Performance Optimization on Huawei Public and Private Cloud

Jinsong Liu <liu.jinsong@huawei.com>
Lei Gong <arei.gonglei@huawei.com>
Agenda

• Optimization for LHP
• Balance scheduling
• RTC optimization
Agenda

• Optimization for LHP
• Balance scheduling
• RTC optimization
LHP (Lock Holder Preemption)

- More obvious in virtualization
  - vCPU scheduling
  - Task preemption
- Then
  - Potentially blocking the progress of other vCPUs waiting to acquire the same lock
  - Increasing synchronization latency
  - Performance degradation
LHP (Lock Holder Preemption)

• How to solve or alleviate?
  – PLE (Pause Loop Exiting)
  – DLHS (Delay LH scheduling)
  – Co-scheduling
  – Balance scheduling
PLE

• Hardware support
  – VMCS configuration

• Optimization for Lock Waiters
  – VM Exit actively
  – Avoid waste vCPU cycles for invalid spin

• Pros.
  – Supported by upstream

• Cons.
  – Setting appropriate values of ple_gap and ple_windows is difficult
    • Workloads adjustment
  – Find an appropriate vcpu to yield
DLHS (Delay Lock Holder Scheduling)

• Background & precondition
  – Usually, lock holders are under interrupt disable contexts
  – Normally, the period of holding lock is shortly
  – Hardware support (e.g. intel VT-X)
    • interrupt window exiting
  – Software support
    • Hrtimer, …
DLHS (Delay Lock Holder Scheduling)

• Solution
  – Set a grace period for LH before scheduling
  – If one vCPU is LH
    • Start one hrtimer, and
    • Set interrupt window exiting for VMCS
    • If the hrtimer expire
      – Clear interrupt window
      – Continue to schedule for vCPU
  – Judge the vCPU release the lock
    • PLE happened
    • Interrupt window exiting happen
    • then
      – Cancel hrtimer
      – Release grace period
      – Schedule the vCPU immediately
DLHS – performance

Hackbench results (CPU overcommit 1:3)

Unit: sec  Lower is better

VM1-patched
VM2-patched
VM3-patched
VM1
VM2
VM3
Agenda

• Optimization for LHP
• Balance scheduling
• RTC optimization
Co-scheduling & Balance scheduling

Co-scheduling

Run all vCPUs on Time x

Balance-scheduling

Disperse all vCPUs AMAS
Co-scheduling

- **CPU fragmentation**
  - Reduces CPU utilization
  - Delay vCPU execution

- **Priority inversion**
  - Degrades I/O performance

```
+----------+----------+----------+----------+----------+
| pCPU 0   |       | vCPU0    |       | vCPU0    |
| XXX      | vCPU0  | XXX      | vCPU0  |          |
| pCPU 1   | vCPU1  | I/O      | vCPU1  |          |
+----------+----------+----------+----------+----------+
| T_0      | T_1     | T_2      | T_3     | T_4      |
```
Balance scheduling

• Balances vCPU siblings on pCPUs
  – without precisely scheduling the vCPUs simultaneously

• How to?
  – Uses a bitmap to record all used pCPUs for VM
  – Scheduler adjustment
    • Enqueue & dequeue
    • Migration/find_idle_cpu/select_task_rq etc.
Performance evaluation

• **Workload:**
  – Pushserver in Huawei Private Cloud
  – Continuous testing for 24 hours

• **Results**

![Proportion of building chains](chart.png)

<table>
<thead>
<tr>
<th></th>
<th>With balance sched</th>
<th>Without balance sched</th>
<th>1:1 vcpupin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proportion of building chains (%)</td>
<td>93.50%</td>
<td>70%</td>
<td>95.30%</td>
</tr>
</tbody>
</table>
Agenda

• Optimization for LHP
• Balance scheduling
• RTC optimization
RTC on KVM

- Windows use RTC as clock event device
- RTC emulation in Qemu, three timers
  - rtc_periodic_timer
    - Generates periodic interrupts
    - Programmable to occur according to interrupt rate
  - rtc_update_timer
    - Generates alarm interrupts
    - Occur one per second to once per day
  - rtc_coalesced_timer
    - Generates compensation interrupts
    - Slews the lost ticks since different reasons
    - Getting worse and worse with the VM density increase
- Pain points
  - Some operations need to hold BQL
  - Context switching between user space and kernel space
  - Interrupt injecting from user space
  - Performance degradation
    - Latency increase
    - Windows guest density decrease
RTC optimizations on KVM

- Minimize influence of BQL
  - Placing RTC memory region outside BQL
- Using irqfd inject interrupts
- Hyperv features
  - hyperv clock, …
  - Decreases read/write ioports
- Decreases ioport access on 0x70/0x71
- Move RTC emulation to hypervisor
  - Inject interrupts in KVM
  - Reduce context switching
  - But
    - Large attack surface
    - Incompatible with new features, such as split irqchip
- Optimize RTC compensation solution
RTC compensation solution

- Slew RTC ticks in hypervisor directly
- Count the coalesced interrupts
  - When an RTC interrupt injecting failed
  - Adjust the count when RTC interrupt rate changes
- Inject coalesced interrupts after EOI handler if exist
  - Don’t need a separate timer
  - More timely
  - Throttle the speed if there is too many coalesced interrupts
- Live migration support
  - Save the coalesced interrupts in src side
  - Restore them in dest side
  - Both KVM and Qemu need to be patched
Optimization evaluation

Before optimization

After optimization
Thank You!