Red Hat Enterprise Linux

virtio-fs

A Shared File System for Virtual Machines

Dr. David A. Gilbert
<dgilbert@redhat.com>

Stefan Hajnoczi
<stefanha@redhat.com>

Miklos Szeredi
<mszeredi@redhat.com>

Vivek Goyal
<vgoyal@redhat.com>

Red Hat

What is virtio-fs?

virtio-fs

Shares a host directory tree with the guest

Desired semantics:

- POSIX file system plus modern extensions
- Concurrent access from multiple guests
- Local file system semantics (coherency) where possible

Started in 2018, now being tested and developed by a growing community:
https://virtio-fs.gitlab.io/

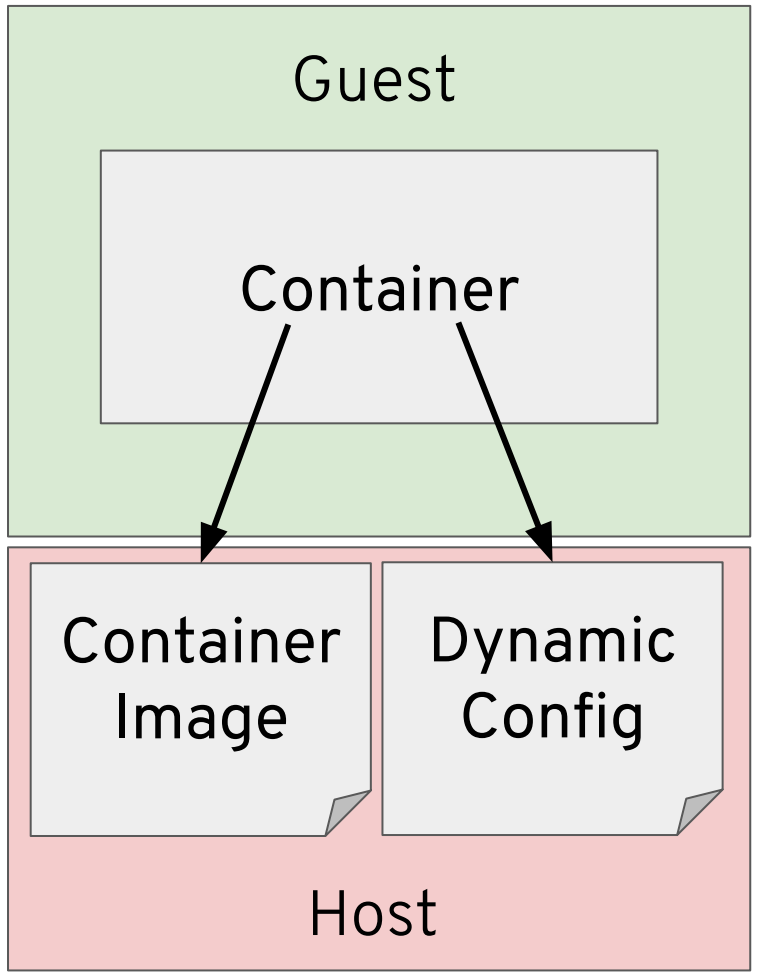
Use case: Lightweight VMs and container VMs

Micro VMs, Kata Containers, Function as a service (FaaS)

Requirements:

- Fast boot time - Avoid copying file contents into guest during boot
- Low memory overhead - Share read-only file contents between all guests
- Access to files from host - Both read and write access

Try it: virtio-fs has been available in Kata Containers since 1.7!

