redhat.

UBLK

HIGH PERFORMANCE GENERIC USERSPACE BLOCK DEVICE

Ming Lei <ming.lei@redhat.com>, platform storage, Red Hat

What is UBLK

- High performance generic userspace block device
- Goals:

  - High performance

  - Expose generic block device, and support all kinds of block/queue settings/parameters

  - Move all block IO logic in userspace

  - Implement userspace target/backend easily

What can UBLK help

- simplify driver development by moving logic to userspace
  - application: more program languages, more libs, more debug tools, more developers, …
- performance evaluate

- simulate block device quickly
  - generic interface for setting block parameters/settings
  - such, easy to simulate one zoned, compressed, encrypted device,...

Background

- NBD, merged to linux kernel 2.1.15 in 1997

  - expose nbd device node, socket communication

- VDUSE, merged to linux kernel 5.5 in 2021

  - expose as virtio_blk, io command via traditional read/write on char device

- UBLK, merged to linux kernel 6.0 in 2022, io command via io_uring pt cmd
- BDUS: 2021 https://dl.acm.org/doi/10.1145/3456727.3463768
- BUSE: 2021 https://github.com/acozzette/BUSE
- DM-USER: 2020 https://lwn.net/Articles/838986/
- More...

UBLK framework

- ublk drv
  - merged linux kernel v6.0
  - IO command communication & pages copy(ublk block request and ublksrv)
  - admin task(add/del/list/recovery device)
- ublk server: ublksrv (userspace)
  - https://github.com/ming1/ubdsrv
  - libublksrv
  - ublksrv generic target/backend
  - ublksrv target/backend
  - preferred io handling: io_uring, but support other kind of aio handling too
  - so far supported targets(null, loop, qcow2, nbdublk, …)

IO command communication

- IO descriptor
  - each IO has unique per-queue tag

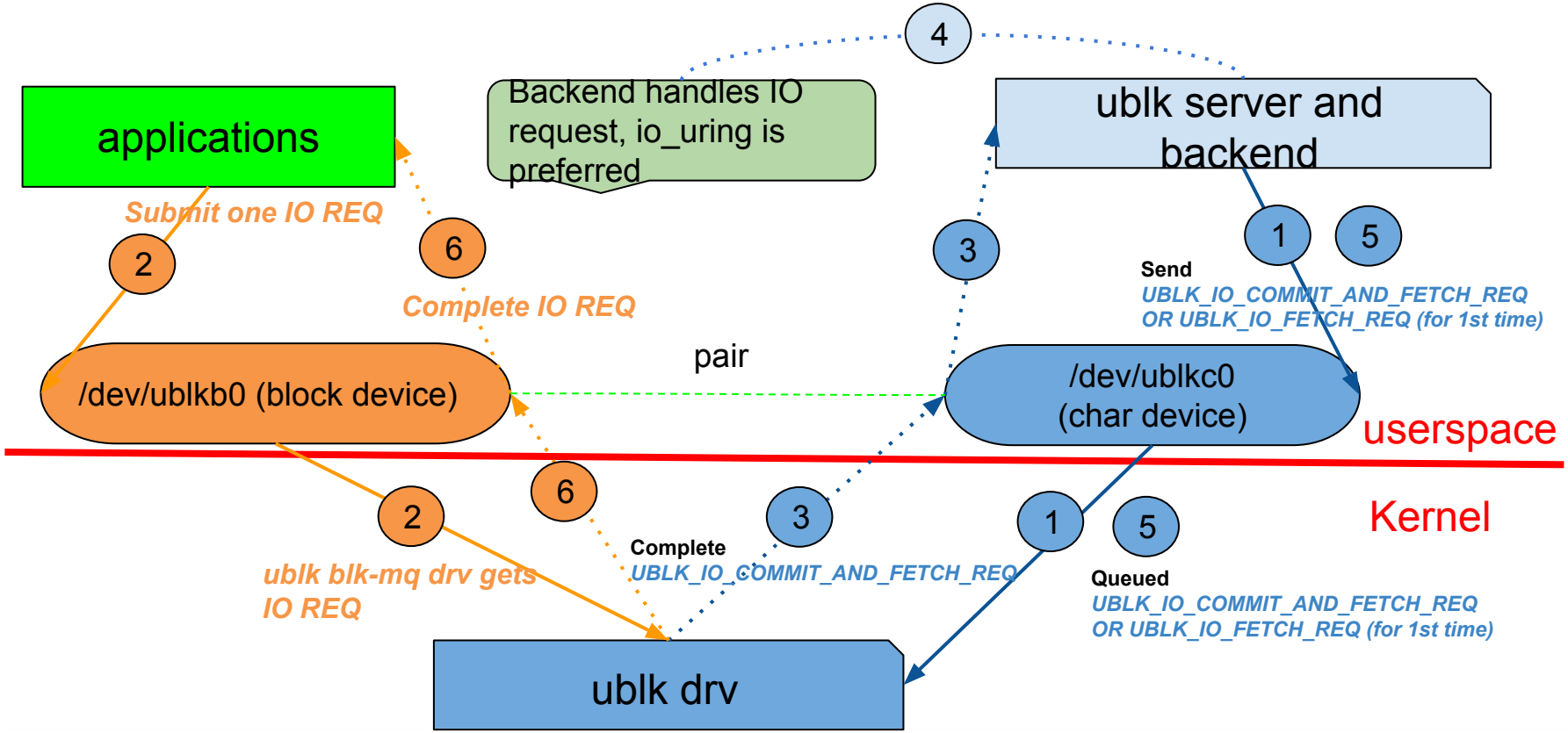
  - IO descriptor is written to shared/mmapped area which can be indexed by io tag, read-only for ublk server, and write-only for ublk drv

