Practical and efficient out-of-process storage backends

Kevin Wolf <kwolf@redhat.com>

KVM Forum 2024

Red Hat

Background

Why out-of-process backends?

Short recap of the KVM Forum 2022 talk

- Isolation for improved security
- Separation of concerns (VMs vs. storage)
- Offline block jobs
- Sharing a backing chain between multiple VMs
- Sharing a CPU for polling
- Sharing a single disk between multiple VMs

KubeVirt and storage backends

Our specific motivation currently

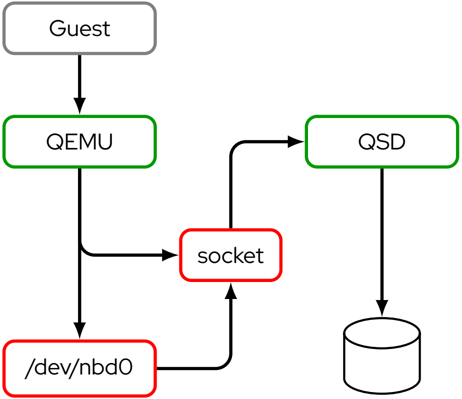
KubeVirt considers storage Someone Else's Problem

- CSI plugins provide access to storage on Kubernetes
- Idea: HW vendor provides a CSI driver for storage operations
- Practice: The CSI driver often doesn't fulfill the requirements
- QEMU already implements the functionality in software, so we should just expose it (to VMs *and* normal containers!)

What are our options?

NBD

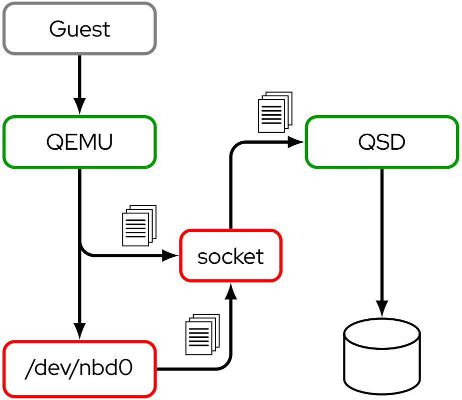
The obvious and familiar solution

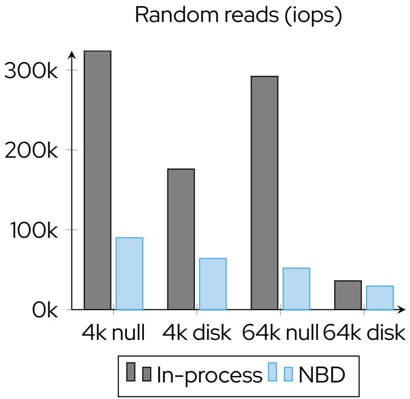


- All building blocks have existed for a long time
- A single solution that covers the network, too (migration)
- Can be attached as a host block device

Problems with NBD

It would have been too easy





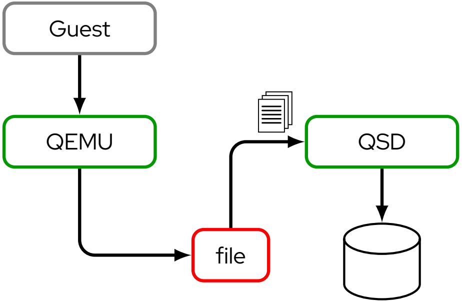
Problems with NBD

It would have been too easy

- Copying everything through a socket
- Both QEMU and an external process in the I/O path
- How to access it?
  - A socket is not really suitable for Kubernetes CSI
  - Kernel NBD client for block devices requires privileges and doesn't support all features