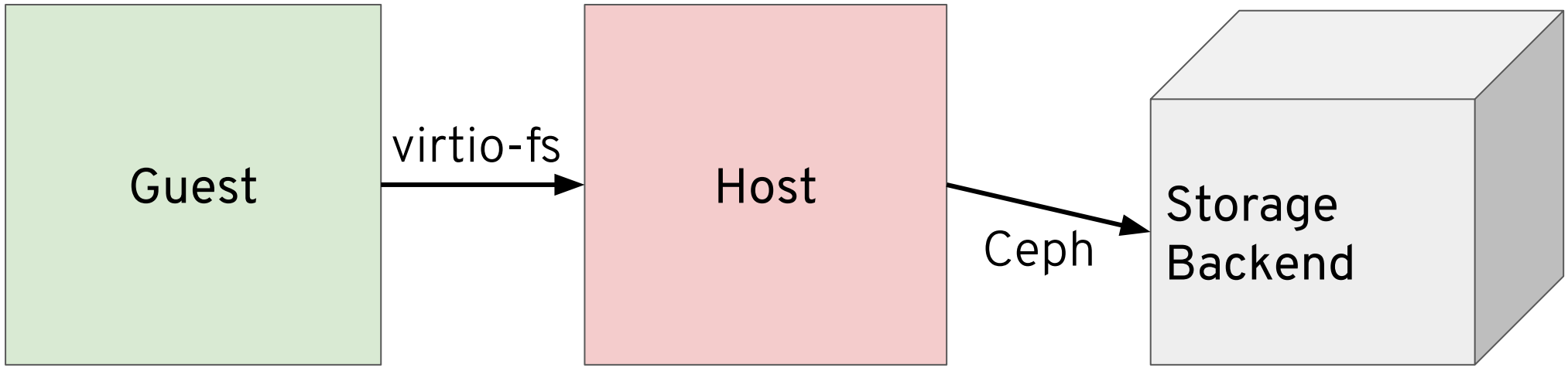


Use case: File system-as-a-service

Provide access to NFS, Gluster, Ceph, etc storage

Requirements:

- No guest network access - Isolate guest from storage network for security
- Hide storage details - Change storage technology without affecting guests

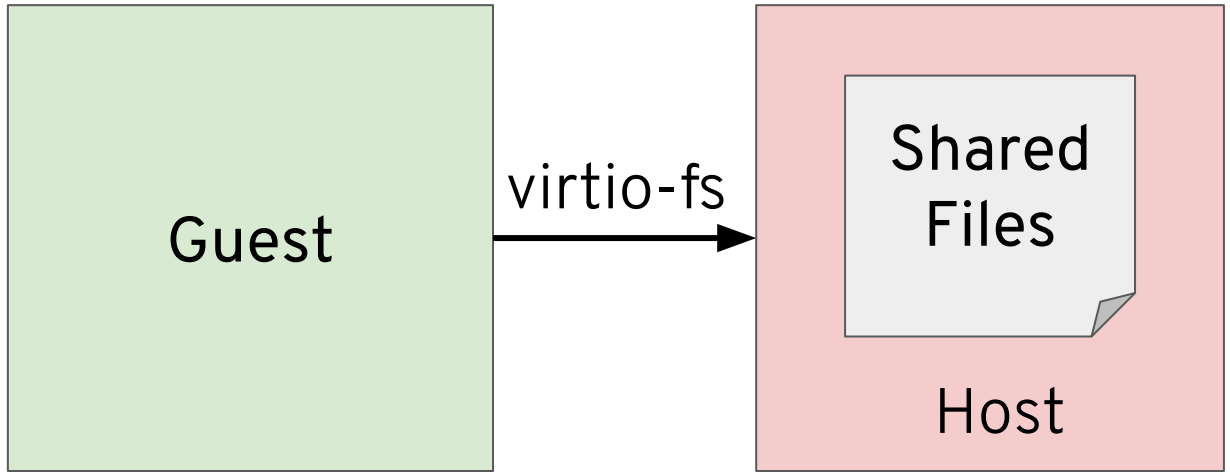


Use case: Traditional file sharing

Share a host directory with the guest

Requirements:

- No manual setup - Easy to implement as management tool command
- Add/remove directories at will - Hotplug support



