Towards High-availability for Virtio-fs

Jiachen Zhang & Yongji Xie
ByteDance STE Team

KVM Forum 2021

Agenda

- Background
- Virtio-fs Daemon Crash Recovery
- Virtio-fs Live Upgrade & Live Migration
- Status & Future Work

Background

ByteDance字节跳动

Virtio-fs

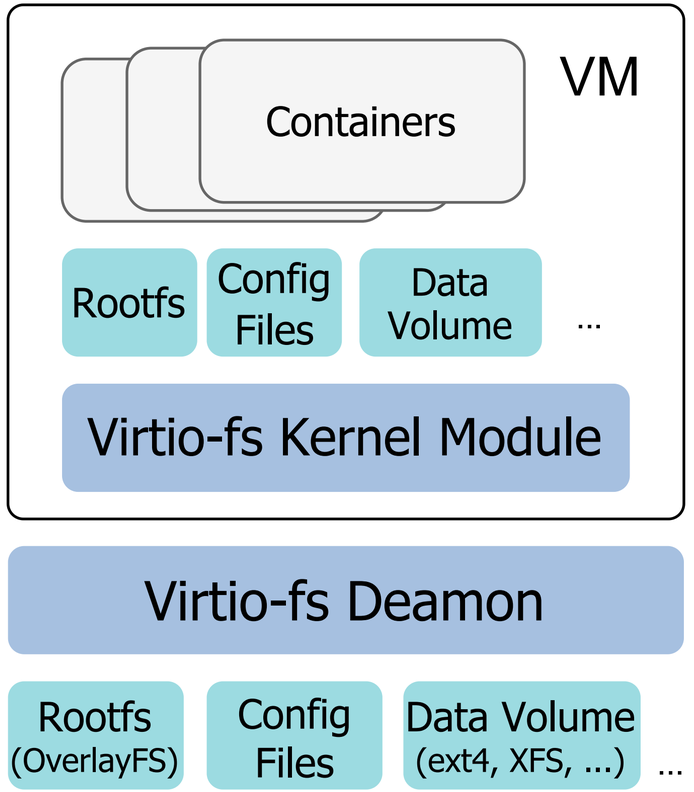
- A pass-through file system
  - Lets VMs access a shared host directory tree.
  - Designed for local FS semantics and performance.

- Being actively developed
  - Linux: fs/fuse/virtio_fs.c
  - QEMU: tools/virtiofsd/*, hw/virtio/vhost-user-fs*
  - Rust-VMM crates: cloud-hypervisor, fuse-backend-rs, ...
  - Kata Containers, libvirt, ...

- More information see https://virtio-fs.gitlab.io

Virtio-fs Use Cases

- VM host directory pass-through
  - Guest applications can access Host FS without file copying
- Kata Containers directory pass-through
  - Data volumes, container image rootfs, ...
  - Overlayfs can be prepared in host
- Implementing customized virtio-fs daemons
  - As a distributed FS client, similar to ceph-fuse
  - For direct container image access (e.g. Nydus)
- Also see the KVM Forum 2019 talk:
  https://www.youtube.com/watch?v=969sXbNX01U

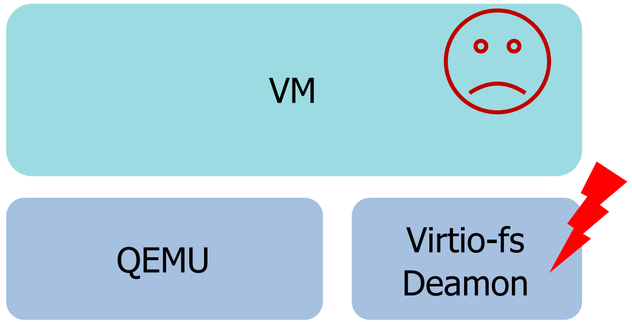


Kata Containers Use Case

Why High-availability for Virtio-fs

- High-availability features
  - Crash recovery
  - Live upgrade
  - Live migration

