redhat.

UBLK

HIGH PERFORMANCE GENERIC USERSPACE BLOCK DEVICE

Ming Lei <ming.lei@redhat.com>, 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

Motivations

- can be written many programming languages.
- can use libraries that are not available in the kernel.
- can be debugged with tools familiar to application developers.
- Crashes do not kernel panic the machine.
- Bugs are likely to have a lower security impact than bugs in kernel
- can be installed and updated independently of the kernel.
- can be used to simulate block device easily with user specified parameters/setting for test/debug purpose

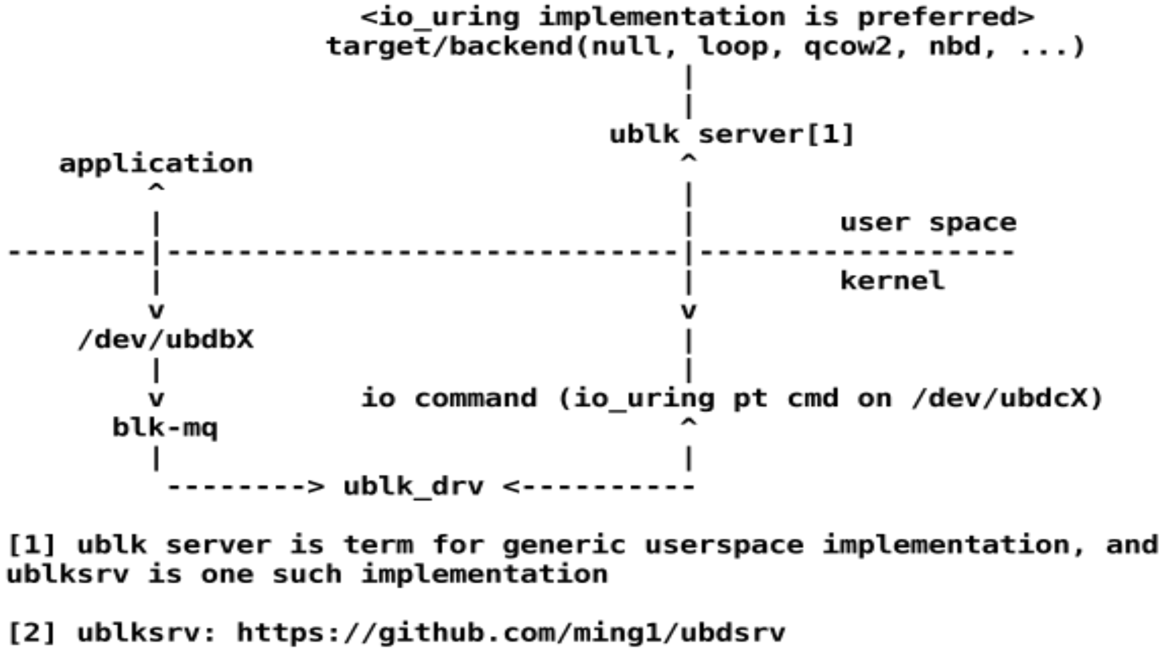
Background

- 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



```
                    <io_uring implementation is preferred>
                  target/backend(null, loop, qcow2, nbd, ...)
                                      |
                                      |
                                  ublk server[1]
   application                        ^
        ^                             |
        |                             |        user space
--------|-----------------------------|-------------------
        |                             |        kernel
        v                             v
    /dev/ubdbX                        |
        |                             |
        v           io command (io_uring pt cmd on /dev/ubdcX)
      blk-mq                          ^
        |                             |
         --------> ublk_drv <----------

[1] ublk server is term for generic userspace implementation, and
ublksrv is one such implementation

[2] ublksrv: https://github.com/ming1/ubdsrv
```

UBLK framework

- ublk drv

  - in linux kernel v6.0

- ublk server: ublksrv

  - libublksrv

  - ublksrv generic target/backend

  - ublksrv target/backend

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)