- UBLK_IO_FETCH_REQ(io_uring pt cmd)
  - sent once from ublk server for setting up IO communication

- UBLK_IO_COMMIT_AND_FETCH_REQ (io_uring pt cmd)
  - When ublk IO req comes, the issued *_FETCH_REQ is completed

  - After the IO is handled by ublk server, this command is issued to ublk drv for both committing previous IO result and start to fetch new request

IO command communication



ublk-qcow2

- Basic functions
  - so far only support read/write, not hard to support compression
  - some work is needed for snapshot
- Design
  - OOP, implemented by C++, need c++20 for coroutine support, basically rely on libstdc++ only
  - each IO is handled in one standalone stackless coroutine context, and io tag is coroutine context id too
  - meta data loading is done as foreground IO in current IO handling co context
  - meta data flushing is in way of soft update as background IO, need extra tags (extra coroutine context)
  - both data and meta IOs are handled by io_uring

UBLK performance

- ublk-null:
  - **single queue, single job**, bs: 4k, dio, libaio/io_uring, IOPS can reach 1.2M IOPS
  - **create /dev/ublkb0:** ./ublk add -t null -n 0
  - **switch elevator to null:** echo none > /sys/block/ublkb0/queue/scheduler
  - pull fio: https://github.com/axboe/fio.git && configure && make
  - t/io_uring -p 0 -B 1 -F 1 -T 1 -X 1 /dev/ublkb0
  - reach **~1.2M IOPS** on VM created in my laptop(T590), related with memory bandwidth in the machine
- ublk-loop: IOPS is basically same level with kernel loop with –directio=on
  - https://lwn.net/Articles/903855/
  - https://lore.kernel.org/all/20220713140711.97356-1-ming.lei@redhat.com/

UBLK performance

- ublk vs. qemu-nbd, by comparing qcow2 target
  - > ~3X IOPS in random IO test
  - > 30% improvenet in sequential big chunk IO
  - https://lore.kernel.org/lkml/Yza1u1KfKa7ycQm0@T590/

- ublk vs. vduse:
  - 1job 1 io depth: ½ latency of vduse over null_blk
  - 4job 128 io depth: ~3X IOPS of vduse
  - https://lore.kernel.org/lkml/50827796-af93-4af5-4121-dc13c31a67fc@linux.alibaba.com/

Why does UBLK perform so well

- High performance io uring passthrough command
  - io_uring pt cmd is proved as efficient, even more than io_uring over block IO
  - IO command is submitted beforehand, minimize io command forward latency
  - IO command multiplexing: one command covers both result committing and fetching new req
- target/backend IO handling by io_uring too
  - share same io_uring context, maximize io batching in single syscall
- IO handle efficiently
  - each IO has its unique tag, submit io command/allocate resource beforehand
  - work together with per-IO stackless coroutine, minimize context switch and maximize IO parallelization
  - meantime simplify IO handling development

Future development

- Container-ware ublk
  - to be unprivileged, actually both io command submission & completion & handling are done in user task
- Zero copy for big chunk IO
  - is it possible to avoid the single pages copy for big chunk IO?
- All kinds of performance improvement
  - sequential big chunk IO has improvement space, get user pages latency
  - batching io handling for /dev/ublkb*
- Cross platform
  - io_uring is supported by windows 11
- More targets/backends
  - nbd, zoned, compressed, rbd, iscsi, nvme-tcp, …
  - make full use of io_uring's high performance advantage

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos