Speeding up VM’s I/O sharing host's io_uring queues with guests

KVM Forum 2020

Stefano Garzarella <sgarzare@redhat.com>
Senior Software Engineer @ Red Hat
Speeding up VM's I/O sharing host's io_uring queues with guests

Agenda

- io_uring overview
  - system calls
  - queues
  - resources registration
  - polling
- QEMU and io_uring
  - virtio block with io_uring backend
- io_uring passthrough
- vhost-blk
- vdpa-blk
- Next steps
Speeding up VM’s I/O sharing host’s io_uring queues with guests

**io_uring overview**

- A new **Linux** interface for **asynchronous I/O**
  - Not only for block oriented I/O
- A **pair of rings** shared between kernel and application
  - Submission Queue (**SQ**)  
  - Completion Queue (**CQ**)  
- Three system calls
  - **io_uring_setup**(2)
  - **io_uring_register**(2)
  - **io_uring_enter**(2)
Speeding up VM’s I/O sharing host’s io_uring queues with guests

io_uring system calls

Kernel

APP

Buf

FD
io_uring system calls

- **io_uring_setup(2)**
  - setup a context for performing asynchronous I/O
Speeding up VM’s I/O sharing host’s io_uring queues with guests

io_uring system calls

- **io_uring_setup(2)**
  - setup a context for performing asynchronous I/O

- **io_uring_register(2)**
  - registers resources (e.g. user buffers, files, eventfd, personality, restrictions) in the context
io_uring system calls

- **io_uring_setup(2)**
  - setup a context for performing asynchronous I/O

- **io_uring_register(2)**
  - registers resources (e.g. user buffers, files, eventfd, personality, restrictions) in the context

- **io_uring_enter(2)**
  - initiate and/or complete asynchronous I/O
  - single system call
    - submit new operations to do
    - reap operations result
Submission and Completion Queues

- **Submission Queue (SQ)**
  - Application
    - produces SQEs (SQ Entry)
      - operation to do (opcode)
Submission and Completion Queues

- **Submission Queue (SQ)**
  - Application
    - produces SQEs (SQ Entry)
      - operation to do (opcode)
    - updates `sqe_tail`
    - invokes `io_uring_enter(2)`
Submission and Completion Queues

- Submission Queue (SQ)
  - Application
    - produces SQEs (SQ Entry)
    - operation to do (opcode)
    - updates `sqe_tail`
    - invokes `io_uring_enter(2)`
  - Kernel
    - consumes SQEs
    - updates `sqe_head`
Submission and Completion Queues

- **Submission Queue (SQ)**
  - Application
    - produces SQEs (SQ Entry)
      - operation to do (opcode)
    - updates sqe_tail
    - invokes `io_uring_enter(2)`
  - Kernel
    - consumes SQEs
    - updates sqe_head

- Kernel processes the operation
Submission and Completion Queues

- **Submission Queue (SQ)**
  - Application
    - produces SQEs (SQ Entry)
      - operation to do (opcode)
    - updates sqe_tail
    - invokes `io_uring_enter(2)`
  - Kernel
    - consumes SQEs
    - updates sqe_head
  - Kernel processes the operation

- **Completion Queue (CQ)**
  - Kernel
    - produces CQEs (CQ entry)
      - operation result (res) and user_data
    - updates cqe_tail
Submission and Completion Queues

- **Submission Queue (SQ)**
  - Application
    - produces SQEs (SQ Entry)
      - operation to do (opcode)
    - updates sqe_tail
    - invokes `io_uring_enter(2)`
  - Kernel
    - consumes SQEs
    - updates sqe_head
  - Kernel processes the operation

- **Completion Queue (CQ)**
  - Kernel
    - produces CQEs (CQ entry)
      - operation result (res) and user_data
    - updates cqe_tail
  - Application
    - consumes CQEs
    - updates cqe_head
Resources registration

- `io_uring_register(2)` system call
  - register long term references to reduce per-I/O overhead
    - `user buffers`
    - `file descriptors`
  - register `eventfd` to receive notifications of completion requests
  - `probe` `io_uring` to get information about the opcodes supported
  - register `personality` to issue SQE with certain credentials
  - register `restrictions` to install feature allowlist
  - `enable` ring processing
Polling

- **SQ polling**
  - IORING_SETUP_SQPOLL flag
  - kernel thread is created to perform submission queue polling
    - idle time is configurable
    - io_uring_enter() to wake up the kernel thread
  - Potentially application can submit and reap I/Os without doing a single system call

- **I/O polling**
  - IORING_SETUP_IOPOLL flag
  - busy-waiting for an I/O completion
    - opposed to getting notifications via an asynchronous IRQ
  - file system or block device must support polling
QEMU and io_uring

- QEMU 5.0 supports io_uring for Asynchronous I/O
- **AIO** engine
  - existing: thread, native (Linux AIO)
  - new: io_uring
    - `-drive aio=io_uring`
    - operations
      - `IORING_OP_WRITEV`
      - `IORING_OP_READV`
      - `IORING_OP_FSYNC`
- Developed by Aarushi Mehta, Julia Suvorova, and Stefan Hajnoczi
Speeding up VM’s I/O sharing host’s io_uring queues with guests

virtio block

- Two communication channels
  - **virtqueue**
    - Guest kernel <-> QEMU
  - **io_uring queues (SQ, CQ)**
    - QEMU <-> Host kernel
- QEMU “translates” requests and responses between virtqueue and io_uring queues
- We can bypass the QEMU block layer if we are not using it’s features (e.g. QCOW2)
  - **io_uring passthrough**
io_uring passthrough

