Optimizing for NVMe™ Drives

The 10 Microsecond Challenge

Stefan Hajnoczi
stefanha@redhat.com

Red Hat

What are NVM Express™ drives?

- Standard PCIe interface for Solid State Disks (SSDs)
- Hardware available from multiple vendors
- Standard Linux driver
- Specification available at https://nvmexpress.org/

I/O Latency

I/O Latency is the time to perform a request. Drive spec sheets report “QD1” benchmarks, which means 1 request in flight at a time. Latency varies widely between drives, some are 10-20x slower!

10µs

Read & write

Intel® Optane™ SSD DC P4800X

Enterprise SSD

17µs

Write

Samsung 970 EVO Plus NVMe M.2 SSD

Consumer SSD

Sources:
https://www.intel.com/content/www/us/en/products/memory-storage/solid-state-drives/data-center-ssds/optane-dc-ssd-series/optane-dc-p4800x-series/p4800x-750gb-aic.html
https://www.samsung.com/us/computing/memory-storage/solid-state-drives/ssd-970-evo-plus-nvme-m-2-500gb-mz-v7s500b-am/

“Latency Numbers Every Programmer Should Know”

Based on a slide by Jeff Dean

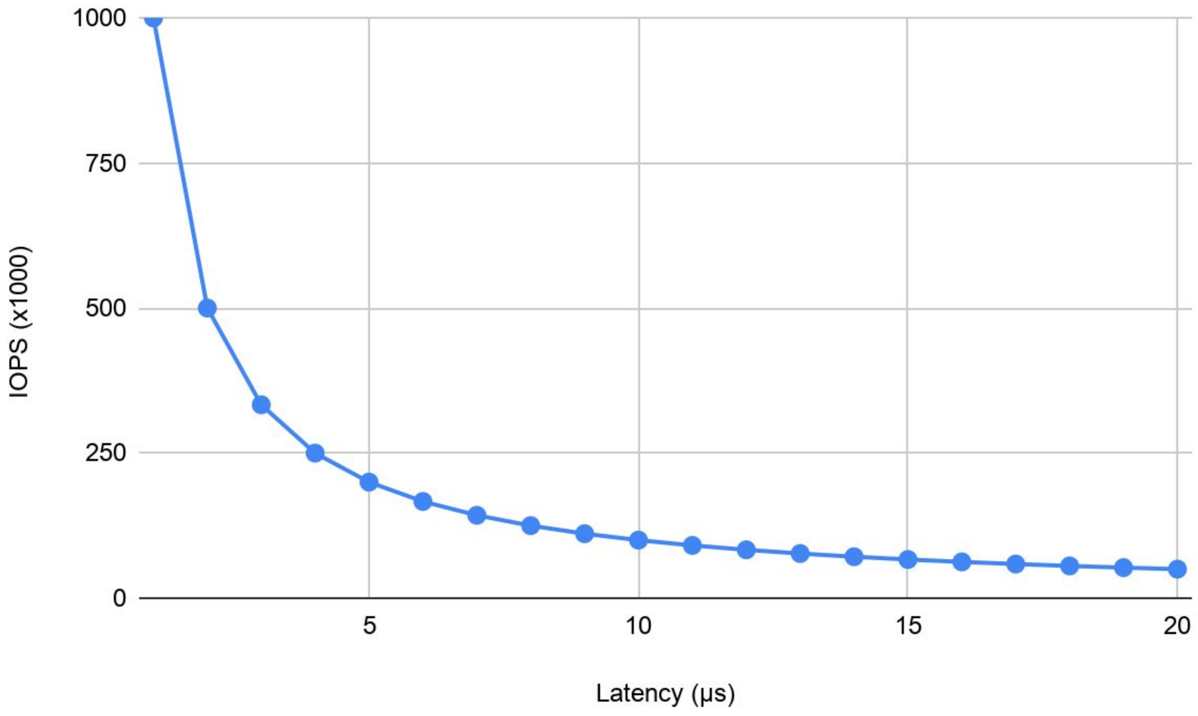
These are not exact values but they are good for comparison.

	Latency	Unit
L2 cache reference	4	ns
Main memory reference	100	ns
Compress 1KB with Zippy	2	µs
SSD Random Read	10	µs
Spinning disk seek	2	ms
Packet roundtrip CA⇆NL	150	ms

Sources:
2012 data - https://gist.github.com/sergekukharev/ccdd49d23a5078f108175dc71ad3c06c
2020 data - https://colin-scott.github.io/personal_website/research/interactive_latency.html

IOPS vs Latency is a reciprocal

When latency is small, IOPS can be misleading



- I/O Operations Per Second (IOPS) at QD1 is Runtime / Latency
- IOPS improves much less when latency is reduced 20→18µs than 4→2µs
- "IOPS increased by 10k" isn't enough information to know how much latency was reduced
- NVMe drives can be 10µs or less, prefer latency to IOPS when comparing performance