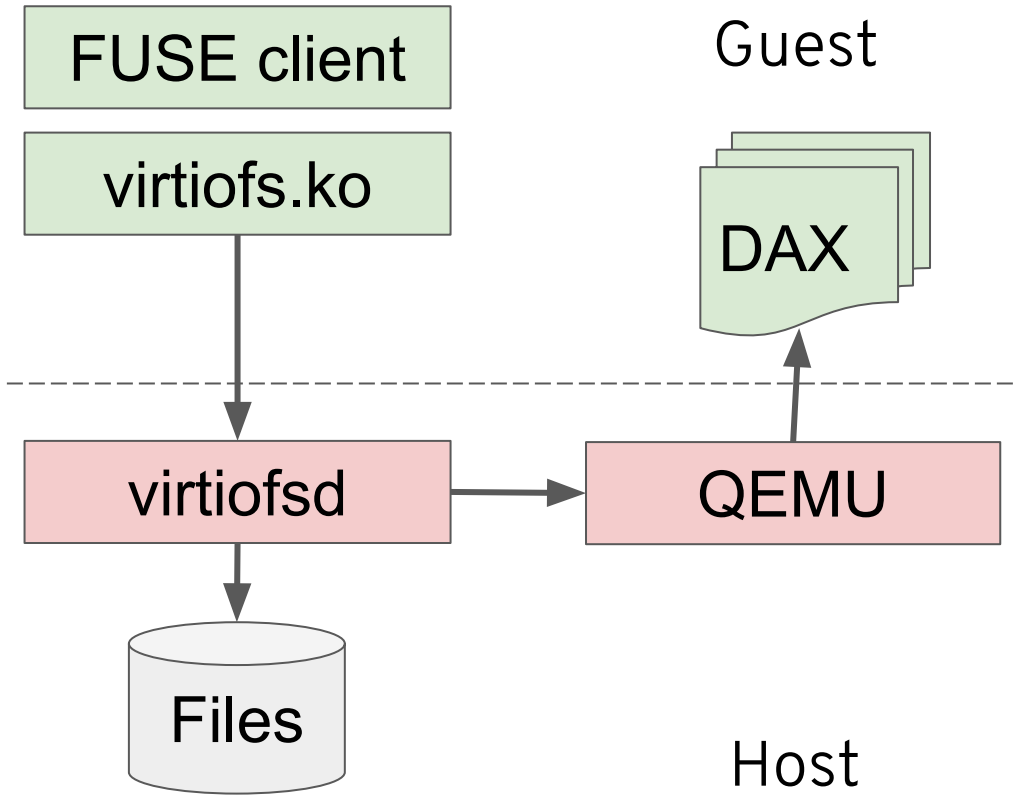
Why virtio-fs?

QEMU/KVM needs a production-quality shared file system

Active development of virtio-9p ceased in 2012

Can we do better than network file systems by taking advantage of co-location between VM and hypervisor?