FUSE

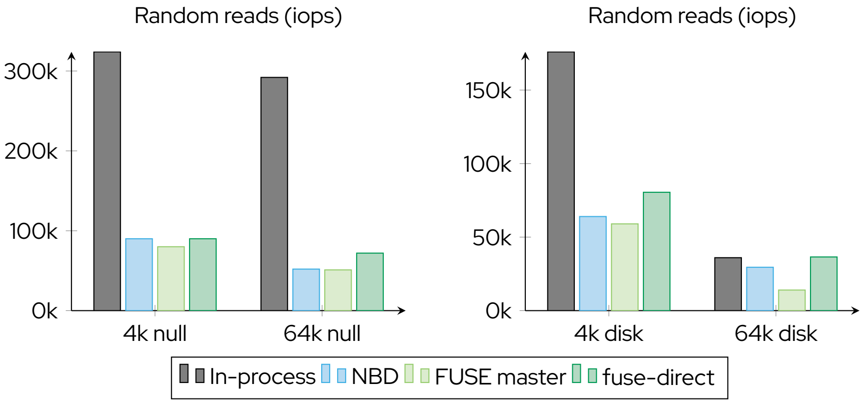
Maybe the least discussed export type



- Only one copy
  - Could maybe achieve zero-copy for common cases with splicing
- Still both QEMU and QSD in the I/O path
- Export shows up as a regular file
- Works as a regular user

FUSE performance

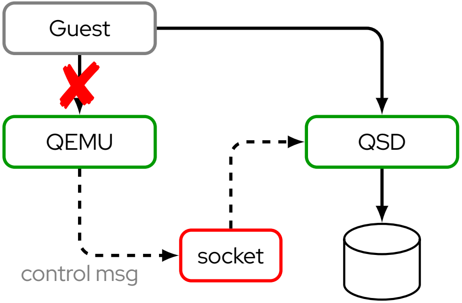
Better than NBD anyway


Random reads (iops)

Random reads (iops)

vhost-user-blk

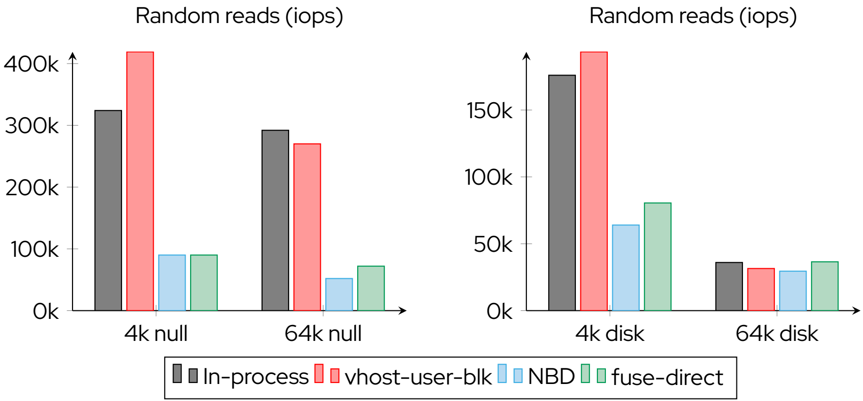
The polar opposite of NBD



- Zero copy: Guest RAM is shared memory
- QEMU not involved in the I/O path
- No privileges required

vhost-user-blk performance

Should be the same as in-process in theory



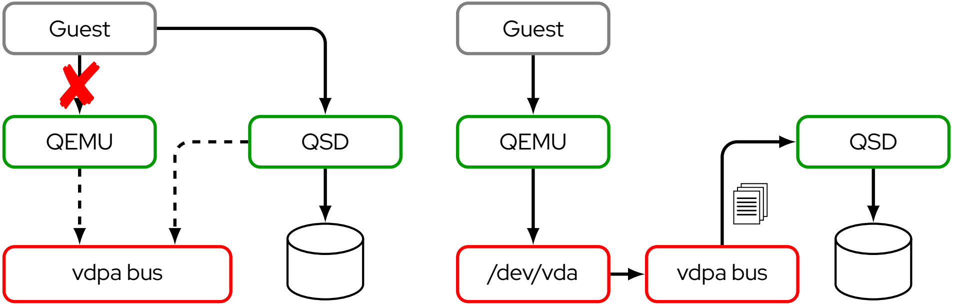
Problems with vhost-user-blk

It comes at a cost

- Requires a socket, no way to use a block device or regular file
- Requires the guest RAM to be shared memory
  - Conflicts with features like KSM, memory ballooning, etc.
- Works optimally only if the guest uses virtio-blk devices
  - libblkio enables other devices, but then QEMU has to be in the I/O path again
- Quite different to manage compared to normal block backends and management tools don't support it yet

vdpa-blk

The best of both worlds (but not at the same time)