- Virtio-fs Daemon (virtiofsd) does not support any high-availability features



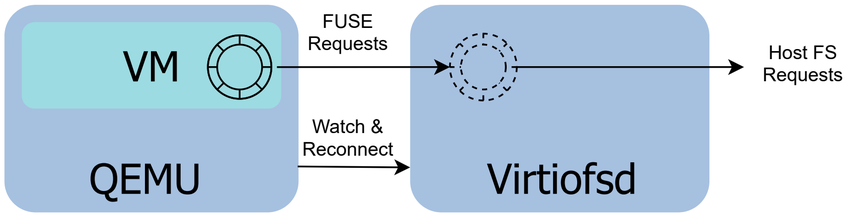
- Mean time to failure (MTTF) without HA features
  - MTTF(VM w/o virtiofs) <= MTTF(QEMU)
  - MTTF(VM with virtiofs) <= Min(MTTF(QEMU), MTTF(virtiofsd))

Virtio-fs Daemon Crash Recovery

ByteDance字节跳动

Restarting on Crash

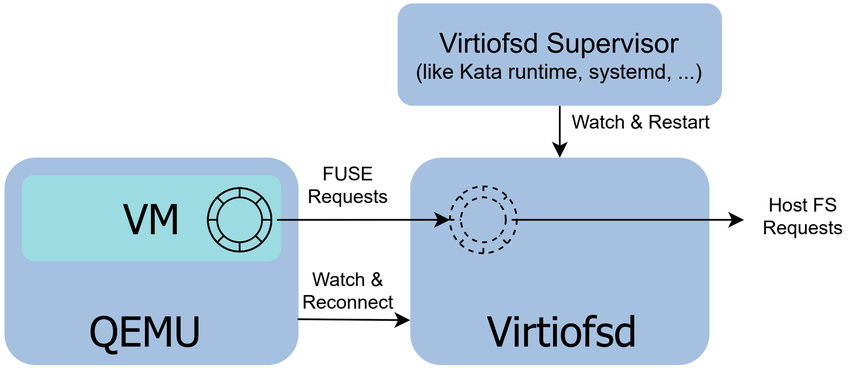
(common issue for vhost-user services)



- “Fail-stop” failure model
  - Mistakenly killed by other processes
  - Killed by kernel OOM killer
  - Crashes for other reasons like SIGSEGV

Restarting on Crash

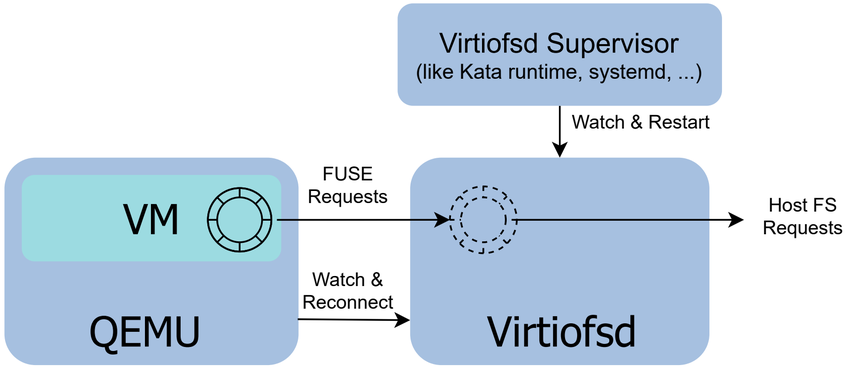
(common issue for vhost-user services)



- "Fail-stop" failure model
  - Mistakenly killed by other processes
  - Killed by kernel OOM killer
  - Crashes for other reasons like SIGSEGV
- A supervisor process to restart virtiofsd on crash
  - Such as Kata Containers runtime or systemd
  - Virtiofsd crash detection
  - Restart immediately when a crash is detected
  - QEMU keeps trying to reconnect the new vhost-user socket

