupts directly to guests
    • Disable interception of external interrupt
    • Overhead by VM exit/enter on interrupts can be avoided

Intel VT-x case:

VMCS / Pin-Based VM-Execution Controls

- External interrupt exiting:
  • if 1, external interrupts cause VM exits
  • if 0, they are delivered through the guest IDT
- NMI exiting:
  • Similar setting for NMI
Direct interrupt delivery

• Issue #1
  – Can not distinguish whether an interrupt is for host or guest
    • Can not specify whether each vector causes VM Exit or not
    • While it is running, all interrupts are delivered to the guest
Direct interrupt delivery

• Issue #1
  – Can not distinguish whether an interrupt is for host or guest
    • Can not specify whether each vector causes VM Exit or not
    • While it is running, all interrupts are delivered to the guest

• Solution
  – CPU isolation & IRQ affinity
    • Set IRQ affinity to route interrupts to appropriate CPUs
      – Host devices → host cores
      – Passed-through devices → dedicated core
    – Currently only MSI/MSI-X is supported
    – Shared ISA IRQs require forwarding by host
Direct interrupt delivery

- **Issue #2**
  - Can not send normal IPI for host to dedicated CPUs (delivered to guests!)
    - Needed for ...
      - injection of emulated interrupts (virtual IRQ)
      - TLB shoot down on the host’s memory protection change, etc.

- **Solution**
  - Use NMI instead of normal IPI
    - Whether VM Exit happens on NMI can be independently set
  - NMI is non-maskable: handler is called even in irq disabled context
    - NMI is used just to cause VM exit
    - After VM exit, check requests from other CPUs and handle them
Direct interrupt delivery

• Issue #3
  – The host and the guest use different vectors for the same devices
    • Normal KVM host converts the host’s vector to the guest’s vector
    • For Direct IRQ, **PCI devices must be reconfigured with the guest’s vector**
    • Confused if host receives the guest vector
      – This happens while the VM is exiting (during I/O emulation, etc.)
Direct interrupt delivery

• Issue #3
  – The host and the guest use different vectors for the same devices
    • Normal KVM host converts the host’s vector to the guest’s vector
    • For Direct IRQ, **PCI devices must be reconfigured with the guest’s vector**
    • Confused if host receives the guest vector
      – This happens while the VM is exiting (during I/O emulation, etc.)

• Solution
  – Register the guest’s vector also to the host’s vector → irq mapping on the dedicated CPU
    • If the host receives the guest’s vector, inject it to guest as vIRQ
Direct interrupt delivery flow

- Normal KVM interrupt delivery

- Direct interrupt delivery

Copyright (c) 2012 Hitachi Ltd., Yokohama Research Lab. All rights reserved.
Direct EOI

• In hardware with x2APIC, EOI (End Of Interrupt) for passed-through devices can be done directly from the guest
  – x2APIC provides access to APIC via MSRs (Model Specific Registers)
  – VT-x has bitmask to specify which MSR is exposed to the guest

• Direct EOI must not be applied to virtual IRQ
  – EOI for virtual IRQs must be sent to virtual APIC
    → On virtual IRQ injection, disable direct EOI
    → Re-enable after every virtual IRQ is handled
Direct EOI flow

- Direct interrupt delivery + Direct EOI flow

- Virtual interrupt delivery flow
Direct Local APIC Timer access

- Host kernel timer which uses Local APIC Timer (hrtimer etc.) must be disabled on the dedicated core
  - Timer interrupt is delivered to the guest directly!
- Local APIC Timer also can be exposed to the guest
  - Require x2APIC to access APIC via MSRs
  - Exposed timer related APIC registers:
    - **TMICT** (Timer initial count): write to start timer
    - **TMCCT** (Timer current count): read current timer value
    - **TDCR** (divide control register): read/write frequency settings
  - Non-exposed timer related registers:
    - **LVTT** (local vector table for timer): specify vector, timer mode etc.
      - vector settings must be confirmed by hypervisor
    - MSR: IA32_TSC_DEADLINE
      - TSC value in the guest has offset, so needs conversion
• Normal KVM - Virtual Local APIC Timer flow:

• with Direct Local APIC Timer Access:
1. Overview of realtime virtualization

2. Improvement of KVM realtime performance

3. Performance evaluation

4. Current status of development
Performance evaluation

• Experimental setup
  – Machine: Core i7 3770 (Ivy Bridge), 4core, w/o HyperThreading
    16GB Memory
  – Host: Linux-3.5.0-rc6
    + direct IRQ/EOI/LAPIC patch
  – Guest: Linux-3.4.0 or Linux-3.4.4-rt14
    1 vCPU or 1 dedicated core
  – PCI: Intel 10Gb NIC with SR-IOV
    1 VF is Passed-through to the guest