With vhost-vdpa driver

With virtio-vdpa driver

vdpa-blk characteristics

The best of both worlds (but not at the same time)

- One export type to cover high performance and block devices
- vhost-vdpa works much like vhost-user-blk
- virtio-vdpa is similar to the NBD kernel client or FUSE
- Both modes requires privileges
- Kernel support is required and not enabled in all distros yet

What if we could switch?

This may or may not be realistic…

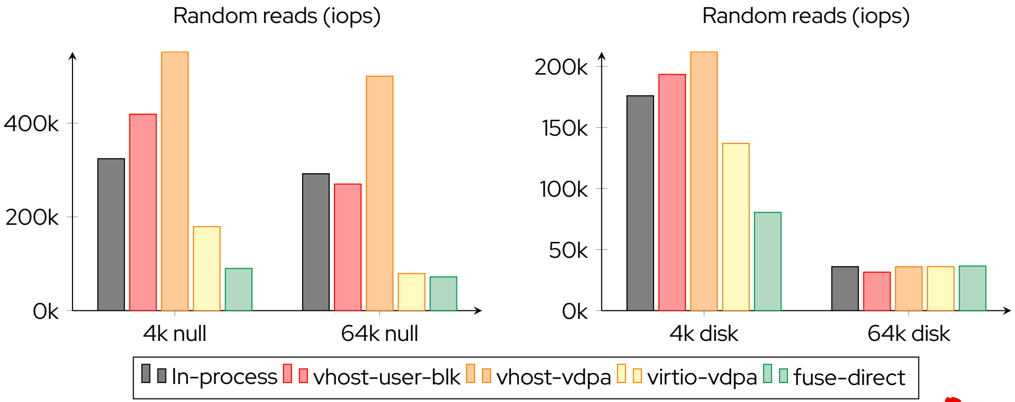
Imagine /dev/vda supported an ioctl INTO_VHOST:

- The block device becomes inactive (e.g. returns -EBUSY)
- The vdpa device is transferred to vhost-vdpa
- The ioctl returns a file descriptor for the vhost chardev

Then we would have a block device for generic use cases, and could still use vhost-vdpa where performance would improve

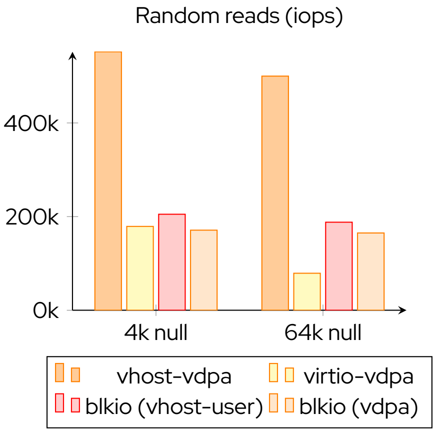
vdpa-blk performance

Not only vhost-user-blk can do better than baseline!



Lifting some vhost restrictions with libblkio

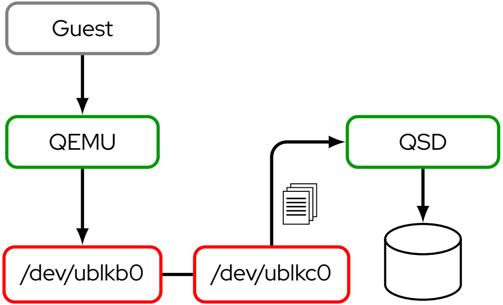
If you want vhost, but still not only virtio-blk



