IOThread Virtqueue Mapping

Improving virtio-blk SMP scalability in QEMU

Stefan Hajnoczi

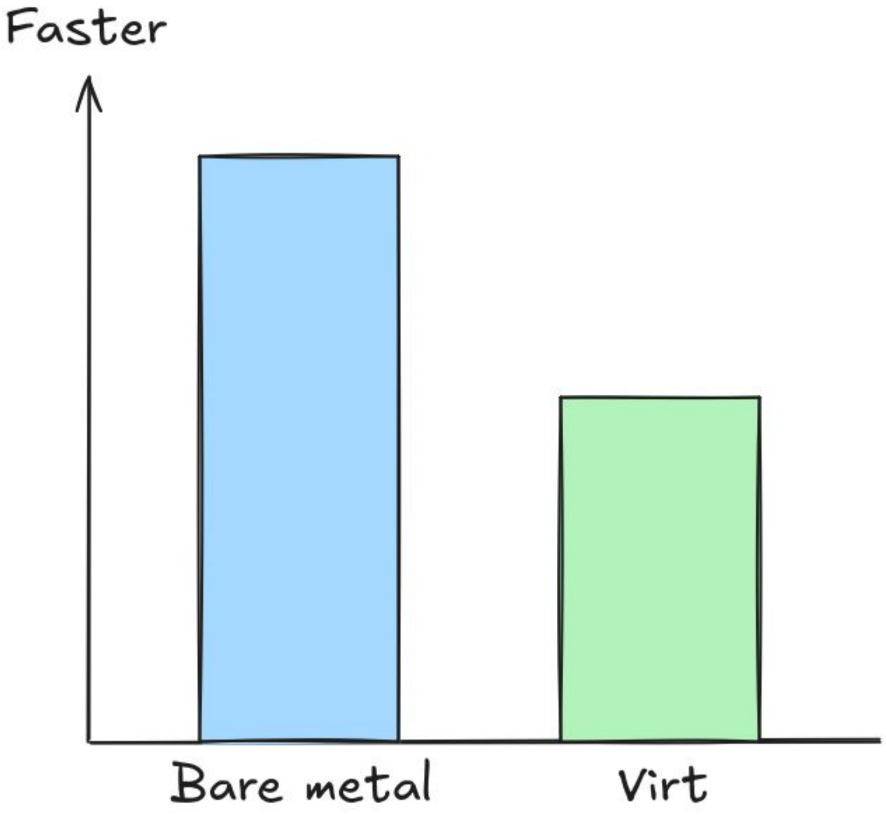
stefanha@redhat.com

Red Hat

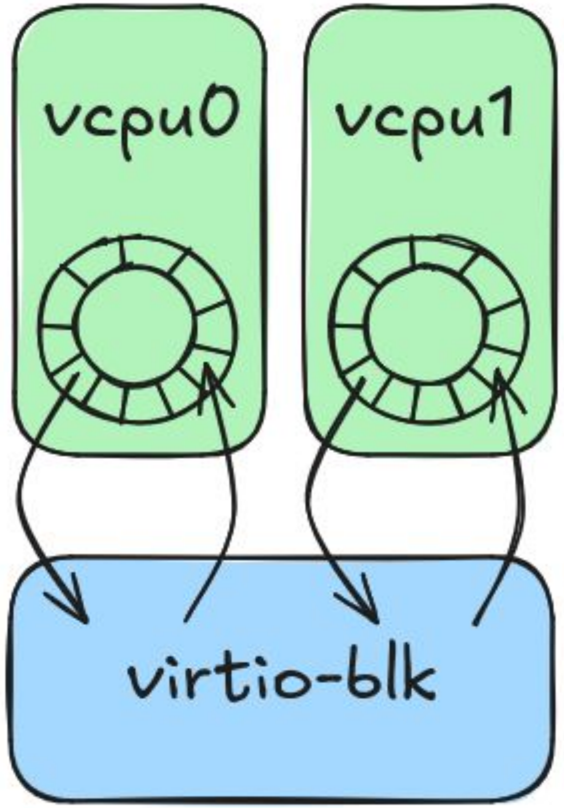
virtio-blk performance challenges in QEMU

High overhead when guests submit I/O from many vCPUs

Important since guests often have many vCPUs



Virtio-blk multi-queue



Each vCPU has its own virtqueue

- **No contention during I/O submission**
- **Completion interrupts go to submitter vCPU**

Enabled by default so why is SMP scalability still a problem?