• cyclicstest: a benchmark to measure realtime performance
  – Measure how quickly a task is woken up by timer
    Interval = 1ms, 300000 loop (5 minutes*)
  – background workload: idle / iperf (I/O load)

Copyright (c) 2012 Hitachi Ltd., Yokohama Research Lab. All rights reserved.
cyclic test results

- Guest: idle / Host: under CPU workload (infinite loop)

(Histogram with logarithmic scale)

<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>Guest</th>
<th>cyclic test latency (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>linux-3.4.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>linux-3.4.4-rt14</td>
<td></td>
</tr>
<tr>
<td>bare-metal</td>
<td>min 1 avg 1 * max 376</td>
<td></td>
</tr>
<tr>
<td>KVM+DirectIRQ</td>
<td>min 2 avg 2 * max 15</td>
<td></td>
</tr>
<tr>
<td>KVM</td>
<td>min 1 avg 2 * max 14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min 7 avg 13 * max 558</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min 6 avg 11 * max 152</td>
<td></td>
</tr>
</tbody>
</table>

* test is too short to evaluate max latency
cyclic test results

- cyclic test results
  - Guest: under network I/O workload (iperf)

<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>Guest</th>
<th>min</th>
<th>avg</th>
<th>* max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>linux-3.4.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bare-metal</td>
<td></td>
<td>3</td>
<td>6</td>
<td>324</td>
</tr>
<tr>
<td>KVM+DirectIRQ</td>
<td></td>
<td>2</td>
<td>14</td>
<td>157</td>
</tr>
<tr>
<td>KVM</td>
<td></td>
<td>6</td>
<td>35</td>
<td>855</td>
</tr>
</tbody>
</table>

* test is too short to evaluate max latency

(Histogram with logarithmic scale)
Network I/O Performance

- Evaluated with traffic between physical NIC ↔ SR-IOV VF
- Throughput (iperf results)

```plaintext
Throughput (Gbps)

KVM+directIRQ: [12 Gbps]
KVM: [10 Gbps]

28% better
```

- Latency (ping results)

```plaintext
Latency (ms)

KVM+directIRQ: [0.06 ms]
KVM: [0.08 ms]

30% faster
```

- Host CPU Usage:
  - 5 - 10% reduced -- because of no need to forward interrupts
1. Overview of realtime virtualization

2. Improvement of KVM realtime performance

3. Performance evaluation

4. Current status of development
Current status

- Patch submission status
  - RFC v1 (June 28):
    - ✓ CPU isolation
    - ✓ direct interrupt delivery
    - × no direct EOI
    - × no LAPIC timer
    - × no SMP guest
    - × no AMD SVM support
    - × no in-kernel PIT emulation
    - × Linux guest only
    - × has an issue in page fault handling
    - × not tested well ...
  - RFC v2 (soon):
    - ✓ CPU isolation
    - ✓ direct interrupt delivery
    - ✓ direct EOI
    - ✓ direct LAPIC timer
    - ✓ SMP guest
    - × no AMD SVM support
    - × no in-kernel PIT emulation
    - × Linux guest only
    - × has an issue in page fault handling
    - × not tested well ...
How to test

1. Apply patch to Linux/KVM and qemu
2. Disable PCI devices to pass-through

   # echo XXXX:XXXX > /sys/bus/pci/drivers/pci-stub/new_id
   # echo 05:00.0 > /sys/bus/pci/drivers/XXXX/unbind
   # echo 05:00.0 > /sys/bus/pci/drivers/pci-stub/bind

3. Offline CPUs to be dedicated

   # echo 0 > /sys/devices/system/cpu/cpu3/online

4. Execute guest VM
   - Currently “–no-kvm-pit” option is required
   - VGA is very slow; not recommended

   # qemu-kvm.patched -m 1024 -cpu qemu64,+x2apic
   -enable-kvm
   -no-kvm-pit
   -serial pty
   -nographic
   -drive file=kvm/test.img,if=virtio
   -device pci-assign,host=05:00.0
Future Plan

- Reduce restrictions
  - in-kernel chip emulation (e.g. PIT)
  - AMD SVM support
  - support Non-Linux guest like RTOS

- Implement direct interrupt (IPI) delivery for virtio
  - Can improve realtime performance with shared devices
  - Migration support?
Thank you!

Questions?
Copyrights and Trademarks Notices

• Linux is a registered trademark of Linus Torvalds.
• All other trademarks and copyrights are the property of their respective owners.