- libblkio allows attaching vhost-vdpa as a normal QEMU block device
- Can use any guest device
- Performance is not worse than virtio-vdpa

ublk

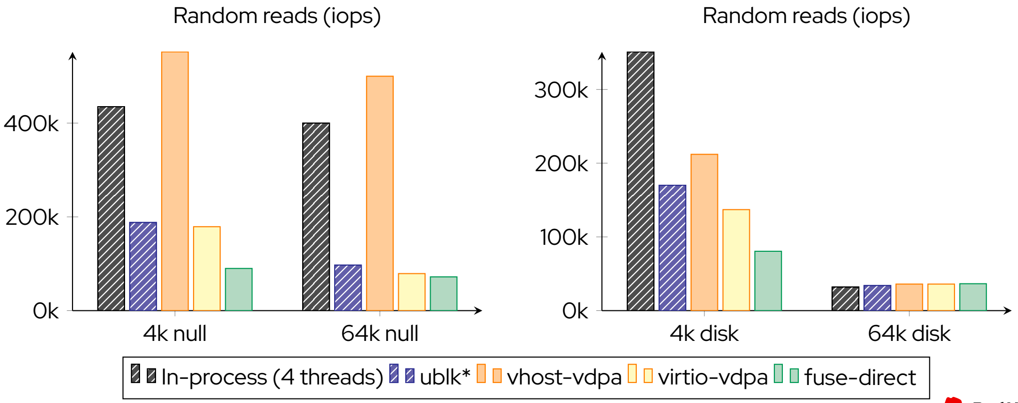
Back to host block devices



- I/O path is similar to virtio-vdpa
- Not implemented in QEMU yet
- Kernel driver isn't very mature yet

ublk performance

Yes, it's comparing apples and oranges



*using ubdsrv, not a QEMU export (⟶ one thread per queue)

Conclusion

Can I mix and match?

- For each property we want, there is an export type that has it.
  But there is nothing that combines all of them.
- In particular, zero-copy seems important.
  But sharing memory and giving access to it is painful.
- If privileges are not a problem, vDPA seems to be a good all-purpose export, but it still requires a trade-off when choosing the driver.

Can we do better?

What export to improve?

Where do we see potential for improvement?

- We almost certainly want a "normal" block device or file
    - Limitations of shared memory seem hard to overcome
    - Bypassing QEMU's block layer makes management very different
- We have no way to remove QEMU from the I/O path then
    - (Except maybe something like io_uring passthrough?)
- We can try to reduce overhead on the QSD side

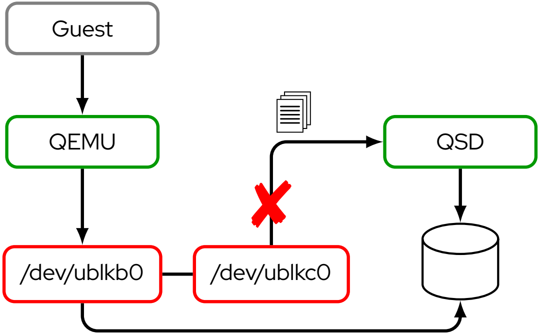
ublk: SQE groups for zero-copy

Payload? Who needs that?

- Userspace often only forwards the payload
- Copy only the request metadata to the userspace daemon
- New io_uring command that the daemon can use to reuse in-kernel buffer for its own requests to backing storage
- v6 patch series by Ming Lei on io-uring/linux-block mailing lists

ublk: Cache mappings in the kernel

Why bother with calling into userspace at all?



- In common cases, image formats only map between offsets
- Why not cut out the userspace daemon instead of QEMU?
- Prototype showed improved iops on file

ublk: eBPF for handling requests in the kernel

Avoid userspace even harder

- Instead of just mappings, allow arbitrary logic
- If eBPF code handles the request, no need to involve userspace
- May allow to do additional things without a context switch (e.g. updating dirty bitmaps)
- Ming Lei wrote some early prototype code