Architecture

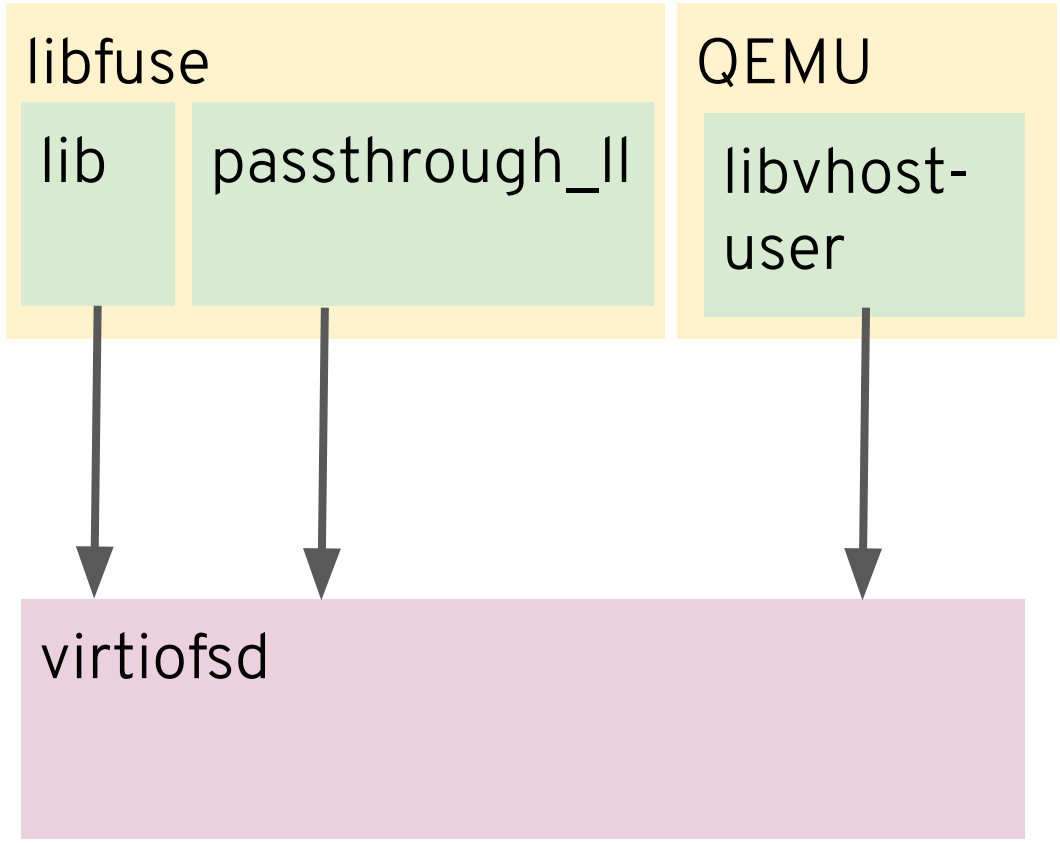


Based on FUSE (but not compatible!)

Sandboxed virtiofsd vhost-user backend

QEMU assists in DAX host page cache sharing

Supports local and remote storage



Virtiofsd

Vhost-user backend consisting of:

Subset of libfuse library

- Modified (not ABI compatible)