Red Hat

Overhead vs Latency

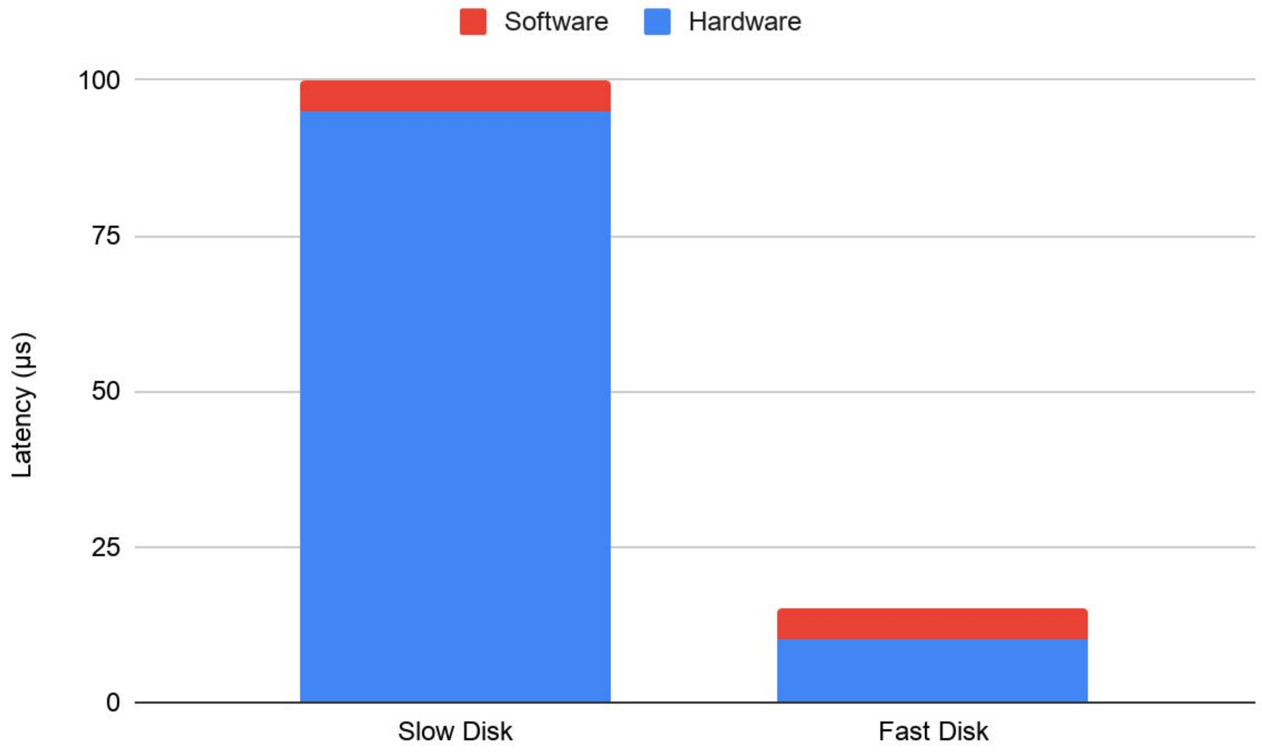
When hardware latency decreases, software overhead grows

Same software, faster hardware

5% → 33%

Software overhead

Software improvements required to preserve low overhead



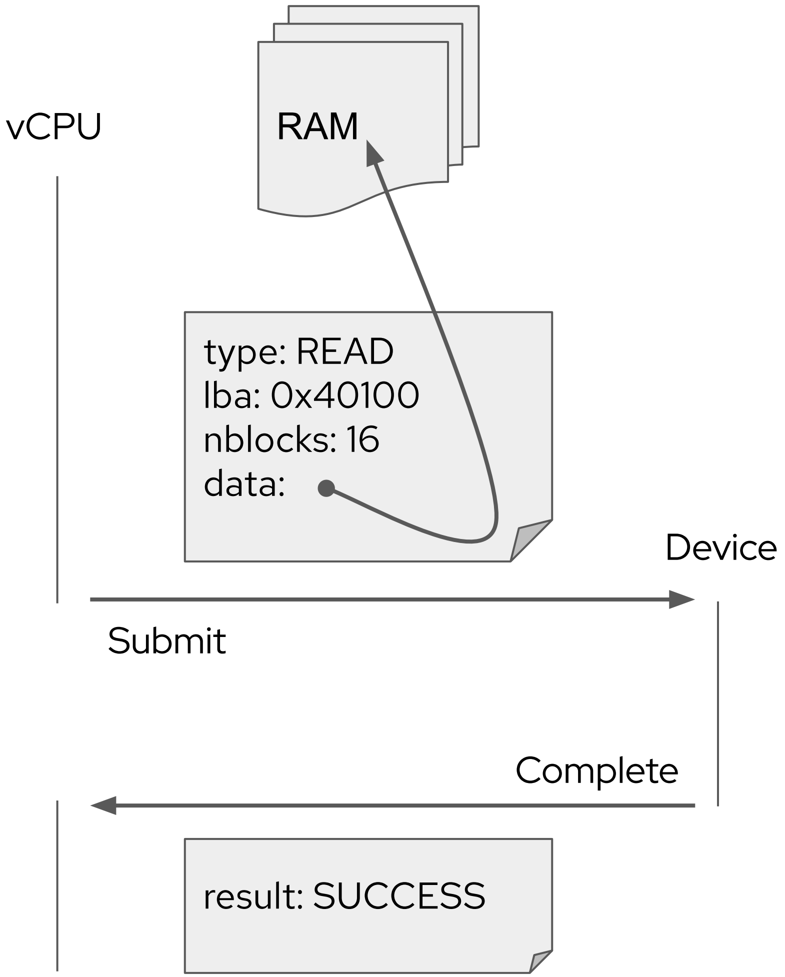
What does this mean?

- Re-examine guest and host software stack
- Rethink architecture because hardware is so much faster
- Micro-optimizations that had little effect are interesting now

The 10 Microsecond Challenge

I/O Request Lifecycle

Simplified Model



Two messages:

- Submit (vCPU→Device)
  Tell device to perform I/O request
- Complete (Device→vCPU)
  Tell vCPU that I/O request has finished

Key choices affecting software overhead:

- Submission mechanism
- Completion mechanism

Focus on QD1 for Latency

- Latency is just one performance factor, but a fundamental one
  - Request parallelism and batching can hide poor latency
  - Let’s optimize latency first before those other factors
- Latency-sensitive applications are most affected by latency
  - Need to complete a request before continuing
- Measure QD1 – only 1 request queued at a time
- Use small block size (4KB) to expose submission/completion latency
- More perspectives:
  - *Comparing Performance of NVMe Hard Drives in KVM, Baremetal, and Docker Using Fio and SPDK for Virtual Testbed Applications* by Mauricio Tavares at KVM Forum 2020
  - *Storage Performance Review for Hypervisors* by Felipe Franciosi at KVM Forum 2019

