Virtio-blk Performance Improvement

Asias He <asias@redhat.com>, Red Hat
Nov 8, 2012, Barcelona, Spain
KVM FORUM 2012
Storage transport choices in KVM

- Full virtualization: IDE, SATA, SCSI
  - Good guest compatibility
  - Lots of trap-and-emulate, bad performance
- Para virtualization: virtio-blk, virtio-scsi
  - Virtio ring buffer provides efficient transport for guest-host communication
  - Provide more virtualization friendly interface, higher performance
- Device assignment
  - Pass hardware to guest, high-end usage, high performance
  - Exclusive access, limited number of slot in a server, hard to do live migration
Why improve virtio-blk

- I/O intensive applications
  - Need high storage performance
- Virtio-blk
  - Simple, Just simple read/write/flush command, no scsi overhead, Fast SSD -> PCIE interface instead of SCSI or SATA interface
  - Available for a while, benefits existing users
  - virtio-blk is about ~3 times faster than virtio-scsi in my setup
- virtio-scsi
  - Rich features: True scsi device, Thousands of disks per virtio-scsi device, Effective SCSI passthrough
Lifecycle of a I/O request in virtio-blk
How to improve virtio-blk performance

1) Bio based virtio-blk

2) vhost-blk
Bio-based virtio-blk: What is it (1/2)

- Two types of block device drivers
  - struct request based
    - Takes the advantages of I/O scheduler
    - Most drivers
  - struct bio based
    - Skips the I/O scheduler
    - Few drivers, e.g. Ramdisk driver
Bio-based virtio-blk: What is it (2/2)

• Vrito-blk block device driver
  • Request-based virtio-blk (original)

Do we really need the I/O scheduling twice in both guest and host? (esp. with high speed SSD device)

• Bio-based virtio-blk (new)
  • Adds bio based I/O path to virtio-blk
  • Shorten the I/O path in Guest
  • Less lock contention (q->queue_lock), lower cpu utilization
  • Higher IOPS
  • Lower Latency
Bio-based virtio-blk: Architecture

Bio

generic_make_request()

q->make_request_fn()

Req-based I/O Path

blk_queue_bio()

I/O scheduler

q->request_fn()

virtblk_request()

Bio-based I/O Path

virtblk_make_request()

virtqueue_add_buf()

virtqueue_kick()

Virtio-blk req
Bio-based virtio-blk: Performance evaluation 1

• 1) On Ramdisk device (fio test 8 vcpu, direct = 1)
  IOPS boost : 28%, 24%, 21%, 16%
  Latency improvement : 32%, 17%, 21%, 16%
Bio-based virtio-blk: Performance evaluation 2

2) On Fusion-io device (fio test 8 vcpu, direct = 1)

- IOPS boost: 11%, 11%, 13%, 10%
- Latency improvement: 10%, 10%, 12%, 10%
Bio-based virtio-blk: Performance evaluation 3

- 3) On Normal SATA device (fio test 8 vcpu, direct = 1)
  - IOPS boost: -10%, -10%, 4.4%, 0.5%
  - Latency improvement: -12%, -15%, 2.5%, 0.8%
Bio-based virtio-blk: How to use

- In mainline kernel already
  - Merged in v3.7 merge window
- No changes in host side are needed
- Kernel module parameter to turn on/off bio-base path
  - Add 'virtio_blk.use_bio=1' to kernel cmdline
  - modprobe virtio_blk use_bio=1
  - Disabled by default
Bio-based virtio-blk: Limitations

- Doesn't help with slow device on seq read/write
  - Merge is very helpful for spin disks
    - Guest+Host scheduling make the merge more aggressive
  - Merge in guest reduces the total number of request to host and reduces number of VMexit
  - The benefit of scheduling is larger than bio path gives
- Features provided by I/O Schedule is not available
  - e.g. CFQ based blkio (Proportional BW Limit)
  - Block layer based blkio (Max BW Limit) works
Bio-based virtio-blk: Future work

- Make it a feature bit in virtio-blk
  - Host can set the feature on/off
  - No need to configure inside the guest
- Make the decision of using bio-base I/O path or not automatically
  - Detect the underlay device
  - Choose the best I/O path
  - Zero configuration in both side
Vhost-blk: Overview

Host side virtio-blk implementations

- 1) QEMU current
  - QEMU global mutex: only one thread can submit I/O
  - In AIO case, io_submit() is under the global mutex
- 2) QEMU data-plane (prototype)
  - Developed by Stefan Hajnoczi
  - 1) Each virtio-blk device has a thread dedicated to handle request
  - 2) Requests are processed without going through the QEMU block layer using Linux AIO directly.
  - 3) Completion interrupts are injected via ioctl from the dedicated thread.
- 3) LKVM (aka kvm tool)
  - Using data-plane similar architecture from the very beginning
- 4) Vhost-blk (prototype)
  - vhost-blk is an in-kernel virtio-blk device accelerator, similar to vhost-net
Vhost-blk: Architecture