Resubmitting Inflight Requests

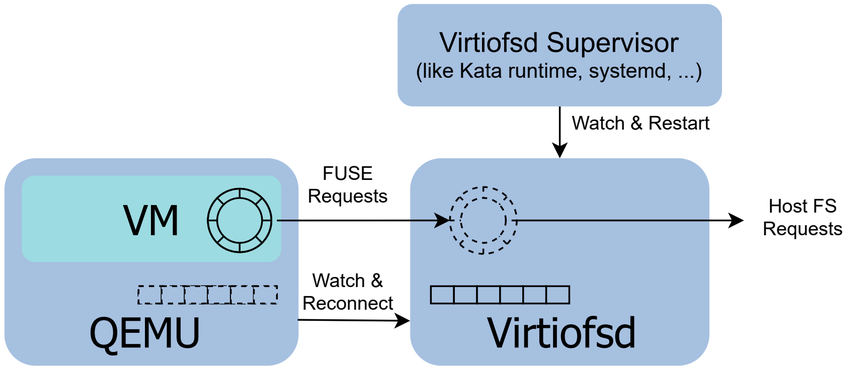
(common issue for vhost-user services)



- Unfinished FUSE requests need to be handled properly after restarting

Resubmitting Inflight Requests

(common issue for vhost-user services)



- Unfinished FUSE requests need to be handled properly after crash restart

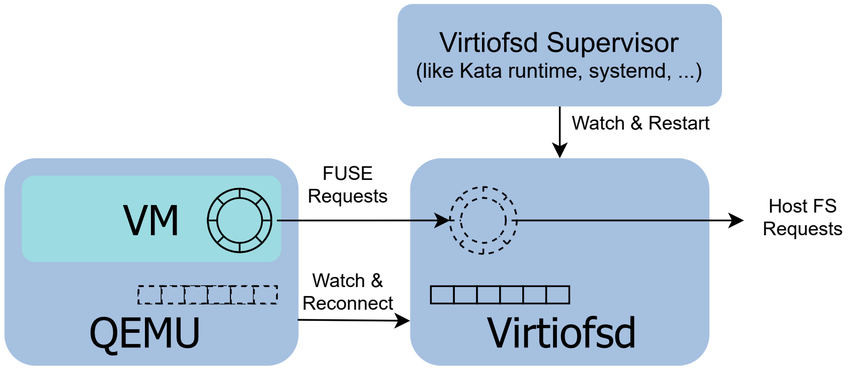
- Vhost-user inflight I/O tracking
  - A common crash recovery vhost-user feature
  - Protocol feature:
    `VHOST_USER_PROTOCOL_F_INFLIGHT_SHMFD`
  - Message types:
    `VHOST_USER_GET_INFLIGHT_FD`
    `VHOST_USER_SET_INFLIGHT_FD`

- Idempotent issues

Crash

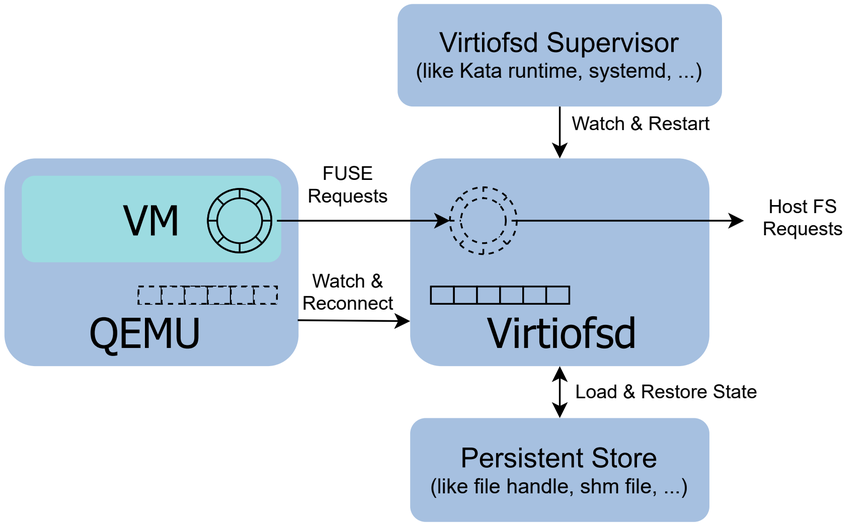
```
void a_fuse_request_handler()
{
    operation_1();
    operation_2();
    // ...
    operation_N();

    reply_complete();
}
```

Saving & Restoring Internal States



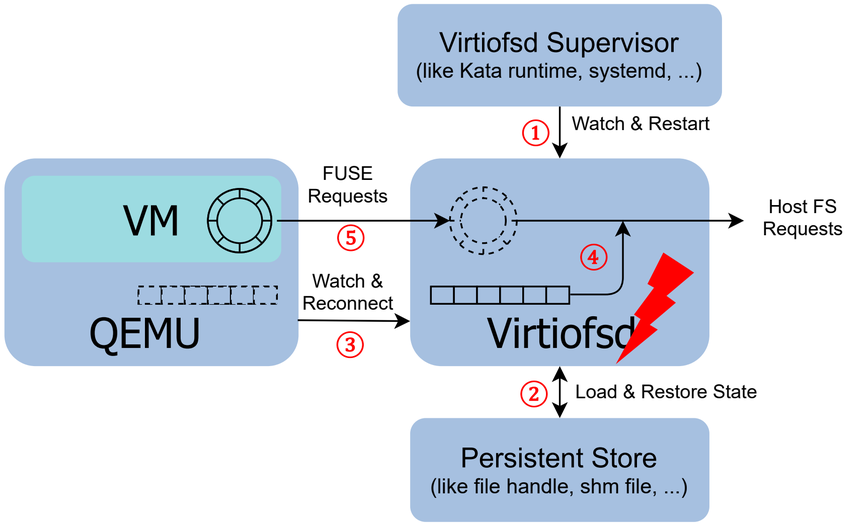
- Virtiofsd internal states
  - Some in-memory states (struct lo_map in QEMU virtiofsd)
  - Open file descriptors

Saving & Restoring Internal States



- Virtiofsd internal states
  - Some in-memory states (struct lo_map in QEMU virtiofsd)
  - Open file descriptors
- Where can we save the states?
  - In-memory states:
    - Shm files (flat-map) or another process
  - Opened file descriptors:
    - File handles
      (open_by_handle_at(2) and name_to_handle_at(2))
    - Or another process
      (Unix domain socket SCM_RIGHTS)
- The general idea
  - When an internal state updates, save them
  - When virtiofsd restarts, restore the states

Crash Recovery Procedures



① The supervisor process detects virtiofsd failure and restarts a new virtiofsd.

② The new virtiofsd restores its internal states from the persistent state store, then listens on the vhost-user socket.

③ QEMU reconnects to the new virtiofsd, re-setups the vhost-user state like VM memory layouts, virtqueue addrs, and sends the inflight I/O tracking log.

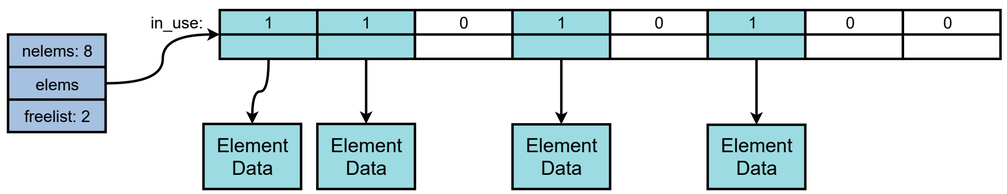
④ The new virtiofsd re-handles the inflight FUSE requests from the inflight I/O tracking log.

⑤ The new virtiofsd starts to handle normal FUSE requests from the VM guest.

Saving In-memory States to Flat-map

- Flat-map: Based on QEMU virtiofsd struct lo_map

```
struct lo_map_elem {
    union {
        struct lo_inode *inode;
        struct lo_dirp *dirp;
        int fd;
        ssize_t freelist;
    };
    bool in_use;
};

struct lo_map {
    struct lo_map_elem *elems;
    size_t nelems;
    ssize_t freelist;
};
```



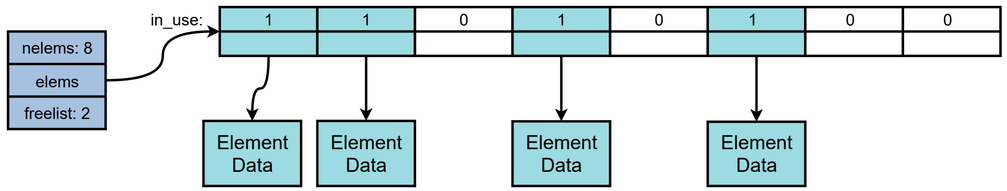
Saving In-memory States to Flat-map

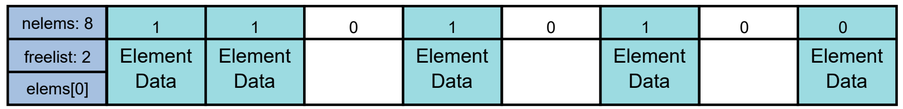
- Flat-map: Based on QEMU virtiofsd struct lo_map, more mmap-friendly
  - Embed element data into slots instead of dynamically allocating and attaching them to pointers
  - Attach the elements to the end of the map meta fields.
  - Crash consistency on flat-map updating is properly handled (For details see the RFC patchset)

```
struct lo_map_elem {
    union {
        struct lo_inode *inode;
        struct lo_dirp *dirp;
        int fd;
        ssize_t freelist;
    };
    bool in_use;
};

struct lo_map {
    struct lo_map_elem *elems;
    size_t nelems;
    ssize_t freelist;
};
```

Make its memory layout more flatten

```
struct flat_map_elem {
    union {
        struct lo_inode inode;
        struct lo_dirp dirp;
        struct lo_fd fd;
    };
    ssize_t freelist;
    gint refcount;
};

struct flat_map {
    size_t nelems;
    ssize_t freelist;
    map_t map_type;
    int map_fd;
    struct lo_map_elem elems[];
};
```





Saving File Descriptors as File Handles

- File handles: man page of name_to_handle_at & open_by_handle_at (2)

```
int name_to_handle_at(int dirfd, const char *pathname,
                      struct file_handle *handle, int *mount_id,
                      int flags);
int open_by_handle_at(int mount_fd, struct file_handle *handle,
                      int flags);

name_to_handle_at() returns an opaque handle that corresponds to a specified file;
open_by_handle_at() opens the file corresponding to a handle returned by a previous
call to name_to_handle_at() and returns an open file descriptor.
```

- Save opened fds of virtiofsd to the host kernel by name_to_handle_at (2) and restore them by open_by_handle_at (2) when performing recovery.

Idempotent of Resubmitted Requests

- Why idempotent is important ?
  - Idempotent request types always produce the same result when re-handled after recovery
  - Non-idempotent FUSE requests would leave some residual states in virtiofsd and host FS

- Case-by-case analysis:

Classification	FUSE Request Types
Data / Meta Read	lookup, getattr, readlink, opendir, readdir, readdirplus, read, statfs, getxattr, listxattr
Data / Meta Write	setattr, fsyncdir, flush, fsync, write, fallocate, setxattr, removexattr, getlk, setlk, flock,
Meta Create	create, open, opendir, mkdir, mknod, symlink, link
Meta Release	rename, unlink, rmdir, releasedir, release, forget, forget_multi

Idempotent Need relaxed error handling Need journaling

Idempotent: Case Study

Crash

Crash

```
static void lo_fallocate(fuse_req_t req, fuse_ino_t ino,
                         int mode, off_t offset,
                         off_t length,
                         struct fuse_file_info *fi)
{
    int err; // START HERE!
    (void)ino;

    err = fallocate(lo_fi_fd(req, fi), mode, offset, length);
    if (err < 0) {
        err = errno;
    }

    fuse_reply_err(req, err); // COMPLETE HERE!
}
```

Case 1. Idempotent Requests
– lo_fallocate

- fallocate(2) is idempotent
- No virtiofsd internal state is updated
- Therefore, lo_fallocate is idempotent

Idempotent: Case Study (Cont.)

Crash

Crash

```
static void lo_mkdir(fuse_req_t req, fuse_ino_t parent,
                     const char *name, mode_t mode)
{
    int res; // START HERE!

    res = mkdirat(dirfd, path, mode);
    if (res == -1) {

        fuse_reply_err(req, 0);
    }

    // ...

    fuse_reply_err(req, 0); // COMPLETE HERE!
}
```

Case 2. Need relaxed error handling – lo_mkdir

- mkdirat(2) is not idempotent, re-execute it will cause EEXIST error!

Idempotent: Case Study (Cont.)

Crash

Crash

```
static void lo_mkdir((fuse_req_t req, fuse_ino_t parent,
                      const char *name, mode_t mode)
{
    int res; // START HERE!

    res = mkdirat(dirfd, path, mode);
    if (res == -1 &&
        !(lo.reconnect && errno == EEXIST)) {
        fuse_reply_err(req, 0);
    }

    // ...

    fuse_reply_err(req, 0); // COMPLETE HERE!
}
```

Case 2. Need relaxed error handling – lo_mkdir

- mkdirat(2) is not idempotent, re-execute it will cause EEXIST error!
- Relax the error handling in virtiofsd. It's safe, because guest kernel should have handled EEXIST error properly.
- lo_mkdir is idempotent now with relaxed error handling.

Idempotent: Case Study (Cont.)

```
static void lo_forget(fuse_req_t req, fuse_ino_t ino,
                      uint64_t nlookup)
{
    struct lo_inode *inode; // START HERE!

    inode = lo_inode(req, ino);

    // ...

    inode->nlookup -= 1;

    // ...

    fuse_reply_none(req); // COMPLETE HERE!
}
```

Crash

Crash

Case 3. Need journaling – lo_forget

- A virtiofsd internal state (nlookup counter) is updated, re-execute it may cause inconsistency!

Idempotent: Case Study (Cont.)

```
static void lo_forget(fuse_req_t req, fuse_ino_t ino,
                      uint64_t nlookup)
{
    struct lo_inode *inode; // START HERE!

    inode = lo_inode(req, ino);

    // ...

    record_to_journal(req, inode, inode->nlookup);
    inode->nlookup -= 1;

    // ...

    fuse_reply_none(req); // COMPLETE HERE!
}
```

Crash

Crash

Case 3. Need journaling – lo_forget

- A virtiofsd internal state (nlookup counter) is updated, re-execute it may cause inconsistency!
- Introduce journaling, rollback the nlookup counter first when performing recovery.
- lo_forget is idempotent with journaling.

Downtime Optimizations

- Optimization 1. ms-level reconnetion delay
  - Modify QEMU to support milli-second level retrying delay of vhost-user socket reconnection.
- Optimization 2. Lazy file destriptor restore
  - Restarted virtiofsd only restores an opened fd from its file handle when it is accessed.
- Performance Evaluation - downtime < 1s

No. of Opened FDs (NUM_FILES)	Virtiofsd Downtime (ms)		
	Without Opti	Opti.1	Opti.1 + Opti.2
1	1004	11	10
100	1004	69	12
1000	1004	520	85

```
fio --name=virtiofs_dir_test –directory=fio-testfilesfile --nrfiles=$NUM_FILES
--size=10G --runtime=10 --direct=1 --ioengine=libaio --readwrite=randread
--blocksize=4k
```

Virtio-fs Live Upgrade & Live Migration

ByteDance字节跳动

Virtio-fs Live Upgrade

Why?

- Vhostuser renegotiation increase downtime
- Get rid of inflight I/O replay

How?

- Need a communication channel between virtiofsd and the supervisor process to launch a live upgrade
- Virtiofsd internal state inheritance
  - In-memory states: mmapped from flat-map files
  - Opened FDs: can be inherited from old virtiofsd instead of restoring from file handles

Virtio-fs Live Migration

Why?

- Enable the live migration of VMs based on QEMU with virtiofsd

How?

- QEMU needs to support save & load of vhost-user-fs states. (Relatively easy)

- Virtiofsd internal state migration
  - Flat-maps should be transferred to the target node.
  - Opened fds: We need to save more internal states for path reconstruction.

- Inflight I/O Handling
  - drain inflight I/O in source
  - resubmit inflight I/O in dest

Status & Future Work

ByteDance 字节跳动

Status & Future Work

Status

- An RFC patchset of virtiofsd crash recovery feature is already posted to QEMU-devel and Virtio-fs mailing lists:
  - https://patchwork.kernel.org/project/qemu-devel/cover/20201215162119.27360-1-zhangjiachen.jaycee@bytedance.com/

Future Work

- Post virtiofsd crash recovery RFCv2 patchset
- Develop high-availability features for the Rust version virtiofsd-rs
- Enable the high-availability features with Rust-VMM hypervisors

THANKS.