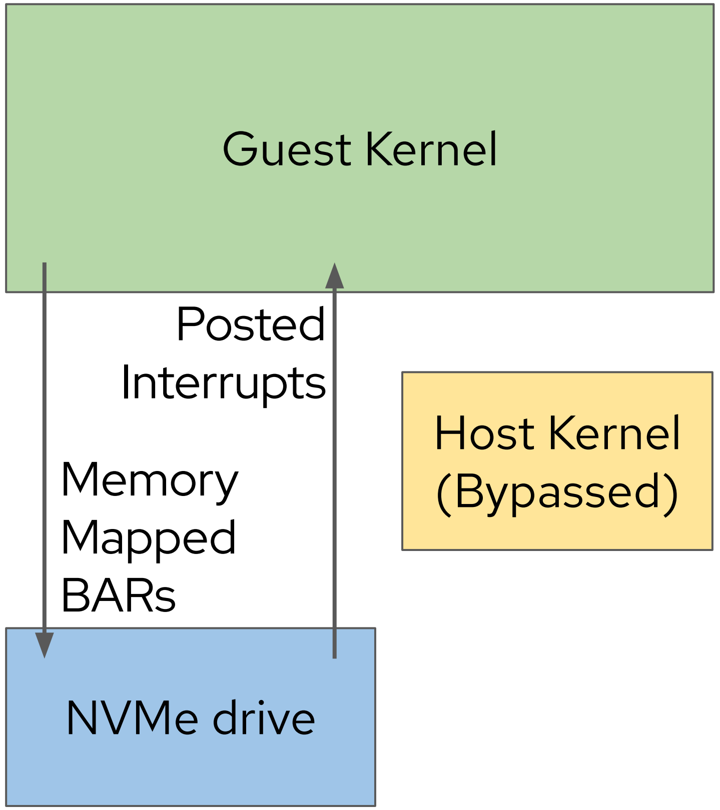
Notification Mechanisms

- Eventfd – file descriptor
  - Read file descriptor to reset counter
  - Coalesces multiple notifications
  - Relies on kernel scheduler to wake threads
  - Used by VFIO interrupts, kvm.ko ioeventfd & irqfd, Linux AIO, io_uring
- Polling – busy wait
  - Peek at memory location
  - Consumes CPU cycles
  - Used by QEMU AioContext, kvm.ko haltpoll_ns, cpuidle-haltpoll, Linux iopoll, DPDK & SPDK

PCI Device Assignment

VFIO PCI Device Assignment

- Guest runs device driver for physical PCI device
- Low overhead thanks to hardware support:
  - BAR access – memory-mapped into guest
  - IRQs – injected directly into running guest
  - DMA – accesses guest RAM via IOMMU
- Pro: Competes with bare metal performance
- Cons:
  - Limited live migration & software features
  - Guests may be tied to physical hardware
  - PCI device is dedicated to 1 guest



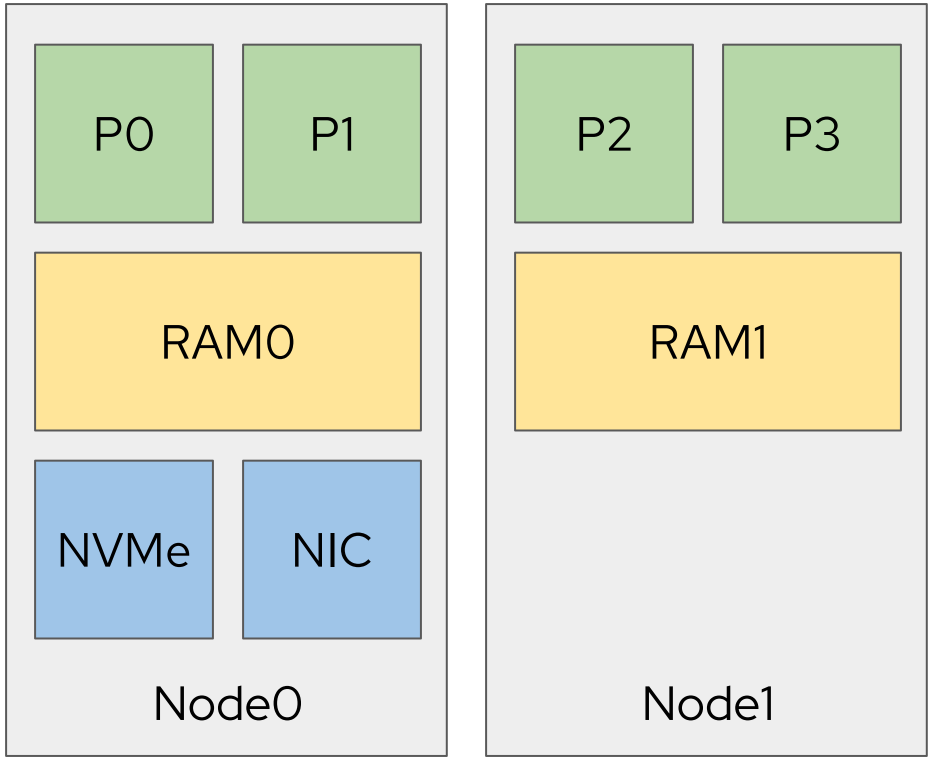
Configuring PCI Device Assignment

```
<hostdev mode='subsystem' type='pci' managed='yes'>
   <source>
      <address domain='0x0000' bus='0x5e' slot='0x00' function='0x0'/>
   </source>
</hostdev>
```


Links:
https://libvirt.org/formatdomain.html#usb-pci-scsi-devices

Red Hat

NUMA Topology



2-Node NUMA System

- Memory access fastest on local node
- Cross-node accesses are slower
- Includes L1/L2/L3 cache and main memory
- CPUs and PCI devices affected
- Tools: numactl and lstopo
- Monitoring: perf counters for CPU cross-node accesses
- More info, see Dario Faggioli's *Virtual Topology for Virtual Machines: Friend or Foe?* KVM Forum 2020 presentation

Red Hat

NUMA Tuning

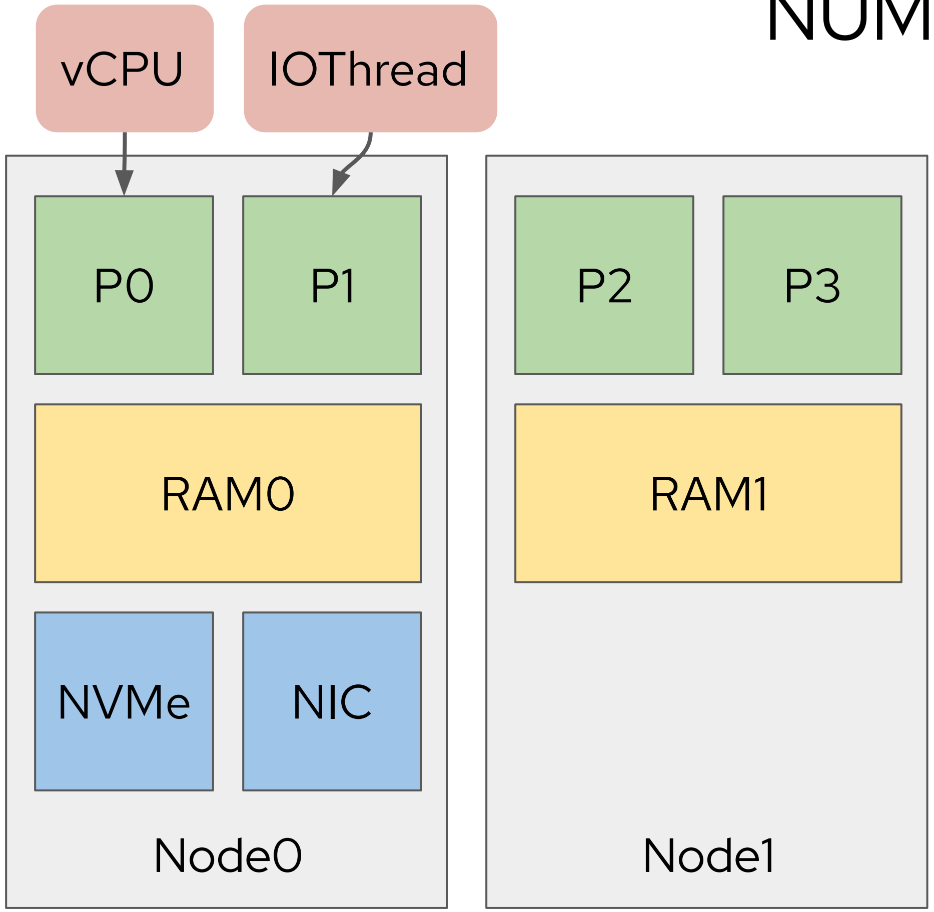
```
<cputune>
  <vcpupin vcpu=”0” cpuset=”1”/>
  <emulatorpin cpuset=”2”/>
  <iothreadpin iothread=”1” cpuset=”3”/>
</cputune>
<numatune>
  <memnode cellid=”0” mode=”strict”
           nodeset=”1”/>
</numatime>
```

- Default NUMA behavior may be suboptimal
- Manual control of NUMA is possible through pinning
- Pinning vCPU, emulator, and IOThreads produces more consistent performance results
- Supported in libvirt domain XML

Links:
https://libvirt.org/formatdomain.html#cpu-tuning
https://libvirt.org/formatdomain.html#numa-node-tuning

Red Hat

NUMA Tuning Example



2-Node NUMA System

- 1-vCPU guest
- Pin vCPU to P0
- Guest RAM only uses memory from Node0
- Pin IOThread to P1
- Why Node0? Proximity to NVMe and NIC.
- Adding another guest makes the decision harder, it depends on the workloads

Red Hat

cpuidle-haltpoll

- Halting a vCPU involves a vmexit and halting the physical CPU
  - Waking up a halted CPU has a latency cost
- cpuidle-haltpoll: When a guest vCPU is ready to halt…
  - Busy wait a little in case a task becomes schedulable
  - Decreases I/O completion latency
- kvm.ko haltpoll_ns is a similar host-side mechanism, but cpuidle-haltpoll avoids the HALT vmexit entirely

Links:
https://www.kernel.org/doc/html/latest/virt/guest-halt-polling.html

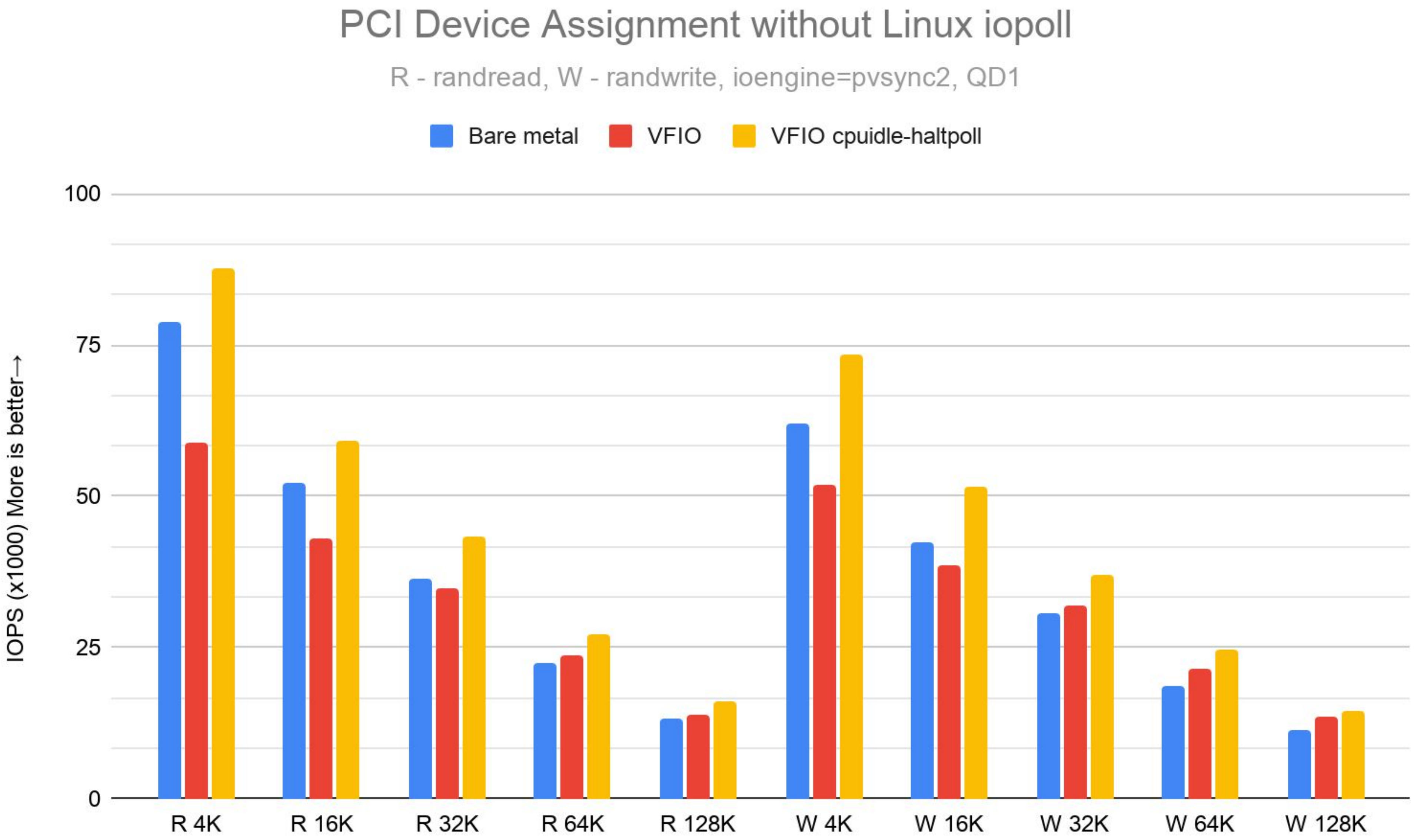
Configuring cpuidle-haltpoll

libvirt 6.10

- Requires Linux 5.4 in guest

```
<cpu mode='host-passthrough' check='none'></cpu>
<features>
    <kvm>
        <hint-dedicated state='on'/>
        <poll-control state='on'/>
    </kvm>
</features>
```

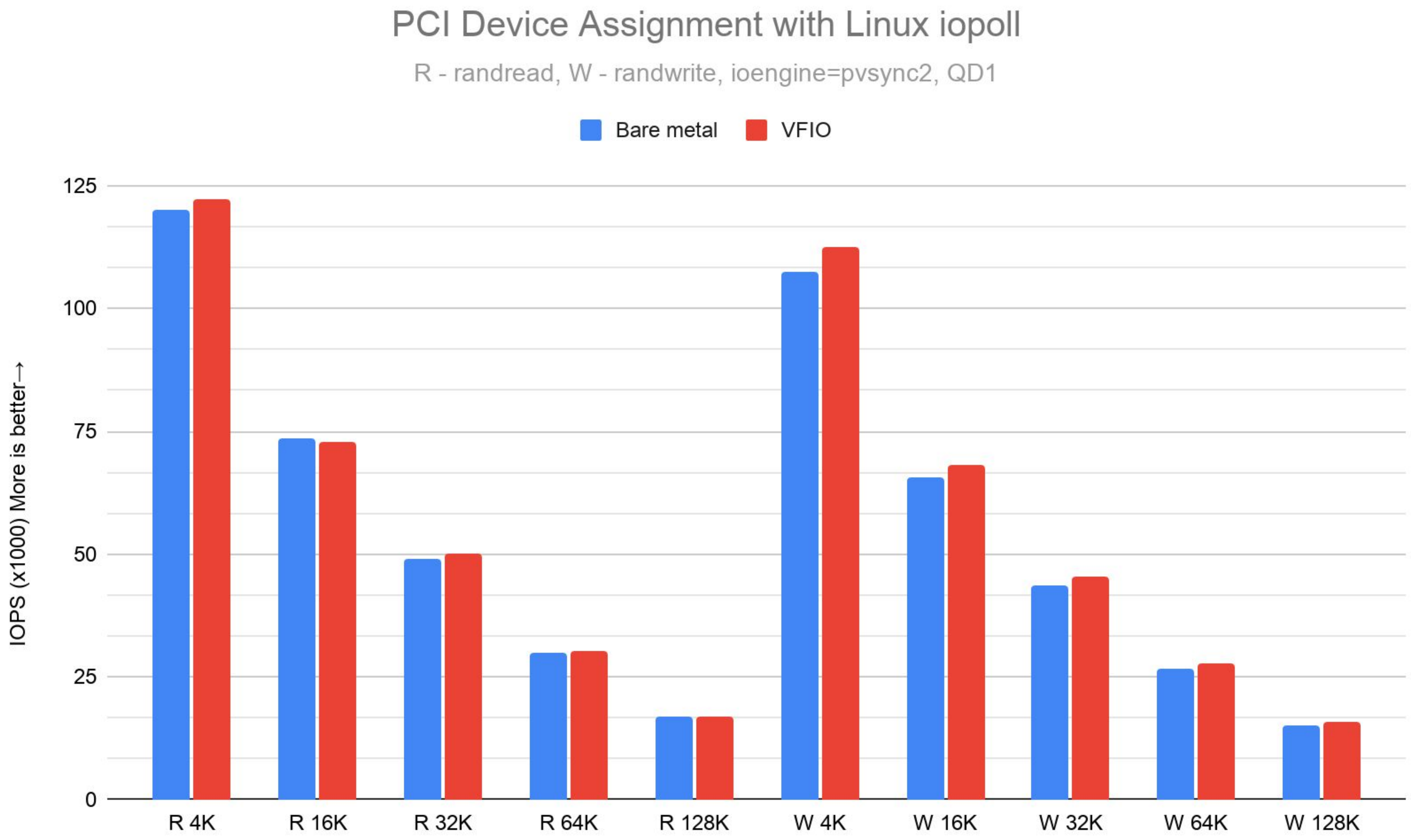

Links:
https://www.kernel.org/doc/html/latest/virt/guest-halt-polling.html


PCI Device Assignment without Linux iopoll
R - randread, W - randwrite, ioengine=pvsync2, QD1
Bare metal
VFIO
VFIO cpuidle-haltpoll
IOPS (x1000) More is better→
100
75
50
25
0
R 4K
R 16K
R 32K
R 64K
R 128K
W 4K
W 16K
W 32K
W 64K
W 128K

NVMe Linux iopoll support

- Linux nvme.ko driver supports several queue types:
  - read/write/poll
- Poll queues don't use a completion interrupt
  - Application must set RWF_HIPRI request flag
  - Kernel busy waits by calling struct blk_mq_ops->poll() driver function
- Improves completion latency more than cpuidle-haltpoll
- Module parameter: `nvme.poll_queues=4`

Red Hat



PCI Device Assignment with Linux iopoll
R - randread, W - randwrite, ioengine=pvsync2, QD1
Bare metal
VFIO
125
100
75
50
25
0
IOPS (x1000) More is better→
R 4K
R 16K
R 32K
R 64K
R 128K
W 4K
W 16K
W 32K
W 64K
W 128K

Red Hat

virtio-blk

virtio-blk

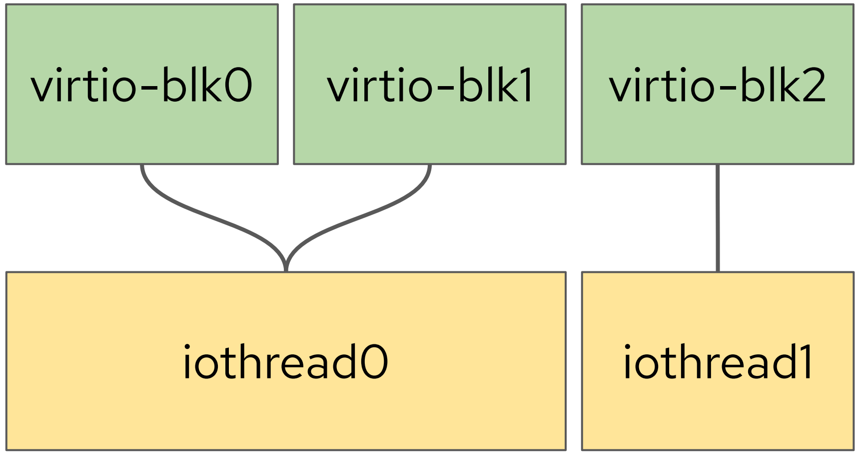
- Optimized paravirtualized storage controller
- Enable multi-queue
  - Completion interrupt handled by same vCPU that submitted request
  - Enables full Linux blk-mq behavior
  - New default: num-queues=num-vcpus QEMU 5.2
- Enable packed virtqueues
  - More efficient virtqueue memory layout

Configuring virtio-blk

```
<disk type='file' device='disk'>
    <driver name='qemu' type='raw'
            cache='none' io='native' iothread='1'
            queues='4' packed='on'/>
    <source file='/dev/nvme0n1'/>
    <target dev='vda' bus='virtio'/>
</disk>
```


Links:
https://libvirt.org/formatdomain.html#hard-drives-floppy-disks-cdroms

IOThreads



- Dedicated threads that perform device emulation & I/O
- Gives users control over CPU pinning of devices
- Adaptive polling event loop for lower latency
- N:1 devices to IOThread mapping
- Pin IOThread to NUMA node of the NVMe drive and guest RAM

Red Hat

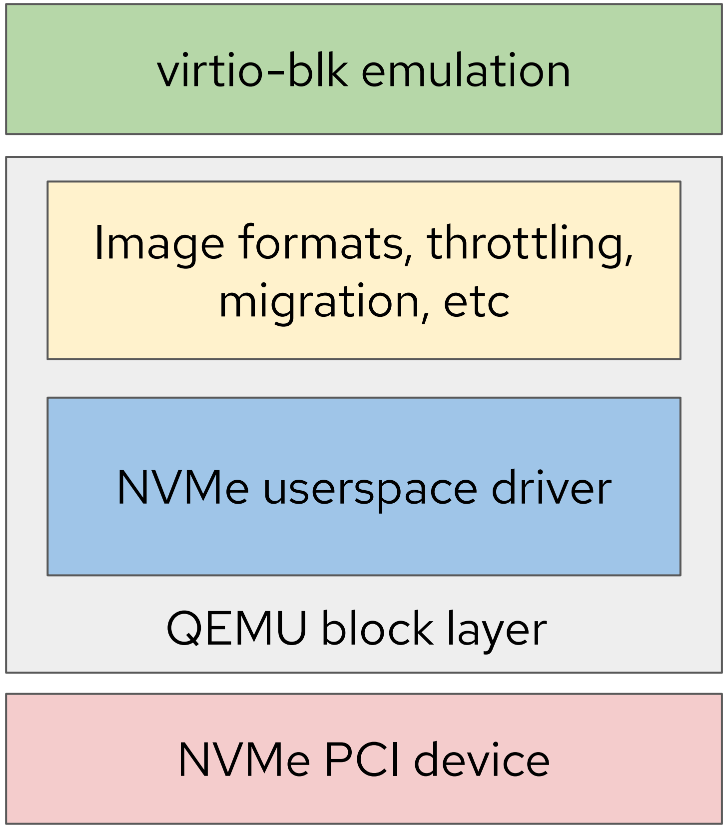
Configuring IOThreads

```
<iothreads>4</iothreads>
<cputune>
    <iothreadpin iothread="1" cpuset="10"/>
    …
</cputune>
<devices>
    <disk type='file' device='disk'>
       <driver name='qemu' iothread='1' … />
```

Links:
https://libvirt.org/formatdomain.html

QEMU Userspace NVMe Driver

- Userspace driver added in QEMU 2.12 by Fam Zheng and Paolo Bonzini, additional commands added by Maxim Levitsky
- PCI device is assigned to a single guest
- Live migration and QEMU block layer features are available!
- Non-x86 arch support, multi-queue, and more in development by Philippe Mathieu-Daudé and Eric Auger



Red Hat

Configuring the NVMe Userspace Driver

```
<disk type='nvme' device='disk'>
    <driver name='qemu' type='raw'/>
    <source type='pci' managed='yes' namespace='1'>
      <address domain='0x0000' bus='0x01' slot='0x00'
function='0x0'/>
    </source>
    <target dev='vda' bus='virtio'/>
  </disk>
```

Links:
https://libvirt.org/formatdomain.html#hard-drives-floppy-disks-cdroms

Red Hat

Polled Queues in Userspace NVMe Driver

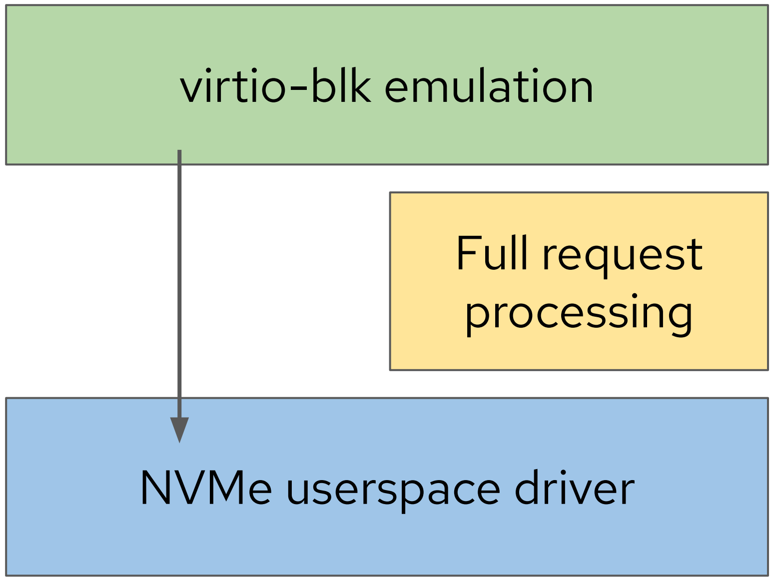
PROTOTYPE

- NVMe supports interrupts and polled-mode queues
- Upstream QEMU only creates queues with interrupts
- Patch adds polled-mode queues
- Requires io_uring so QEMU can continue to monitor file descriptors while polling for extended periods of time
  - Avoids starving file descriptors that are being monitored

Link: https://github.com/stefanha/qemu/commits/virtqueue-mq-scalability

AIO fast path PROTOTYPE

- Re-introduce asynchronous QEMU block driver interface
- Skips coroutine-based I/O request queuing in QEMU
- Only possible when software features like disk image formats, I/O throttling, storage migration, etc are inactive
- Similar ideas in 2014 by Ming Lei, Kevin Wolf, Paolo Bonzini

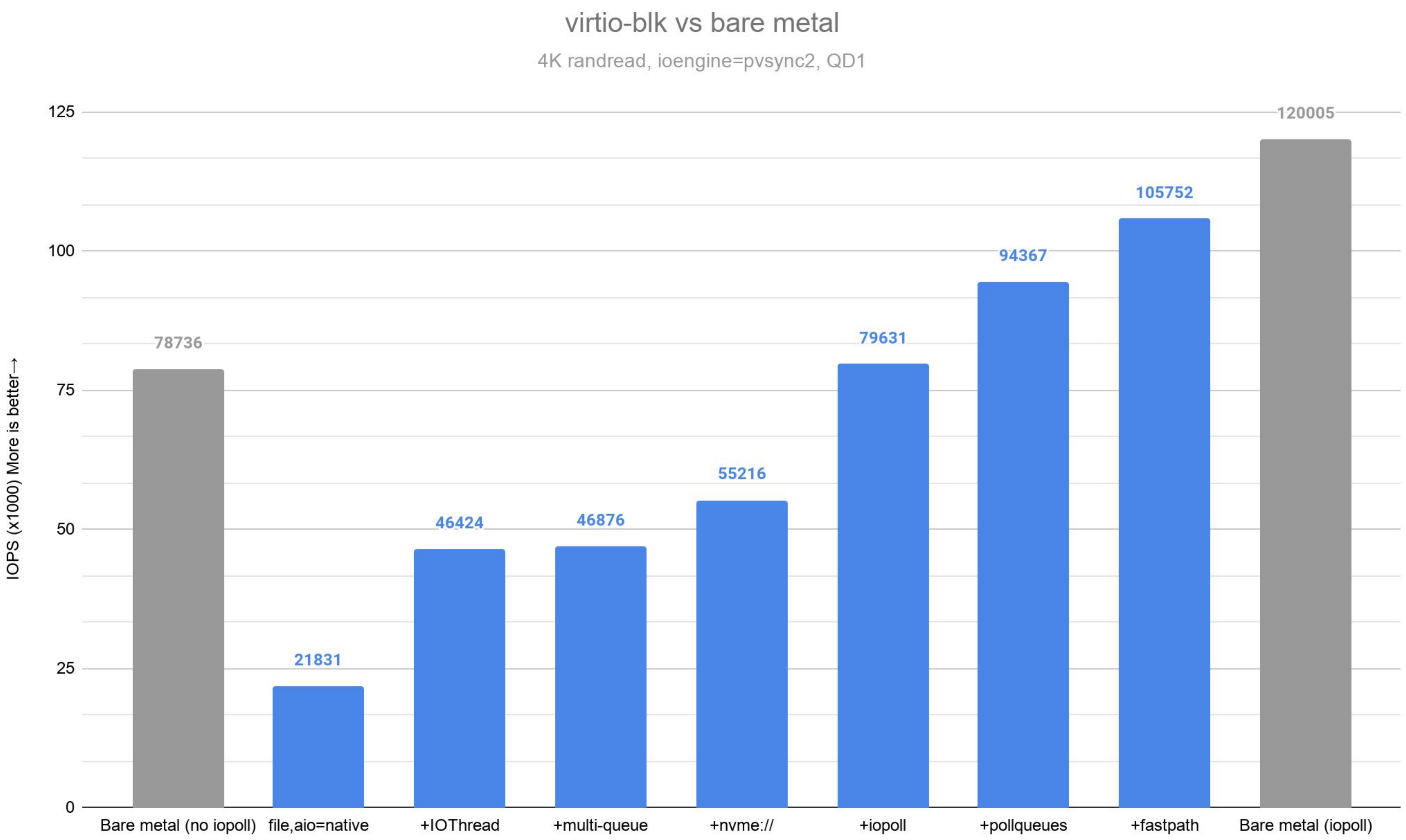


Link:
https://github.com/stefanha/qemu/commits/virtqueue-mq-scalability

virtio-blk Linux iopoll PROTOTYPE

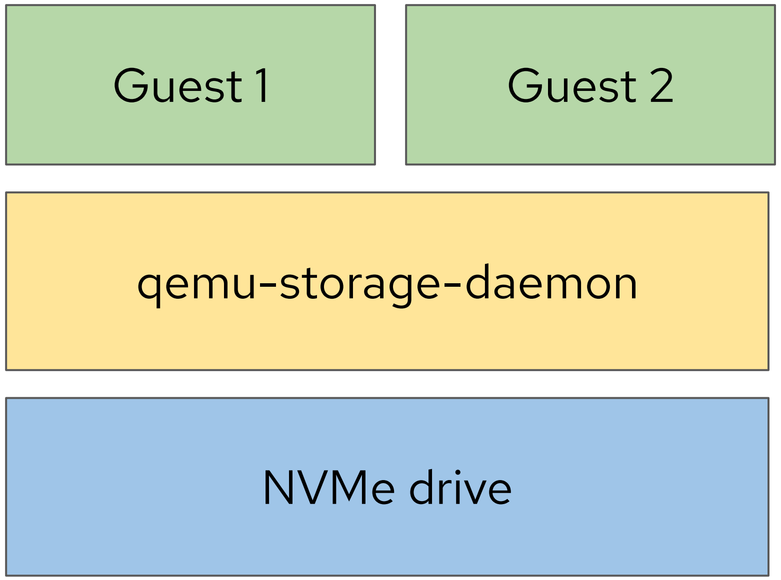
- Userspace sets RWF_HIPRI request flag
- Kernel busy waits by calling struct blk_mq_ops->poll() driver function
- Few applications use RWF_HIPRI but it's a good proof-of-concept
- Add .poll() function to virtio_blk.ko that disables virtqueue used buffer notifications
- Prototype only supports QD1

Link:
https://github.com/stefanha/linux/commits/virtio-blk-io_poll


virtio-blk vs bare metal
4K randread, ioengine=pvsync2, QD1
IOPS (x1000) More is better→
125
100
75
50
25
0
78736
21831
46424
46876
55216
79631
94367
105752
120005
Bare metal (no iopoll)
file,aio=native
+IOThread
+multi-queue
+nvme://
+iopoll
+pollqueues
+fastpath
Bare metal (iopoll)

qemu-storage-daemon QEMU 5.2

- New QEMU tool for running storage-related work in a separate process by Kevin Wolf
    - vhost-user-blk server by Coiby Xu
- Share an NVMe drive between multiple guests
- Available in qemu.git, more optimizations planned
- Bonus: many use cases possible with NBD and FUSE exports, block jobs, etc



Red Hat

Configuring qemu-storage-daemon

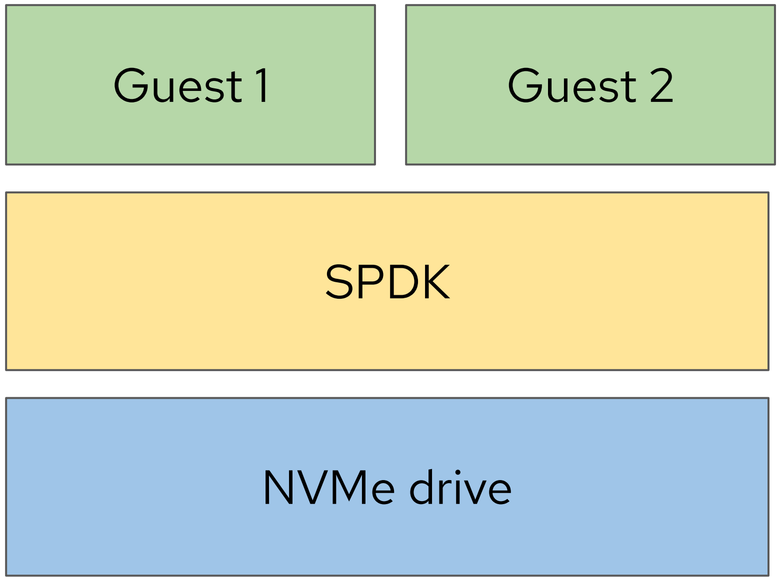

```
$ qemu-storage-daemon \
      --blockdev nvme,node-name=nvme0,... \

      --blockdev raw,node-name=drive0,file=nvme0,offset=0,size=$_16G
      --export vhost-user-blk,id=vhost-user-blk0,node-name=drive0,\
              addr.type=unix,addr.path=/tmp/vhost-user-blk0.sock

      --blockdev raw,node-name=drive1,file=nvme0,offset=$_16G,size=$_32G
      --export vhost-user-blk,id=vhost-user-blk0,node-name=drive0,\
              addr.type=unix,addr.path=/tmp/vhost-user-blk1.sock
```

Storage Performance Development Kit (SPDK)

- Polling architecture
- vhost-user-blk was created for SPDK by Changpeng Liu
- Alternative to qemu-storage-daemon with a lot in common:
  - NUMA and QEMU tuning is the same
  - Guest optimizations benefit SPDK & QEMU
  - Overlap in developer communities



Link:
https://spdk.io/

Red Hat

What about non-NVMe use cases?

- PCI Device Assignment works for other storage controllers too
- cpu-idle haltpoll, virtio-blk iopoll, etc help non-NVMe cases
- See Stefano Garzarella's *Speeding Up VM's I/O Sharing Host's io_uring Queues With Guests* KVM Forum 2020 presentation

Red Hat

Future Direction

Short Term

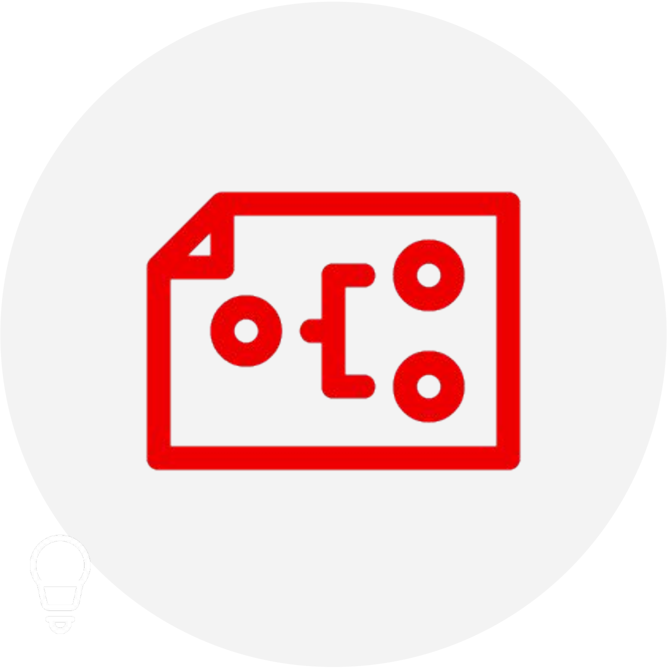
- AIO fast path & polled NVMe queues in QEMU
- Guest completion polling

Long Term

- No software in fast path, application direct to hardware

Summary

How to optimize for NVMe drives



Configuration & tuning

NUMA, cpuidle-haltpoll, IOThreads

Consider PCI Device Assignment

Minimal overhead, limited live migration & software features

Virtio-blk with QEMU Userspace NVMe Driver

Userspace NVMe driver boosts performance

qemu-storage-daemon for Sharing Drives

Share NVMe drive between multiple guests

Thank you

See QEMU blog for more resources on storage:
https://www.qemu.org/blog/category/storage/index.html

Benchmark Ansible playbooks available here:
https://github.com/stefanha/qemu-perf/commits/kvm-forum-2020

blog.vmsplice.net

stefanha@redhat.com

stefanha on #qemu IRC

Red Hat

Benchmark Configuration

- Intel® Xeon® Silver 4214 CPU @ 2.20GHz
  - 2 sockets x 12 cores x 2 hyperthreads
- 32 GB RAM
- Host kernel: 5.7.7-100.fc31.x86_64
- Guest kernel: 5.5.0
- QEMU: 4.2.0+
- NVMe: Intel Optane P4800X (8086:2701)

```
$ cat fio.job
[global]
ioengine=pvsync2
hipri=1
direct=1
runtime=60
ramp_time=5
clocksource=cpu
cpus_allowed=2
[job1]
```