Guest

- vda
- virtio-blk

ioeventfd vring irqfd

Host Kernel

- vhost
- Vhost-blk
- virito req -> bio

- Generic Block Layer
- IO Scheduler
- Block Device Driver

Hardware Disks

Asias He, Red Hat Inc.
Vhost-blk: Implementation

- Using vhost infrastructure
- Send request
  - vhost-<pid> kernel thread to send request
    - created by vhost infrastructure
  - Convert guest's virtio-blk requests to bio
    - get_user_pages_fast() to convert iov based request to page
    - bio_add_page() to prepare the bio
    - set bio->bi_end_io = vhost_blk_req_done as bio completion callback
  - Use submit_bio() to submit the bio to host kernel block layer
- Complete request
  - vhost-blk-<pid> kernel thread to complete request
    - Do request and complete in parallel
    - Uses irqfd to inject interrupt to guest
Vhost-blk: Performance evaluation 1

- **LKVM-userspace v.s LKVM-vhost-blk**

Fio with libaio ioengine on Fusion IO device using LKVM

<table>
<thead>
<tr>
<th>IOPS(K)</th>
<th>userspace</th>
<th>vhost-blk</th>
<th>Improvement</th>
<th>Native</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq-read</td>
<td>107</td>
<td>121</td>
<td>+13.0%</td>
<td>127</td>
</tr>
<tr>
<td>seq-write</td>
<td>130</td>
<td>179</td>
<td>+37.6%</td>
<td>196</td>
</tr>
<tr>
<td>rnd-read</td>
<td>102</td>
<td>122</td>
<td>+19.6%</td>
<td>122</td>
</tr>
<tr>
<td>rnd-write</td>
<td>125</td>
<td>159</td>
<td>+27.0%</td>
<td>175</td>
</tr>
</tbody>
</table>
Vhost-blk: Performance evaluation 2

• **QEMU-userspace v.s QEMU-vhost-blk**

Fio with libaio ioengine on Fusion IO device using QEMU

<table>
<thead>
<tr>
<th>IOPS(K)</th>
<th>userspace</th>
<th>vhost-blk</th>
<th>Improvement</th>
<th>Native</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq-read</td>
<td>76</td>
<td>123</td>
<td>+61.0%</td>
<td>127</td>
</tr>
<tr>
<td>seq-write</td>
<td>139</td>
<td>173</td>
<td>+24.4%</td>
<td>196</td>
</tr>
<tr>
<td>rnd-read</td>
<td>73</td>
<td>120</td>
<td>+64.3%</td>
<td>122</td>
</tr>
<tr>
<td>rnd-write</td>
<td>75</td>
<td>156</td>
<td>+108.0%</td>
<td>175</td>
</tr>
</tbody>
</table>
Vhost-blk: Performance evaluation 3

- QEMU-userspace v.s QEMU-vhost-blk

IOPS (K)
fio test on 8 ramdisk based device with 4KB rand read and write

- QEMU-userspace v. QEMU-vhost-blk
  - rand-read: 63.8 vs 145.8 (2.285x)
  - rand-write: 63.7 vs 145.8 (2.288x)
Vhost-blk: Performance evaluation 4

- QEMU-userspace v.s QEMU-vhost-blk

Latency (usec)

fio test on 8 ramdisk based device with 4KB rand read and write

![Graph showing latency comparison between QEMU-userspace and QEMU-vhost-blk]

- Rand-read: QEMU-userspace 16761.56, QEMU-vhost-blk 6981.79
- Rand-write: QEMU-userspace 15316.45, QEMU-vhost-blk 6974.55

QEMU-userspace v.s QEMU-vhost-blk

2.400x

2.196x
Vhost-blk: Why

- No QEMU userspace, No QEMU global mutex
- Code path is shorter
  - Guest talks to host kernel directly
  - Host kernel BIO interface
- Save a bunch of system calls
  - `epoll_wait()` & `read()`: wait for the eventfd which guest notifies us
  - `io_submit()`: submit the aio
  - `read()`: read the aio complete eventfd
  - `io_getevents()`: reap the aio complete result
  - `ioctl()`: trigger the interrupt
- Benefits to all KVM implementation
  - e.g. Both QEMU and LKVM
Vhost-blk: How to use

• Source Code
  • KERNEL
    • git@github.com:asias/linux.git blk.vhost-blk
  • LKVM
    • git@github.com:asias/linux-kvm.git blk.vhost-blk
  • QEMU
    • git@github.com:asias/qemu.git blk.vhost-blk

• Cmdline
  $ sudo modprobe vhost-blk
  $ sudo lkvm run -d /dev/sdb,vhost
  $ sudo qemu -drive \
    file=/dev/sdb,if=virtio,cache=none,aio=native,vhost=on
Vhost-blk: Limitations & Future work

- Only support raw image format
  - No other image format support, e.g. Qcow2
- No file based image support currently
  - Lack of proper in-kernel aio interface
    - bio interface is used in current version
    - Raw block device only
    - /dev/sda, /dev/VolGroup/LogicalVolume
  - Once the work-in-progress in-kernel aio interface goes to mainline (Zach Brown and Dave Kleikamp)
    - it's easy to support raw file based image
- No migration support
Future work

- Multiqueue virtio-blk support
  - Jens' multiqueue linux block layer <-> multiqueue virtio
- More performance test and analysis
  - Different storage configurations / workload
Thanks for listening!

Comments / Questions?