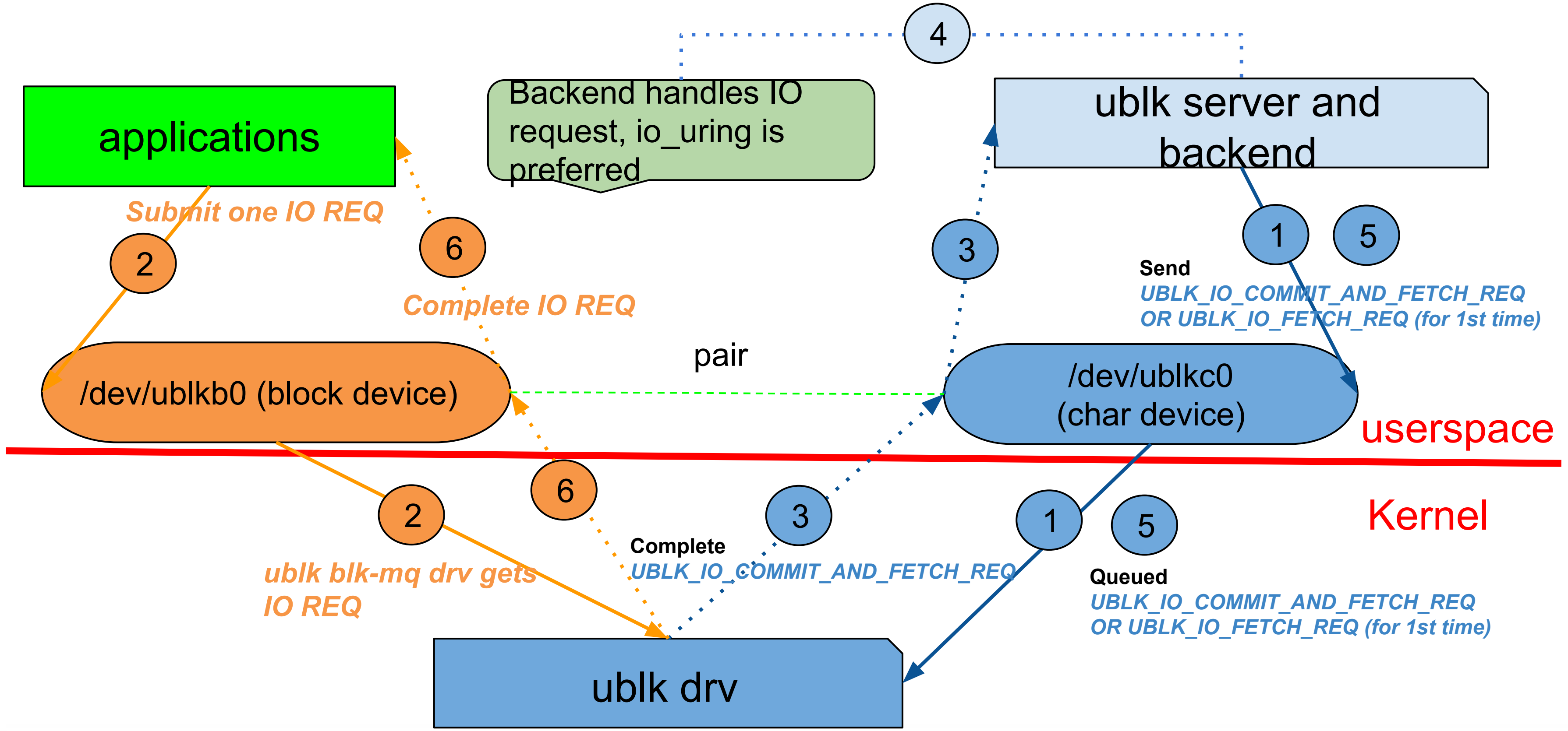
  - sent once from ublk server for setting up IO communication

- 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


4
Backend handles IO request, io_uring is preferred
applications
ublk server and backend
Submit one IO REQ
2
6
3
1
5
Send
UBLK_IO_COMMIT_AND_FETCH_REQ
OR UBLK_IO_FETCH_REQ (for 1st time)
Complete IO REQ
pair
/dev/ublkb0 (block device)
/dev/ublkc0 (char device)
userspace
2
6
3
1
5
Kernel
Complete
UBLK_IO_COMMIT_AND_FETCH_REQ
ublk blk-mq drv gets IO REQ
Queued
UBLK_IO_COMMIT_AND_FETCH_REQ
OR UBLK_IO_FETCH_REQ (for 1st time)
ublk drv

ublksrv

- Userspace part:
  - https://github.com/ming1/ubdsrv
  - in Fedora Rawhide
  - add / del / list / recovery device
  - libublksrv: provide generic interface for standalone target development, such nbdublk,
  - IO target/backend implementation
- Supported targets(early stage)

  null, loop, qcow2(basic read/write)
- Preferred target io handling

  io_uring for getting top performance

UBLK performance

- ublk-loop: IOPS is close to kernel loop with –directio=on

  - https://lwn.net/Articles/903855/

  - https://lore.kernel.org/all/20220713140711.97356-1-ming.lei@redhat.com/

- ublk vs. qemu-nbd, by comparing qcow2 target

  - > 2~3X IOPS in random IO test

  - 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

- 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

Questions

- Email / github

  - ming.lei@redhat.com
  - https://github.com/ming1/ubdsrv.git

- Welcome to participate in ublk development redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos