redhat.

Applying Polling Techniques to QEMU

Reducing virtio-blk I/O Latency

Stefan Hajnoczi <stefanha@redhat.com>
KVM Forum 2017

Agenda

Problem: Virtualization overhead is significant for high IOPS devices

QEMU's event-driven architecture

Polling techniques and adaptive polling

Event sources that can be polled from userspace

Performance results for virtio-blk

About me

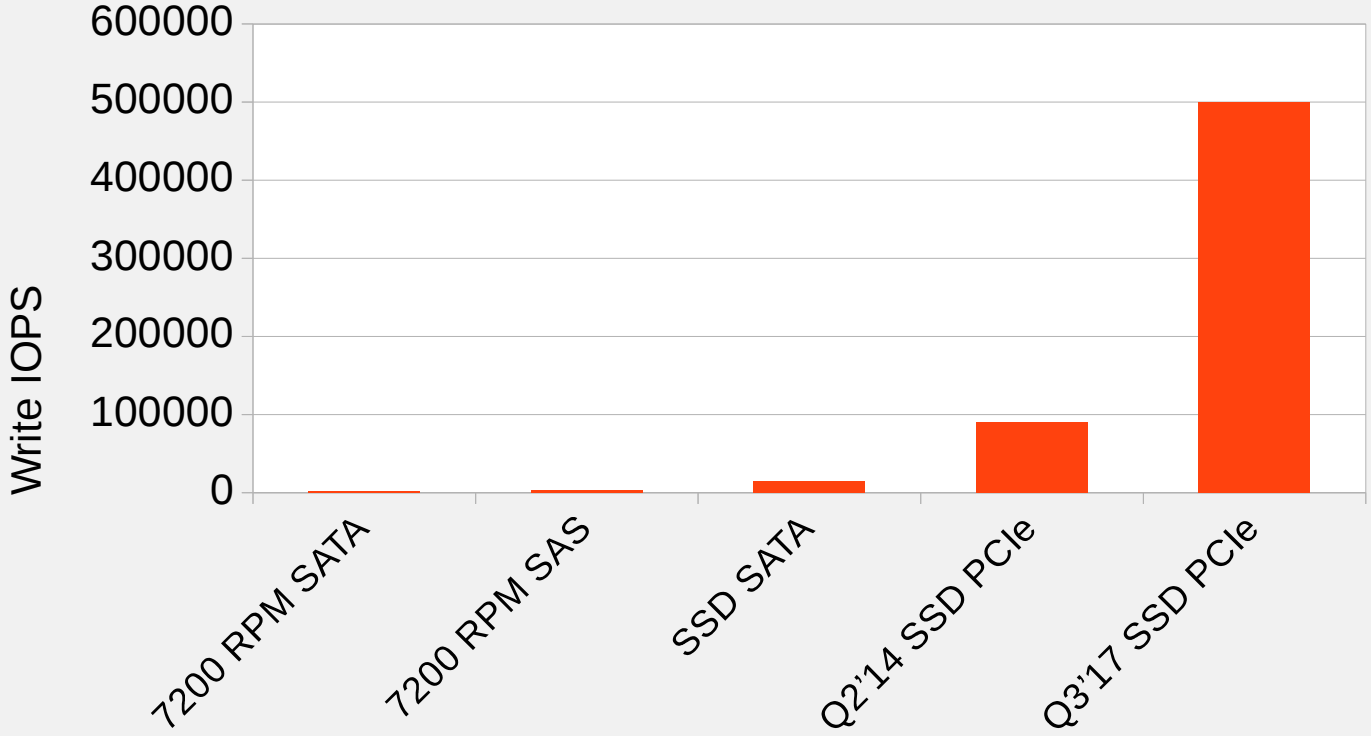
QEMU contributor since 2010

Focus on storage, tracing, performance

QEMU Google Summer of Code and Outreachy organizer

Work in Red Hat's virtualization team

Block I/O Performance Trends



* This comparison is approximate because disk vendor specs vary (e.g. sequential vs random I/O). Estimation was required to graph this data on a single graph.

WD1003FBYZ 7200 RPM SATA 6 Gb/s – sustained host-drive throughput 128 MB/s

Samsung PM863a SSD SATA 6 Gb/s – 520 MB/s read, 480 MB/s write, 97k iops 4 KB random read, 15k iops 4 KB random write

WD4001FYYG 7200 RPM SAS 6 Gb/s – sustained host-drive throughput 182 MB/s

Intel SSD DC P3700 PCIe (Q2'14) – 2800 MB/s read, 1900 MB/s write, 460k iops read, 90k iops write, 20 us read/write latency

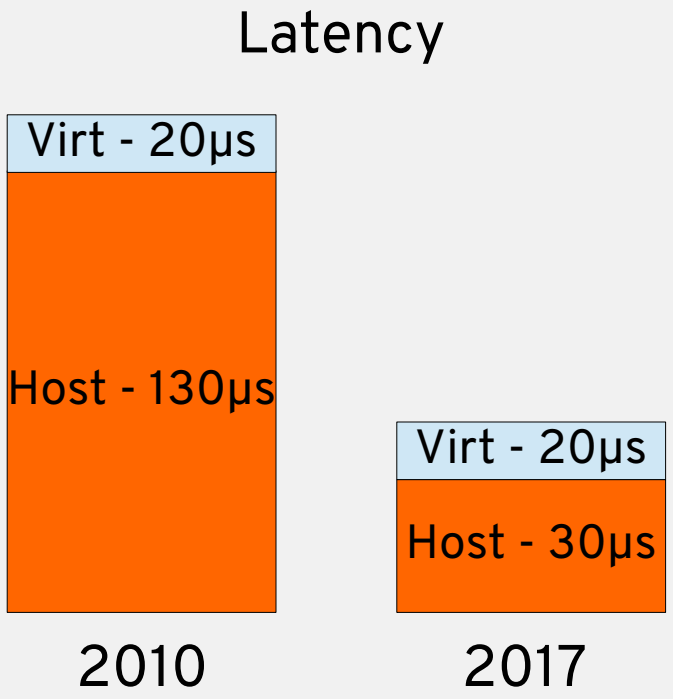
Intel Optane SSD DC P4800X PCIe (Q3'17) – 2400 MB/s read, 2000 MB/s, 550k iops read, 500k iops write, 10 us read/write latency

10 us latency!