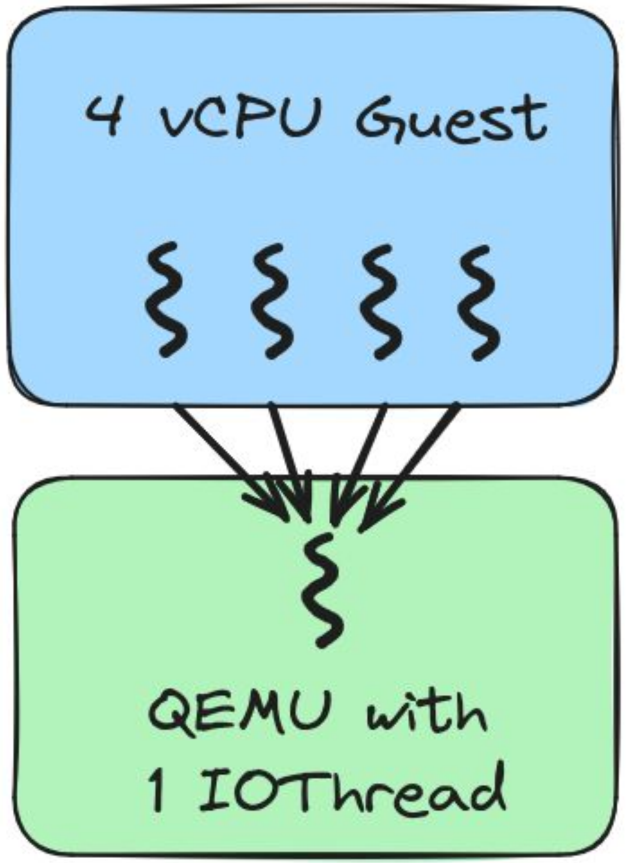
How IOThreads process virtio-blk I/O

A virtio-blk device is emulated in one QEMU thread

Its virtqueues are all emulated in that single thread!

Things to do:

- Parse virtqueue requests
- Image formats (qcow2), block layer features, etc
- Submit & complete I/O via Linux AIO or io_uring
- Build virtqueue response
- Inject completion interrupt into guest



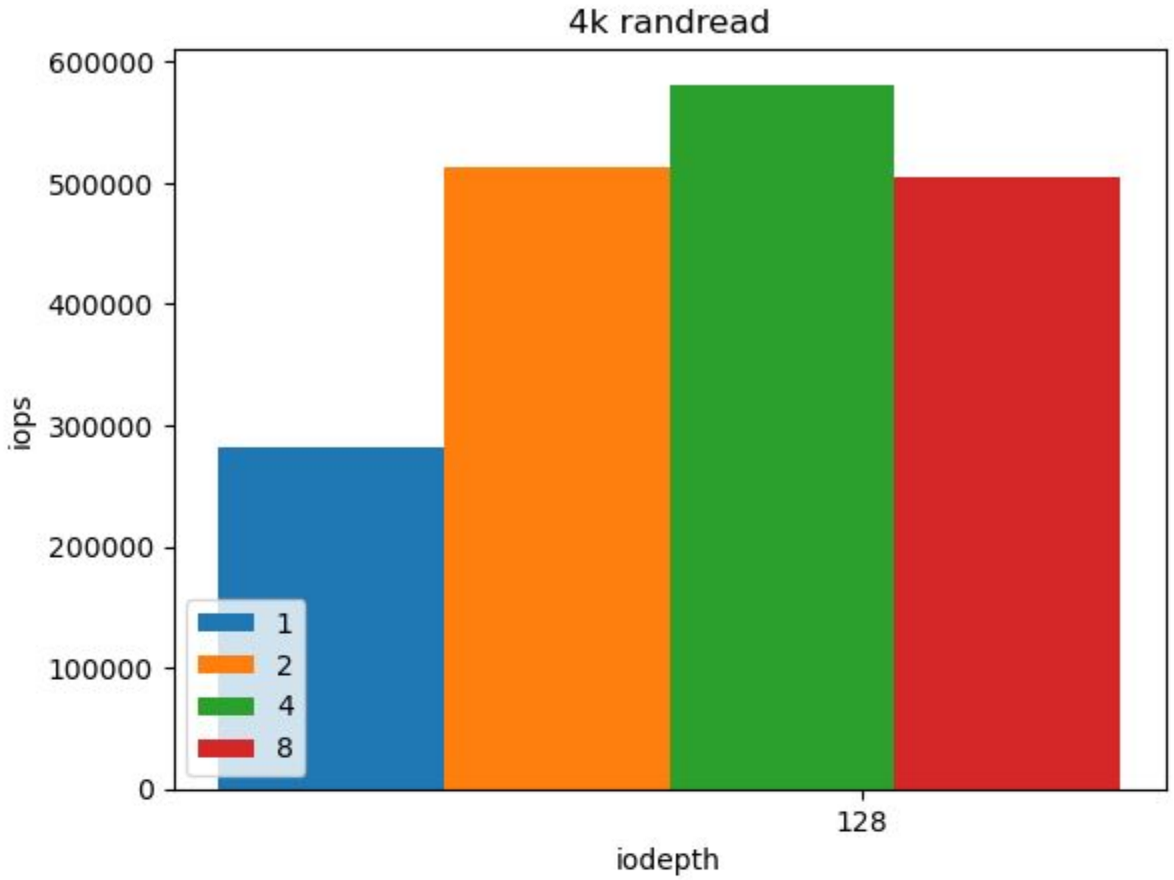
Simulating a multi-threaded virtio-blk implementation