Random other observations

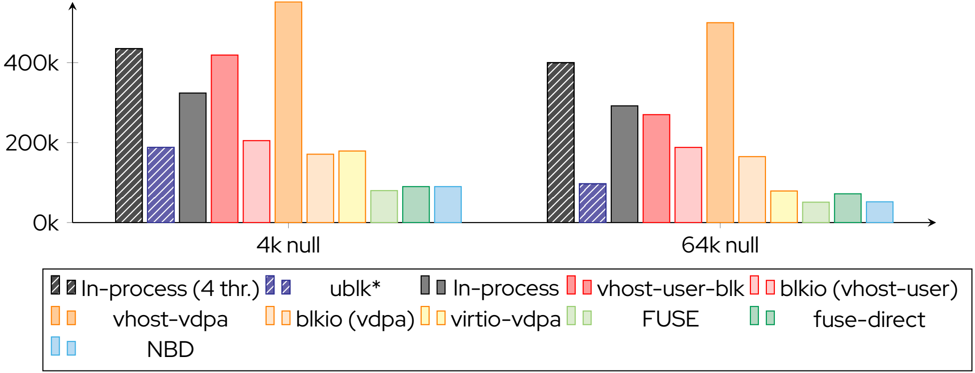
Other exports can still improve, too

- The vhost-user-blk export has a hard-coded queue size of 128.
  With vDPA, it's configurable and 256 by default.
- ublk benefits from allowing multiple I/O threads.
  Exports should implement iothread-vq-mapping like virtio-blk.
- Something seems to be wrong with the NBD implementation.
  A slower backend should hide its overhead, but it only gets worse.

Bonus data

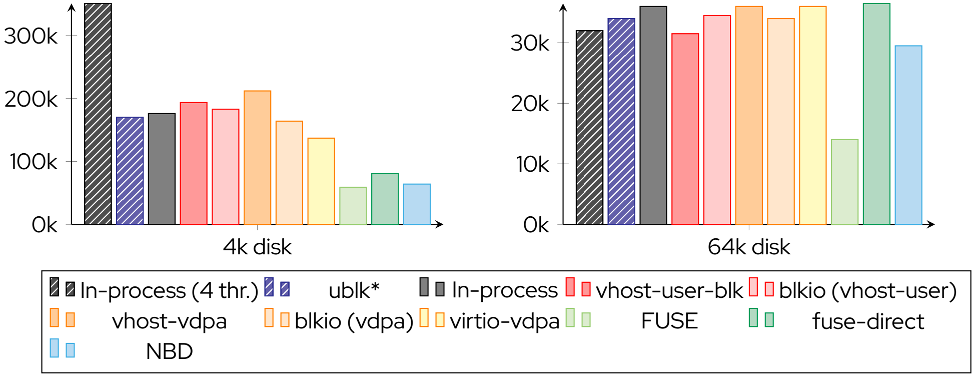
Null device (all exports)

4 vCPUs, 4 virtqueues, 1 iothread, 16 GB null device



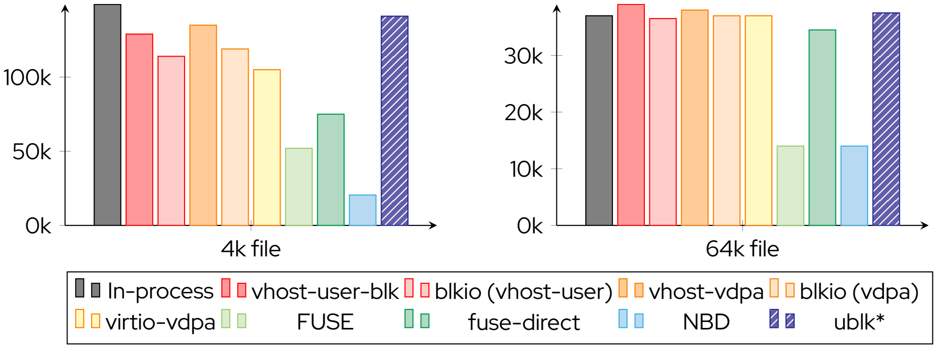
Disk backed (all exports)

4 vCPUs, 4 virtqueues, 1 iothread, 16 GB partition on NVMe



File backed (all exports)

4 vCPUs, 4 virtqueues, 1 iothread, 16 GB file on XFS+LVM+LUKS



Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

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