Virt Overhead for Block I/O

Previously a small fraction of request latency, but now **significant due to faster storage devices.**

Latency profile from 2010:
https://www.linux-kvm.org/page/Virtio/Block/Latency



Software Features (Why Not PCI Passthrough?)

Despite overhead, we still need a software layer for:

- Multiple guests share one storage device
- Snapshots
- Encryption
- Storage migration
- Backup
- I/O throttling
- And more...

Quantifying Virtualization Overhead

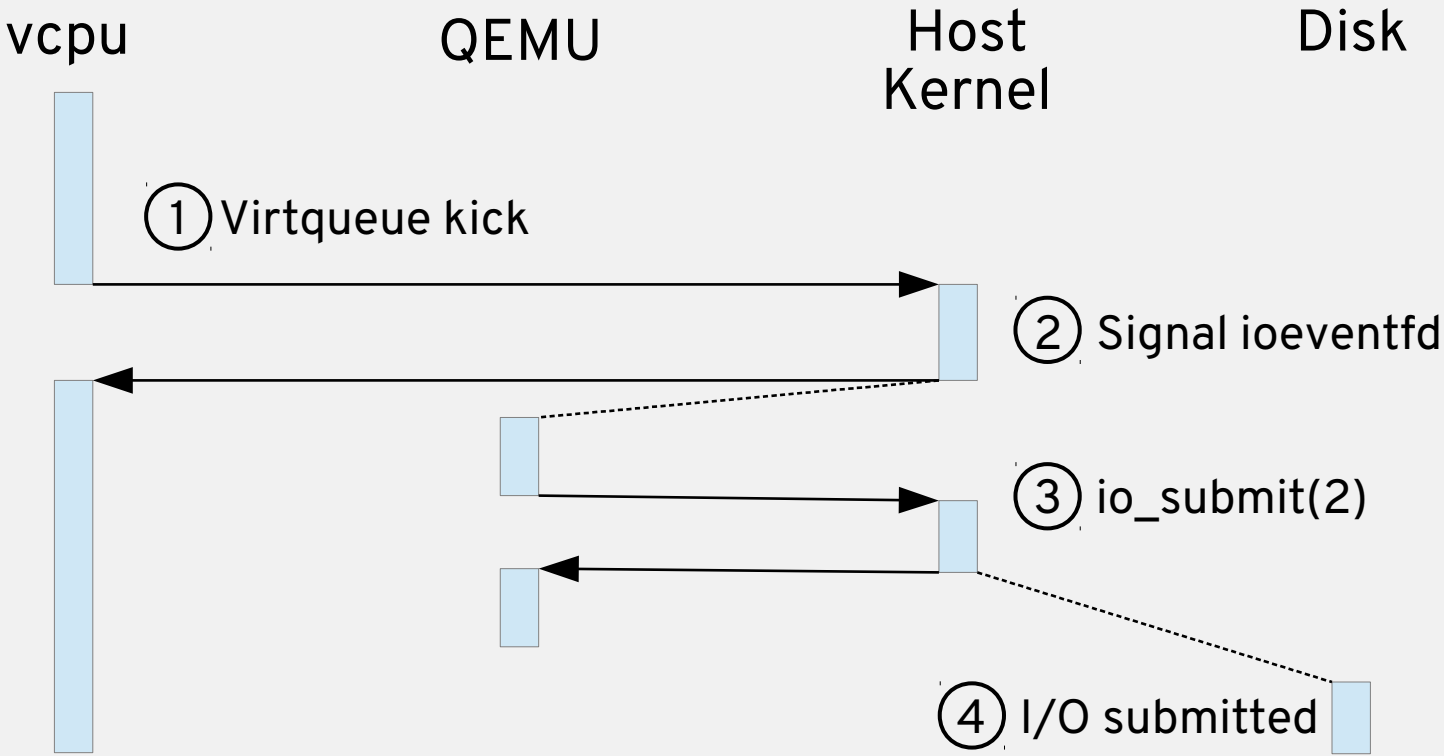
Karl Rister collected traces showing 20 µs overhead and poor performance on NVMe drives.

What is going on?