- io_uring’s **SQ/CQ** are **memory mapped** in the guest

- virtio-blk driver modified to use the “fast path”
  - handle io_uring’s SQ/CQ
  - eventfd registered to inject interrupts (irqfd)

- **polling**
  - block io_poll in the guest driver to avoid IRQs in the guest
    - modified to poll CQ
  - SQPOLL enabled in the host to avoid notification from the guest (vmexit)
  - IOPOLL enabled in the host to avoid IRQs in the host
Speeding up VM's I/O sharing host's io_uring queues with guests

io_uring changes for passthrough

- eventfd disabling
  - merged upstream (Linux 5.8 - liburing 0.7)
  - patches
    - [PATCH v2 0/2] io_uring: add a CQ ring flag to enable/disable eventfd notification
      - https://lkml.org/lkml/2020/5/15/912
io_uring changes for passthrough

- **eventfd disabling**
  - merged upstream (Linux 5.8 - liburing 0.7)
  - patches
    - [PATCH v2 0/2] io_uring: add a CQ ring flag to enable/disable eventfd notification
      - https://lkml.org/lkml/2020/5/15/912

- **restrictions**
  - merged upstream (Linux 5.10 - liburing 0.8)
  - **Operations restrictions for io_uring**
    - https://lwn.net/Articles/826053/
  - patches
    - [PATCH v6 0/3] io_uring: add restrictions to support untrusted applications and guests
Speeding up VM's I/O sharing host's io_uring queues with guests

io_uring changes for passthrough

- **eventfd disabling**
  - merged upstream (Linux 5.8 - liburing 0.7)
  - patches
    - [PATCH v2 0/2] io_uring: add a CQ ring flag to enable/disable eventfd notification
      - https://lkml.org/lkml/2020/5/15/912

- **restrictions**
  - merged upstream (Linux 5.10 - liburing 0.8)
  - **Operations restrictions for io_uring**
    - https://lwn.net/Articles/826053/
  - patches
    - [PATCH v6 0/3] io_uring: add restrictions to support untrusted applications and guests

- **memory translation**
  - Guest PA <-> Host VA
  - to do
io_uring restrictions

- Install feature **allowlist** on an io_uring context
  - only operations defined in the allowlist can be executed
  - new io_uring features do not accidentally become available

- How to install restriction?
  - using the new `io_uring_register(2)` opcode:
    ```
    IOURING_REGISTER_RESTRICTIONS
    ```
  - rings must start disabled (`IORING_SETUP_R_DISABLED`)
    - enabled with `IORING_REGISTER_ENABLE_RINGS`

- What we can restrict?
  - `io_uring_register(2)` op. codes
  - SQE op. codes
  - SQE flags for each operation
    - allowed
    - required
Speeding up VM’s I/O sharing host’s io_uring queues with guests

io_uring passthrough PoC performance

fio (rw=randread, bs=4k, ioengine=io_uring, hipri)

- QEMU virtio-blk w/ iothread
- io_uring passthrough PoC
- bare-metal

<table>
<thead>
<tr>
<th>iodepth</th>
<th>1</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>QEMU</td>
<td>73</td>
<td>104</td>
<td>119</td>
<td></td>
</tr>
<tr>
<td>PoC</td>
<td>532</td>
<td>550</td>
<td>577</td>
<td>585</td>
</tr>
<tr>
<td>Bare-metal</td>
<td>306</td>
<td>312</td>
<td>585</td>
<td>585</td>
</tr>
</tbody>
</table>
vhost block

- An alternative to io-uring passthrough to have a single communication channel
  - virtqueue shared between guest and host kernels
- Some implementations was published upstream but never merged
  - Asias He’s vhost-blk [2012]
    - https://lore.kernel.org/patchwork/patch/344823/
      - bio API
    - Vitaly Mayatskih’s vhost-blk [2018]
      - https://patchwork.kernel.org/cover/10665995/
      - VFS API
- vhost-blk improved adding polling
  - VQ polling
  - I/O polling
Speeding up VM's I/O sharing host's io_uring queues with guests

io_uring passthrough vs vhost-blk

![Bar chart showing performance comparison between io_uring passthrough and vhost-blk]

- vhost-blk
- vhost-blk + VQ poll
- vhost-blk + VQ + IO poll
- vhost-blk + VQ + IO + SQ poll
- io_uring passthrough PoC
- bare-metal
Speeding up VM’s I/O sharing host’s io_uring queues with guests

vDPA block software device

- Similar to vhost-blk but using the new vDPA framework
  - virtio Data Path Acceleration

- PROs
  - **unified software stack** (host, guest, QEMU, hardware)
    - hardware implementation will be available
  - more control than vhost on device lifecycle
  - guest pages pinned
    - copy_in/to() not needed

- CONs
  - guest pages pinned
    - no memory overcommit
  - io-uring passthrough already supports IO polling, SQ polling, VFS integrations, etc.

- Work in progress
Next Steps

- vDPA-blk software device
  - simulator
  - QEMU support
  - Linux vDPA driver with VFS integration

- virtio-blk driver optimizations
  - blk io_poll upstream

- io-uring passthrough
  - memory translation
Thank you!

Stefano Garzarella <sgarzare@redhat.com>

Blog: https://stefano-garzarella.github.io/

IRC: sgarzare on #qemu irc.oftc.net

linkedin.com/company/red-hat facebook.com/redhatinc
youtube.com/user/RedHatVideos twitter.com/RedHat