Guest with multiple virtio-blk devices with an IOThread each accessing the same disk

Is it faster than 1 device?

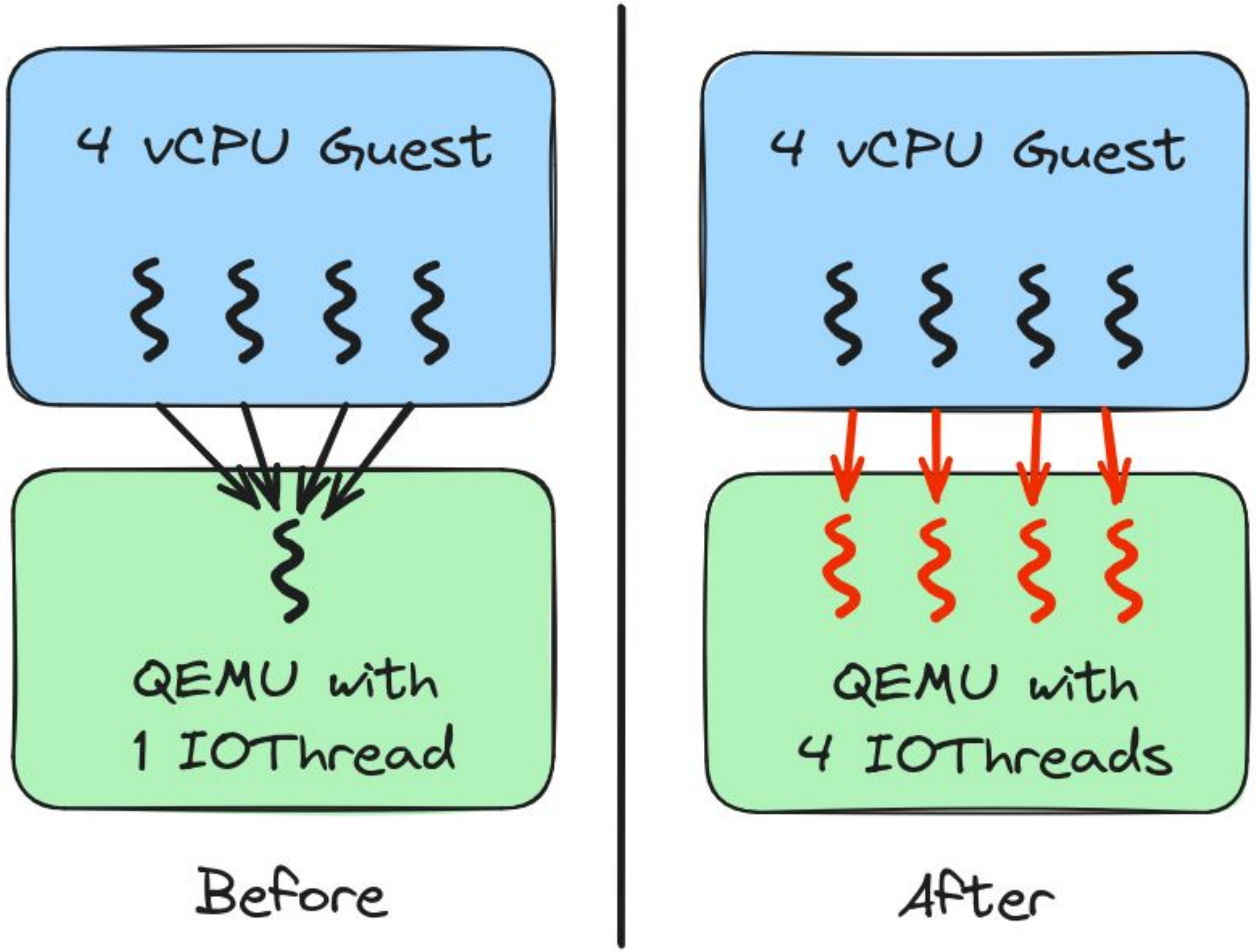
2x improvement with 4 devices

Looks promising, let's implement it properly



Benchmark details:
https://gitlab.com/stefanha/virt-playbooks/-/commit/978a2c17e1f7b744f9d207f7b083d51cdca0b454

IOThread Virtqueue Mapping == multi-queue on host



QEMU multi-queue block layer infrastructure

In QEMU 9.0 the block layer **gained multi-threaded request processing support**

See Kevin & Emanuele's [Multiqueue in the block layer]() talk for details

Team effort between 4 of us

Kevin Wolf

Emanuele Esposito

Paolo Bonzini

Libvirt syntax

Domain XML to enable IOThread Virtqueue Mapping

```
<domain>
  …
  <vcpu>4</vcpu>
  <iothreads>2</iothreads>
  …
  <devices>
    <disk …>
      <driver name='qemu' cache=’none’ io=’native’ …>
        <iothreads>
          <iothread id='1'></iothread>
          <iothread id='2'></iothread>
        </iothreads>
```

Virtqueues are assigned round-robin to IOThreads

This feature is designed for cache=’none’ io=’native’

Added in libvirt 10.0 by Peter Krempa

QEMU syntax and per-virtqueue assignment

QEMU command-line

```
$ qemu-system-x86_64 …
    -smp 4
    -object iothread,id=iothread0
    -object iothread,id=iothread1
    …
    -device ‘{“driver”: “virtio-blk-pci”,
              “iothread-vq-mapping”: [
                  {“iothread”: “iothread0”,
                   “vqs”: [0, 3]},
                  {“iothread”: “iothread1”,
                   “vqs”: [1, 2]}],
              …}’
```

Define virtio-blk-pci device with JSON syntax

Optionally specify individual virtqueue numbers to control exact assignment

Added in QEMU 9.0

IOThreads and io=”threads” are different things

1. IOThreads perform device emulation
2. io=”threads” performs I/O requests in a userspace thread pool (instead of Linux AIO or io_uring)

IOThread Virtqueue Mapping was not designed to work with io=”threads”