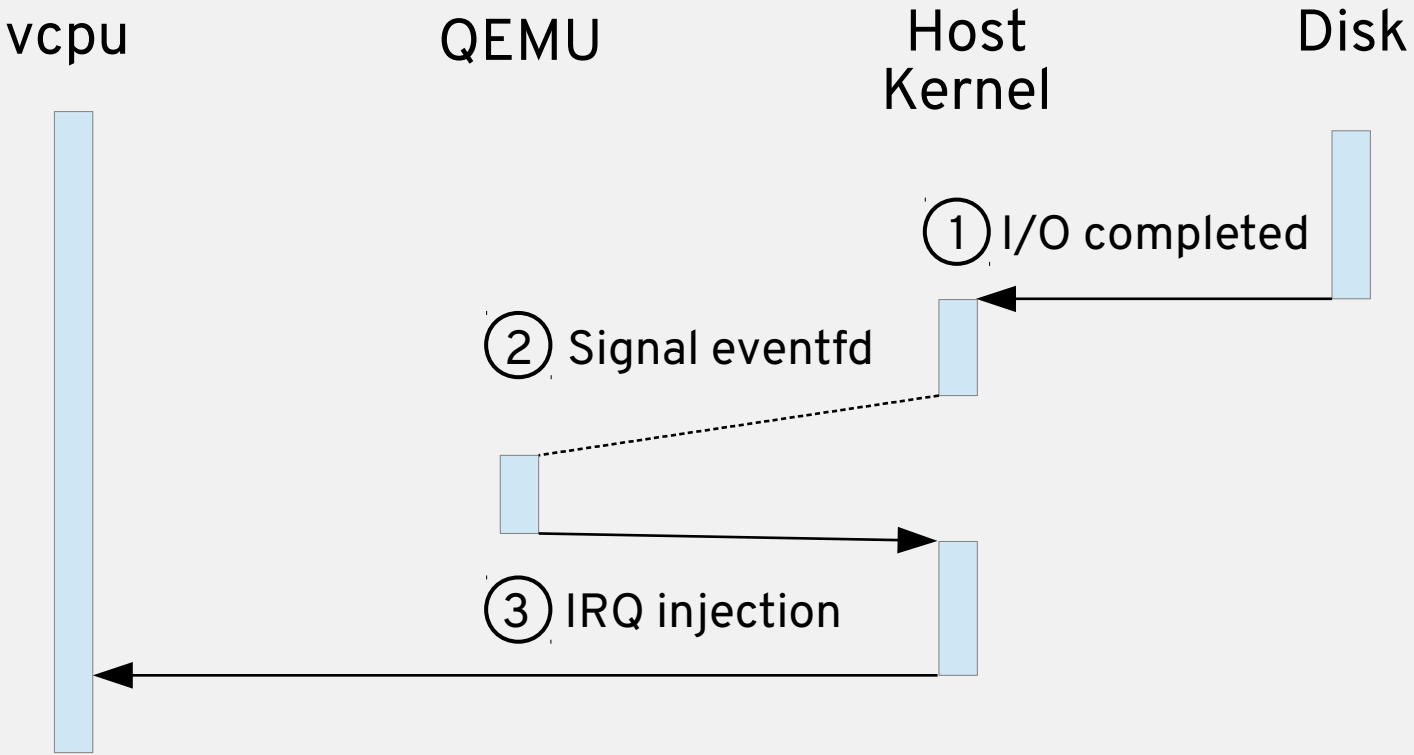
Using perf(1) to show virtqueue kick to I/O submission latency:

```
# perf record -a -e kvm:kvm_fast_mmio -e syscalls:sys_exit_read
CPU 0/KVM 11034 [005] 33228.767076:    kvm:kvm_fast_mmio: 0xfebff000
IO iothread1 11027 [001] 33228.767079: syscalls:sys_exit_read: 0x8
IO iothread1 11027 [001] 33228.767100: syscalls:sys_exit_read: 0x8
```

Virtio-blk I/O Submission Path



Virtio-blk I/O Completion Path



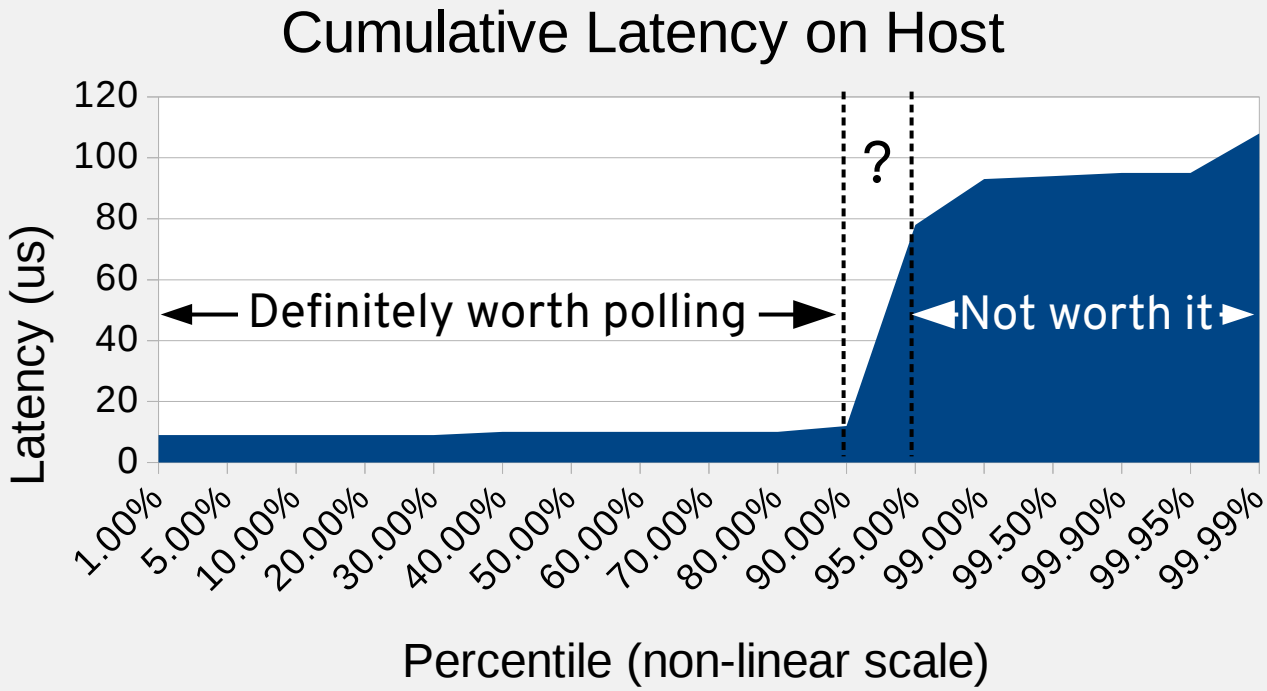
Polling

- Continuously observe item of interest for events
- Reduces latency but prevents other threads from running
- Wastes power if events are rare

```
/* Blocking wait */
while (true) {
    wait(notifier);
    process(item);
}
```

```
/* Polling */
while (true) {
    while (item == last);
    process(item);
    last = item;
}
```

How Long to Poll?



See last slide for benchmark configuration

Adaptive Polling

Avoids burning too much CPU

1) Only poll for a finite amount of time
2) Fall back to notifications if time limit exceeded
3) Adapt polling time limit to fit workload

Solves pathological cases where CPU is wasted

AioContext Polling Algorithm

Based on kvm.ko halt_poll_ns adaptive polling algorithm

```
if (waited <= poll_ns) { /* Do nothing */ }
else if (waited > poll_max_ns) shrink();
else if (poll_ns < poll_max_ns &&
         waited < poll_max_ns)) grow();
```

Adjusts poll time to sweet spot or reduces it when polling is ineffective.

QEMU Event Loop

QEMU uses ppoll(2) to wait on file descriptors

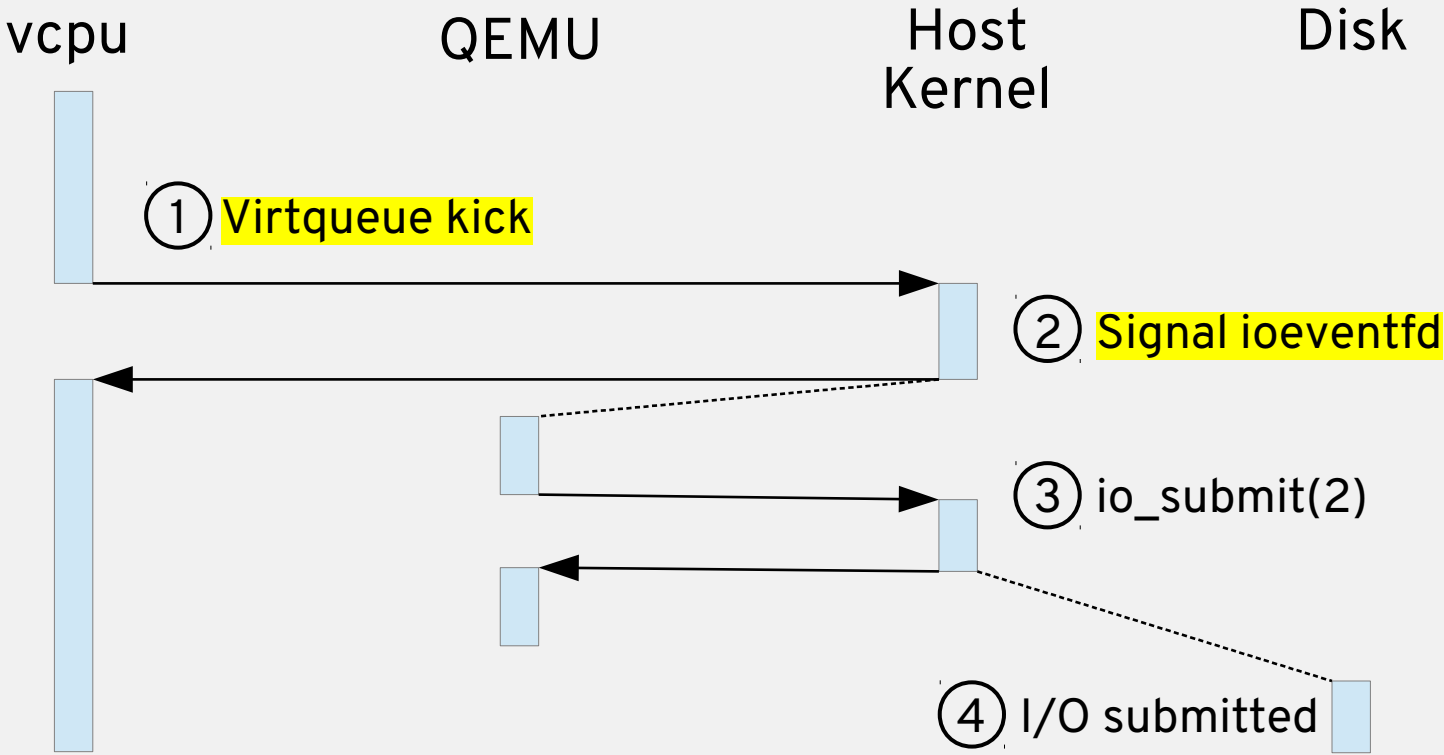
- QEMU actually sleeps until woken up, ignore “poll” in name

Types of file descriptor:

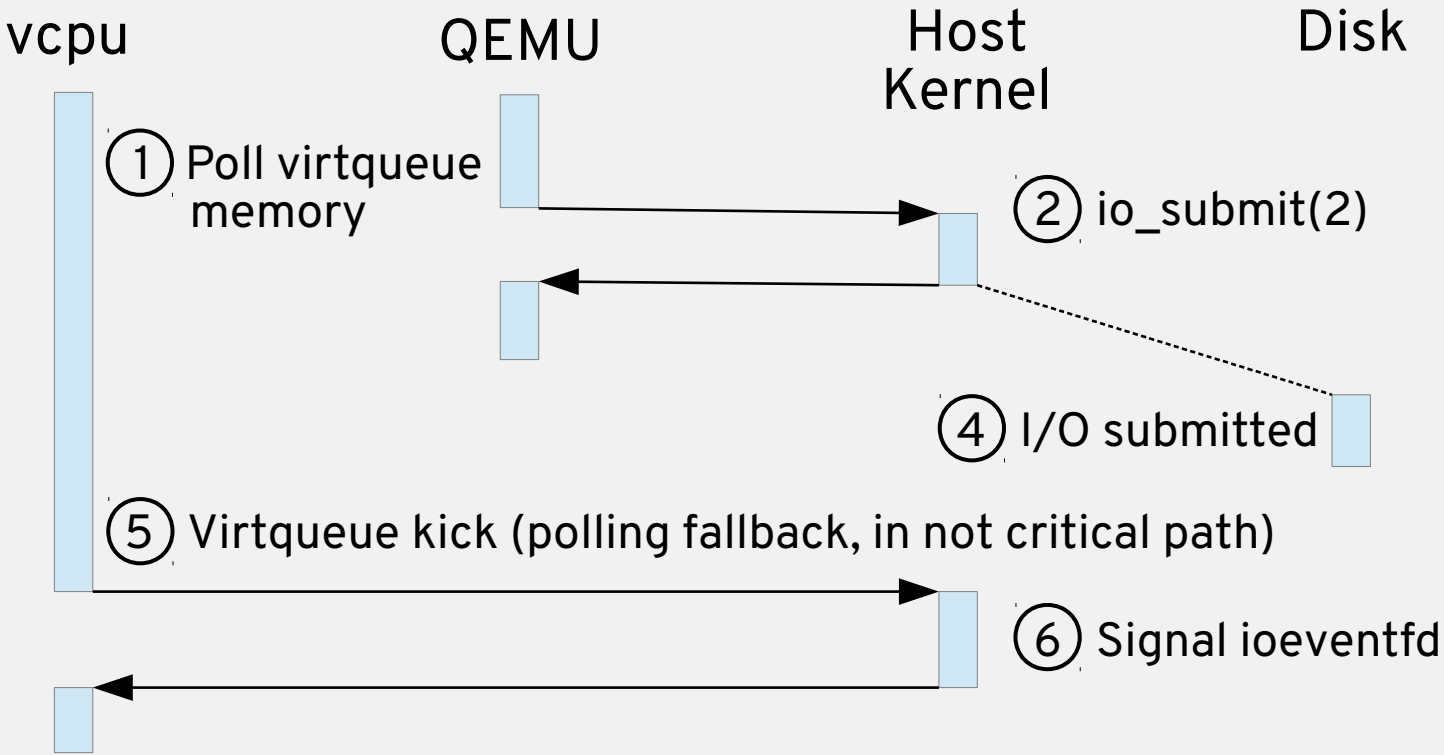
- eventfd(2) – virtqueue kick, Linux AIO completion, …
- sockets – chardev, slirp
- And more…

Need a memory location to peek at for efficient polling

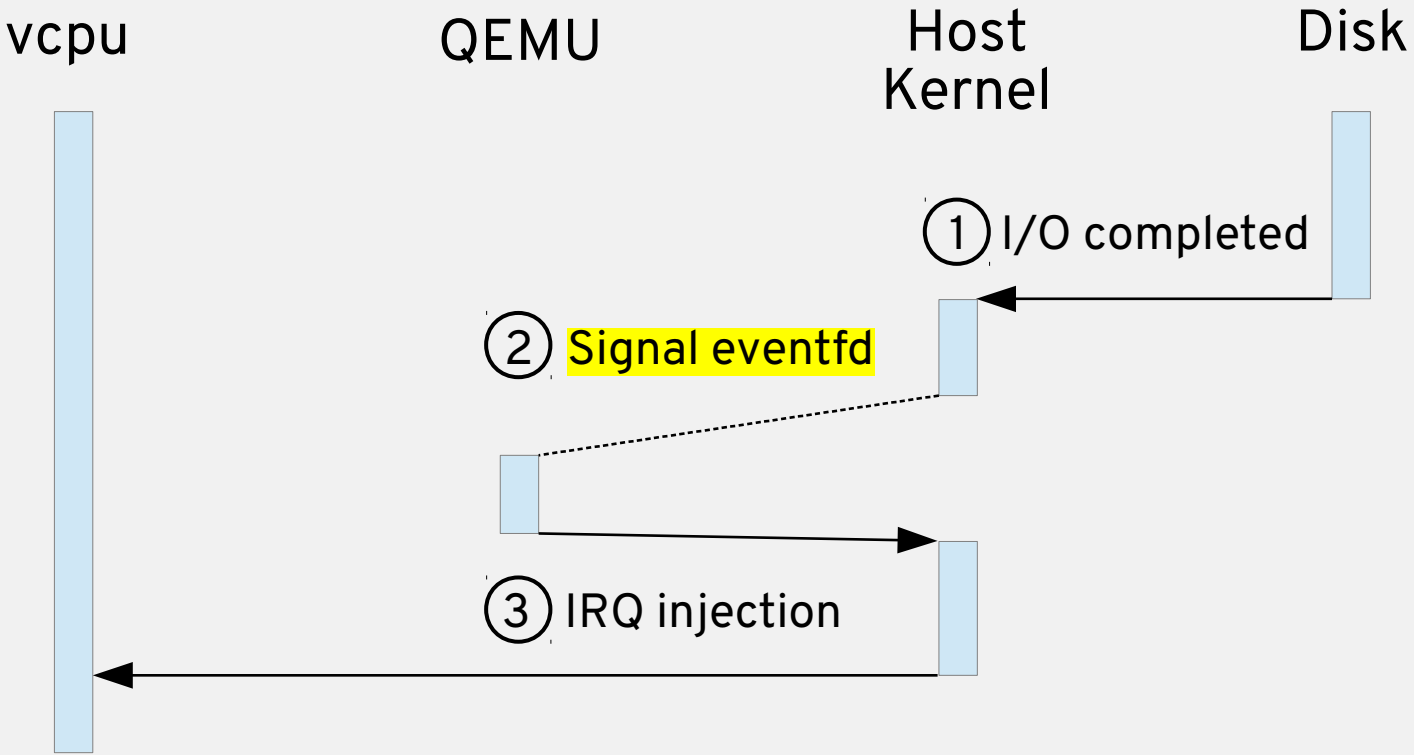
How to Poll Virtqueue Kick?



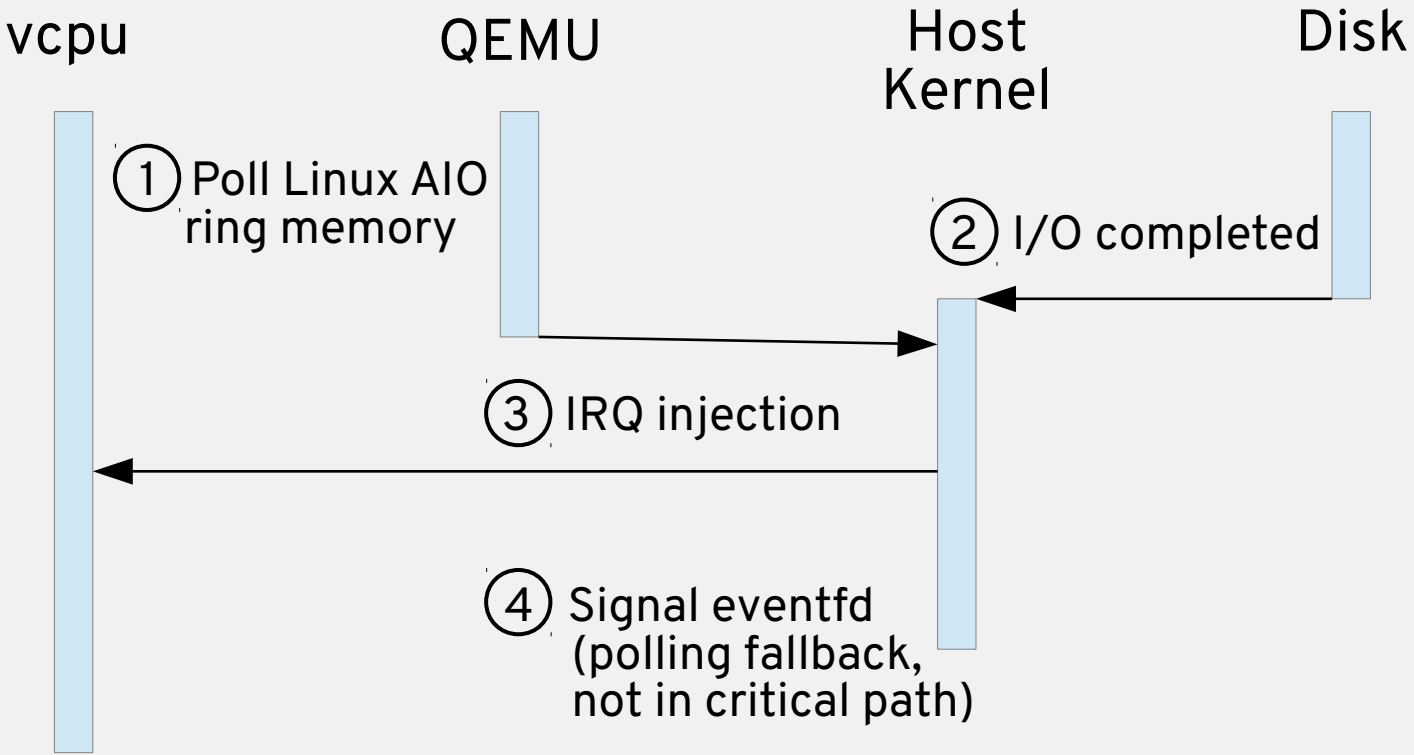
Polling Virtqueue Kick



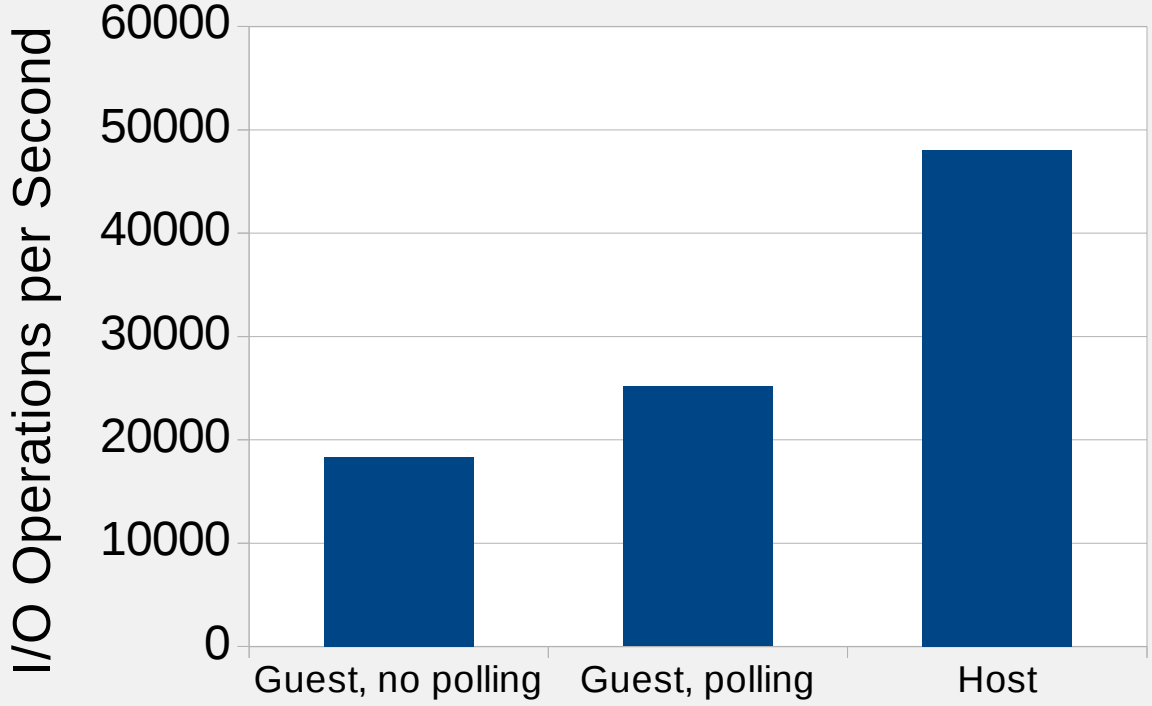
How to Poll Linux AIO Completion?



Polling Linux AIO Completion



4 KB Random Reads on NVMe



Polling IOPS +37%

poll_max_ns=32µs
Queue depth 1

Host performance inconsistent (max 48k IOPS, often 30k)
→ init drive with TRIM + 2x write in future benchmarks

See last slide for benchmark configuration

Kernel Block I/O Polling

How does userspace polling interact with kernel polling?

1) *irq_poll* interrupt mitigation
   - Starts on first interrupt after period of inactivity
   - Finishes when all events have been processed
2) *blk_mq_poll()* polls for a single request completion
   - Not used for asynchronous I/O

Bottom line: AIO programs need their own polling

Kernel vs Userspace Polling Trade-off

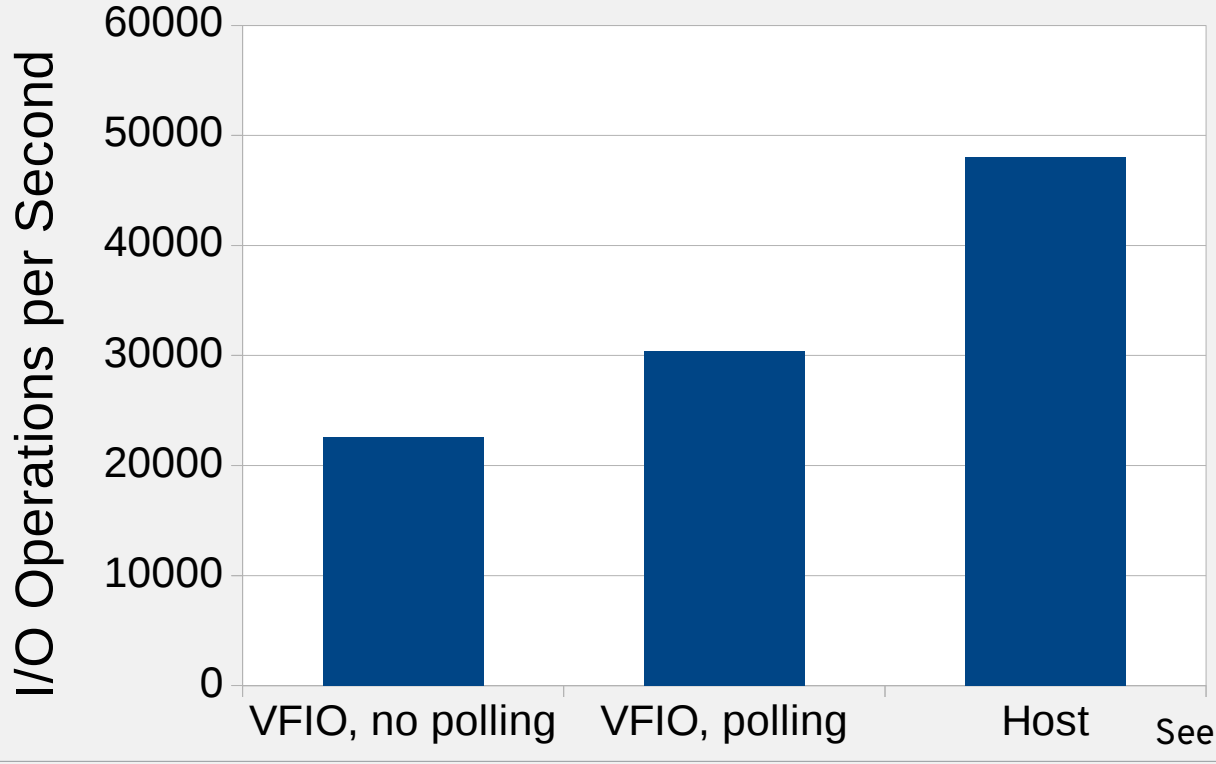
QEMU relies on kernel network and storage drivers

Devices in use by kernel drivers cannot be polled from userspace

Therefore QEMU only polls *kernel completions*, not *device completions*

What if QEMU had userspace device drivers…?

4 KB Random Reads with QEMU VFIO NVMe



Polling IOPS +35%

Fam Zheng's QEMU NVMe VFIO driver (see Fam's KVM Forum presentation)

Bypasses host kernel, retains QEMU block layer features

See last slide for benchmark configuration

Thank You

AioContext polling was released in QEMU 2.9.0.
Automatically enabled for virtio-blk dataplane.

Performance analysis with **Karl Rister** and **Andrew Theurer**
NVMe VFIO driver by **Fam Zheng**
AioContext polling perf testing by **Christian Borntträger**

My nick is ‘stefanha’ on #qemu irc.oftc.net

More on QEMU: http://blog.vmsplice.net/

AioContext Polling Status in QEMU 2.9 & 2.10

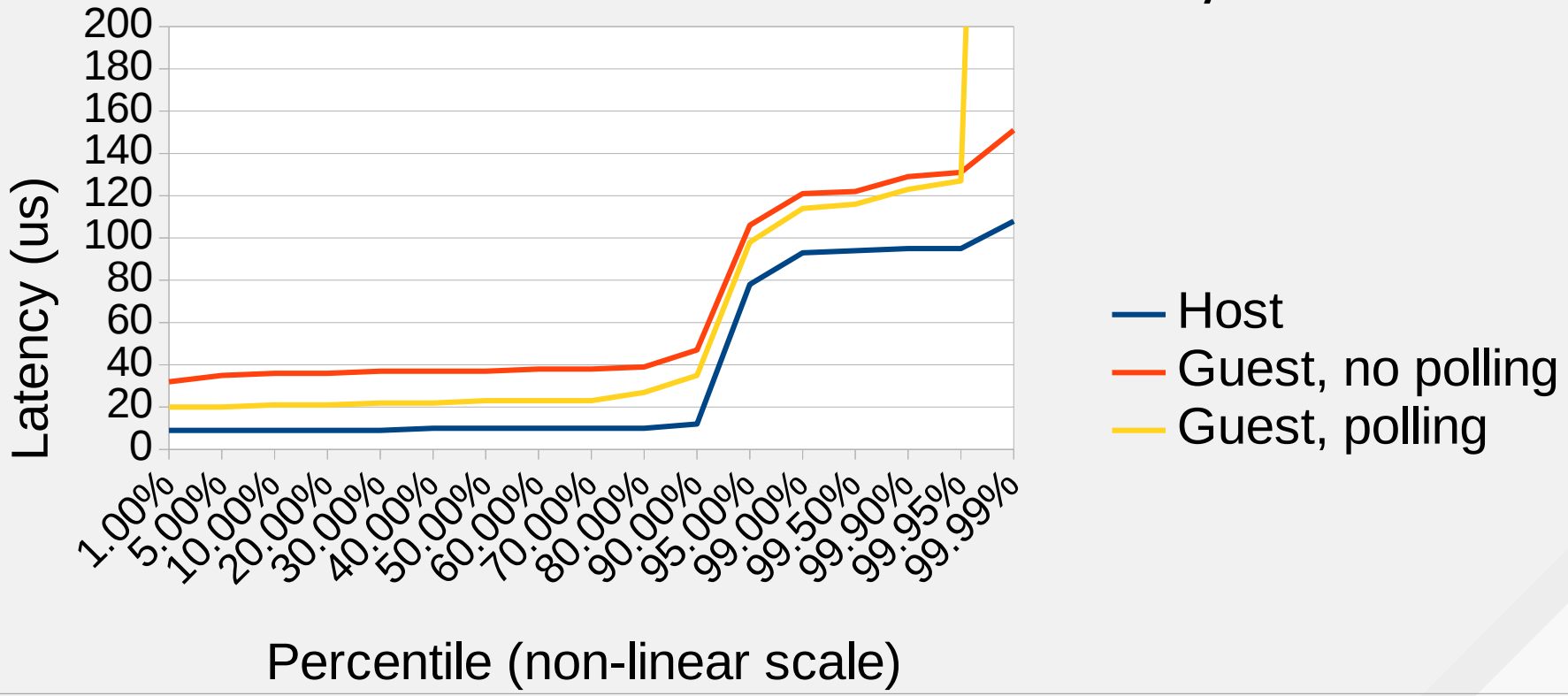
Only affects -object iothread, not the main loop

- You must use -device virtio-blk-pci,iothread=<iothread>

Disables itself if any file descriptor doesn't support polling

Defaults: poll-max-ns=32µs, poll-grow=2, poll-shrink=0

4 KB Random Read Cumulative Latency



Benchmark Configuration

Intel Xeon E5-2620 v2 @ 2.10GHz (Q3'13, 6 cores, 2 hyperthreads/core)

64 GB RAM

1 vCPU guest /w 4 GB RAM (no thread pinning, saw no improvement)

Intel DC P3700 SSD (NVMe 400 GB)

QEMU: qemu.git/master (2.10.1-ish) /w Fam Zheng's NVMe VFIO patches

Host & guest OS: Fedora kernel 4.11.8-300.fc26.x86_64

Host & guest tuned profile: latency-performance (CPUfreq governor: performance, I/O scheduler: deadline)

Virtio-blk /w iothread,aio=native,cache=none,format=raw

fio 4k randread /w direct=1,jobs=1,ioengine=libaio