Libvhost-user

- Provides the basis for the transport

Passthrough_ll

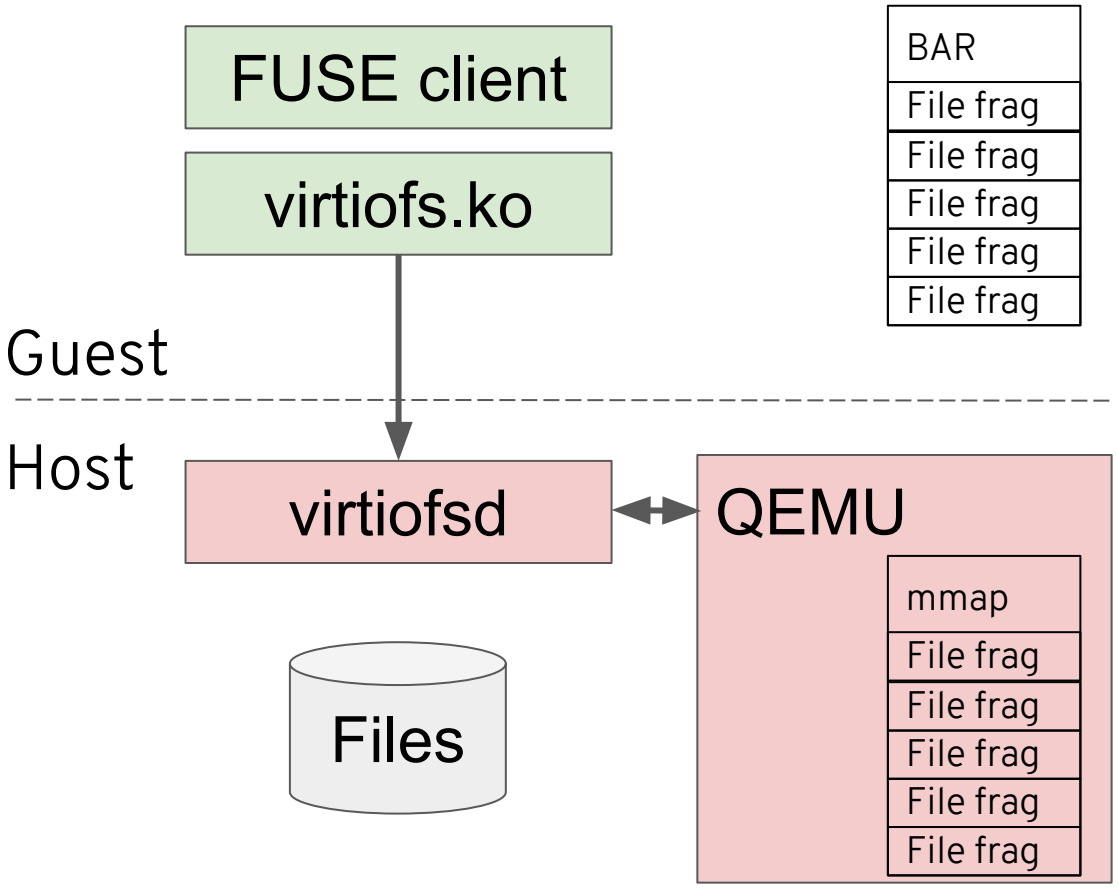
- Loopback FUSE file system

Thread per queue + thread pool for servicing requests

Potential daemons

- Other filesystems
  - Instead of POSIX could access network FS directly (e.g. gluster/ceph/nfs) Rather than through the kernel
  - Or block storage via userspace (see next talk!)
- Other implementations
  - Rust implementation being considered (crosvm but not vhost-user)

DAX



- Guest driver requests (un)mapping by special fuse message
- Mappings appear in PCI-BAR at guest specified offset
- BAR appears almost like DAX device in guest
  - But is only a window into the fs; not the whole fs
- Virtiofsd opens files, QEMU mmap's them

Differences from normal FUSE

- virtio-fs device instead of /dev/fuse
  - FUSE messages are transported over the virtio-fs device
  - Needs vhost-user-fs support in FUSE daemon, can't use libfuse daemons
- Security inversion
  - Traditional FUSE: Kernel is trusted, daemon is untrusted user program
  - Virtio-fs: Kernel is the untrusted guest, daemon cannot trust it
    - Additional checks added to libfuse/passthrough_ll.c
- Reboots
  - Traditional FUSE: Daemon runs under the kernel, reboots restart daemon
  - Virtio-fs:
    - Must handle a guest reboot, or mount/umount (but reset state)

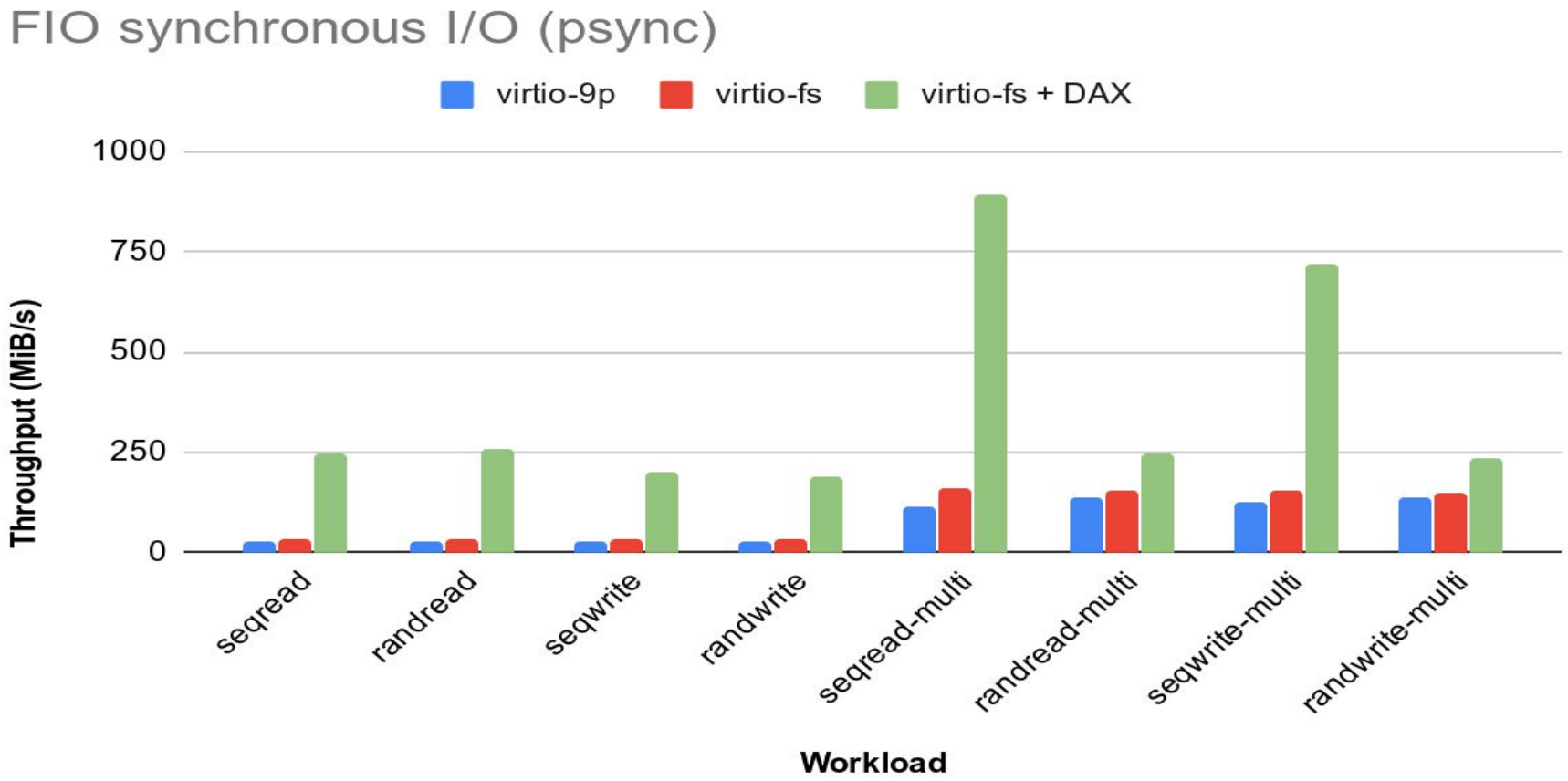
How to try it

Run virtio-fs with QEMU: https://virtio-fs.gitlab.io/howto-qemu.html

```
(host)# virtiofsd --socket-path=/tmp/vhost-fs.sock \
                  -o source=/path/to/shared/dir
(host)$ qemu … \
        -chardev socket,id=char0,path=/tmp/vhost-fs.sock
        -device vhost-user-fs-pci,chardev=char0,tag=myfs
(guest)# mount -t virtiofs myfs /mnt
```

Or try it with Kata Containers: https://red.ht/kata-virtio-fs

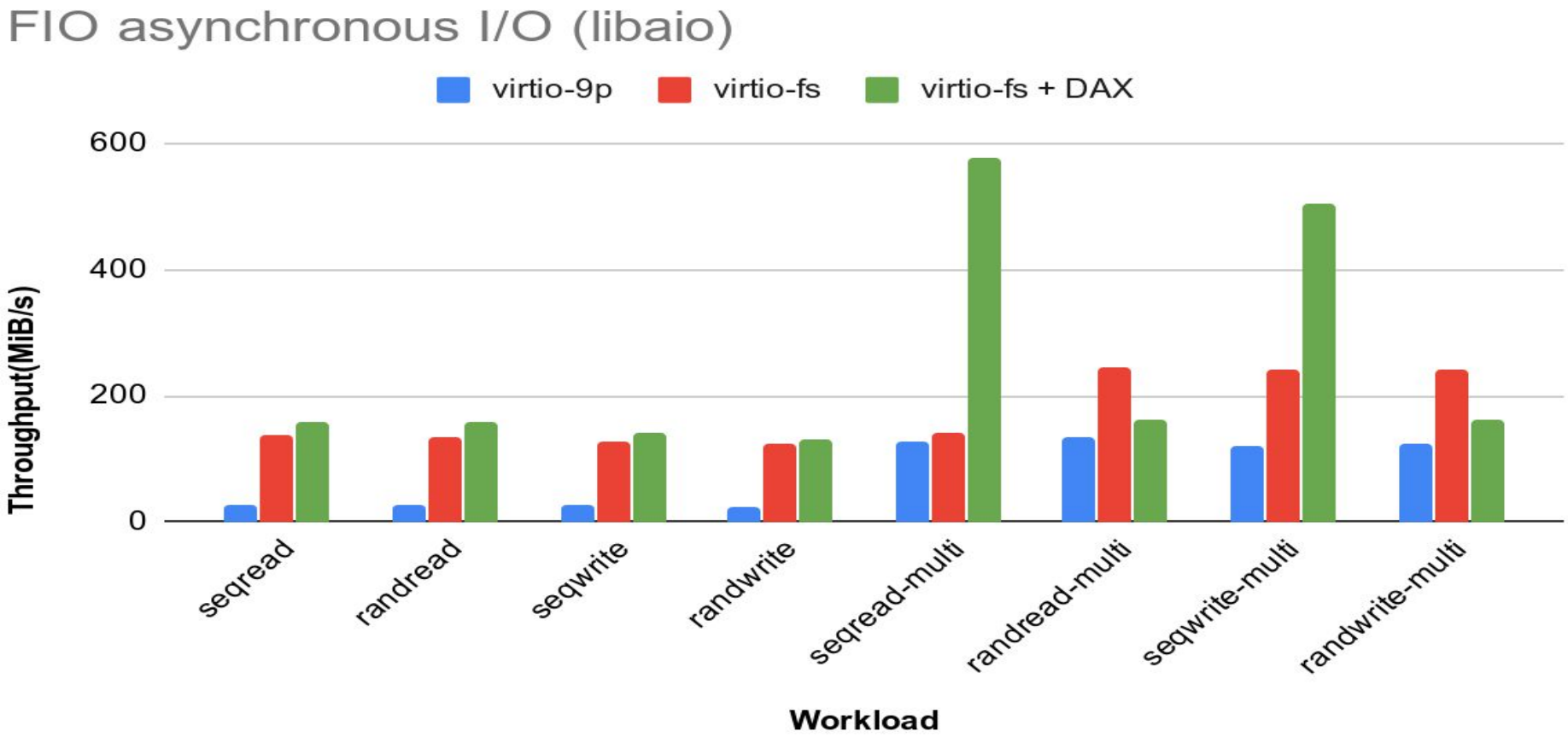
Virtio-fs vs virtio-9p Benchmark



Source:
https://lore.kernel.org/linux-fsdevel/20190821173742.24574-1-vgoyal@redhat.com/

Red Hat

virtio-fs vs virtio-9p Benchmark Contd.

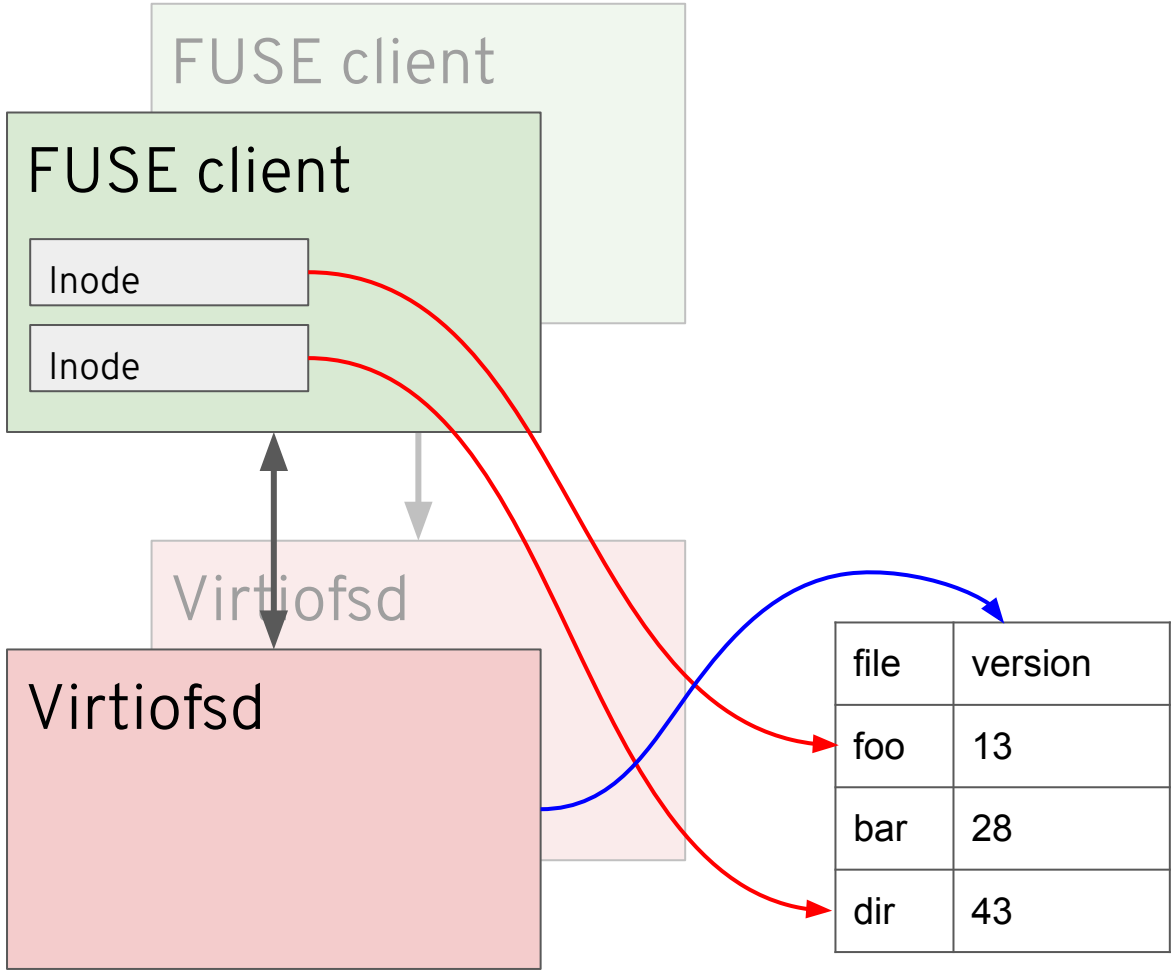


Source: https://lore.kernel.org/linux-fsdevel/20190821173742.24574-1-vgoyal@redhat.com/

Caches

- Cache latency **much less** than roundtrip between host and guest
- Filesystem caches:
  - **Data**: can be shared between host and guest (DAX)
  - **Metadata, pathname lookup**: can't be shared
- If not shared, then need to invalidate on "remote" change
  - Synchronous invalidate → strong coherency
  - Asychronous invalidate or timeout → weak coherency
- Guest cache invalidate should not block (Denial of Service)

Shared memory version table



- Multiple guests ↔ single table
- One possible implementation of **synchronous, non-blocking** invalidation
- Fast validation of cache entry
  - Compare value of two memory locations
- Not (yet) working for host filesystem changes

Current Status

VIRTIO specification - Merged in 1.2

Linux guest driver - Core merged in 5.4, DAX not yet posted

QEMU vhost-user-fs device - Merged in 4.2

QEMU virtiofsd daemon - First part posted

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

Red Hat

Cache modes

Users can choose coherency vs performance trade-off:

- Coherency may require more communication, lower performance

Available modes:

Mode	**Close-to-open consistency**	**Guest page cache**	**Metadata timeout**
none	Yes	No	Instant
auto	Yes	Yes	1 second
always	No	Yes	1 day

Security model

Guest has full control over file uid, gid, and permissions

- Access checks performed inside guest
- Guests sharing a file system must trust each other
- Design choice in current implementation, not inherent in VIRTIO spec

virtiofsd runs as root but is sandboxed:

- Mount namespace restricts access to only the shared directory
- Seccomp whitelist reduces syscall attack surface

Benchmark configuration

Host:

- Fedora 28 host with 32 GB RAM and 24 logical CPUs
- 2 sockets x 6 cores per socket x 2 threads per core
- ramfs as the storage

Guest:

- 8 GB RAM and 16 vCPUs
- 8 GB DAX Window
- 4 x 2 GB fio data files

9p (cache=none), virtio-fs (cache=none), and virtio-fs (cache=none + dax)
iodepth=16