- **Use `<driver cache=”none” io=”native” …>`**

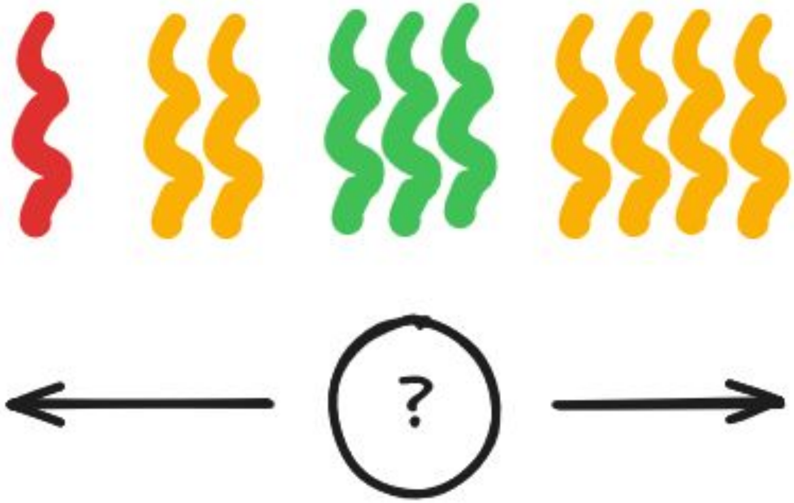
Choosing the number of IOThreads

Too few - cannot saturate drive

Too many - cannot use CPUs for application

Benchmarks on local NVMe drives suggest 4-8 threads

Measure it on your system to determine what's best



Sharing IOThreads between devices

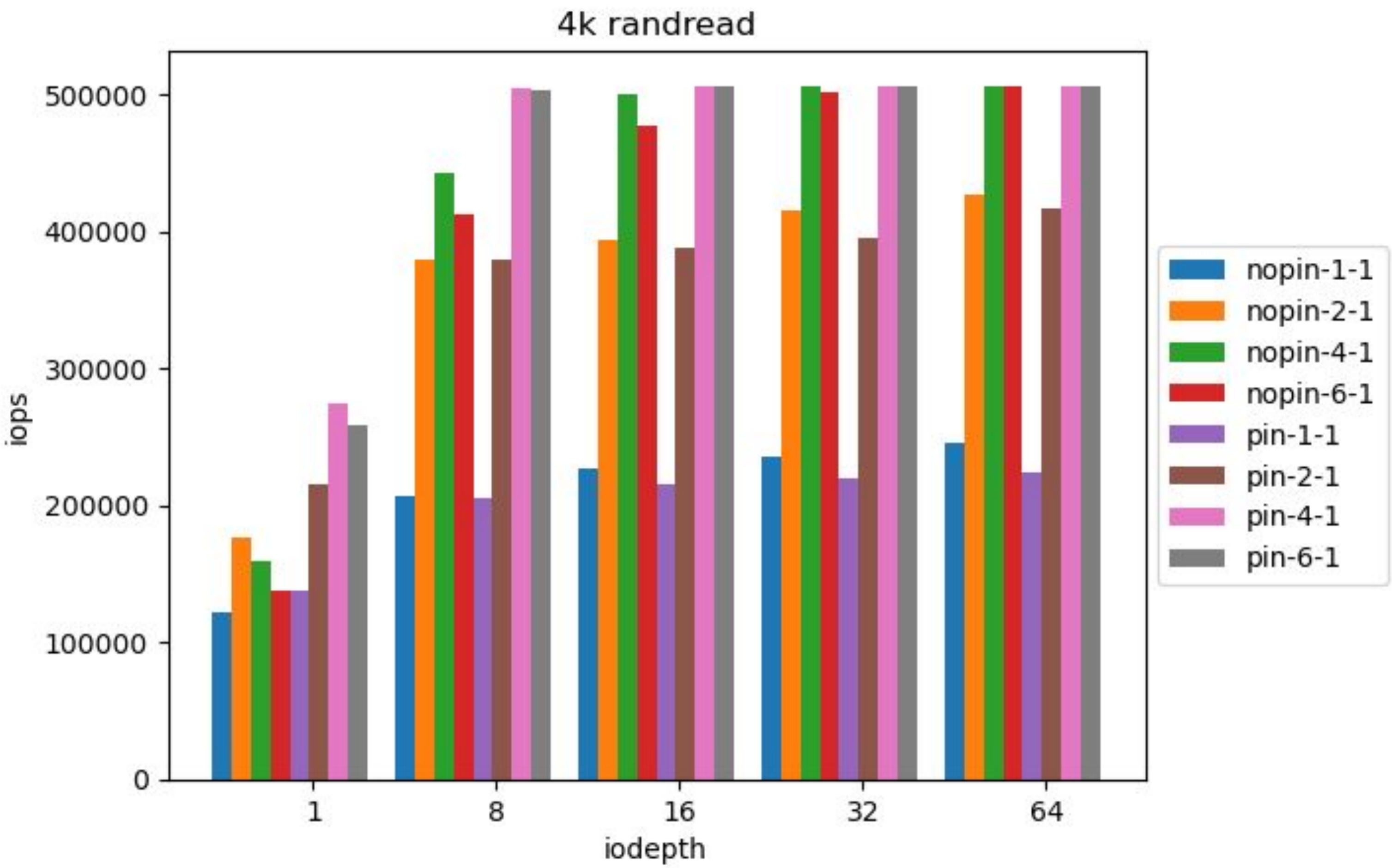
It’s okay to assign multiple virtio-blk devices to the same IOThread

Often not all virtio-blk devices are utilized equally

No need to dedicate IOThreads to low-usage devices

If you have knowledge of your workload, avoid overloading any specific IOThread

Effects of pinning IOThreads



Pinning IOThreads to dedicated CPUs reduces noise

IOThread affinity is set by libvirt’s `<iothreadpin>`

Performance impact when vCPU threads and IOThreads compete for CPU

Benchmark details:
https://gitlab.com/stefanha/virt-playbooks/-/commit/1b3e42c24d31ab1ff69fd69887a5a701e2467136

How and when to pin

Dedicate a host CPU on same NUMA node as guest RAM and storage controller PCI adapter for best performance

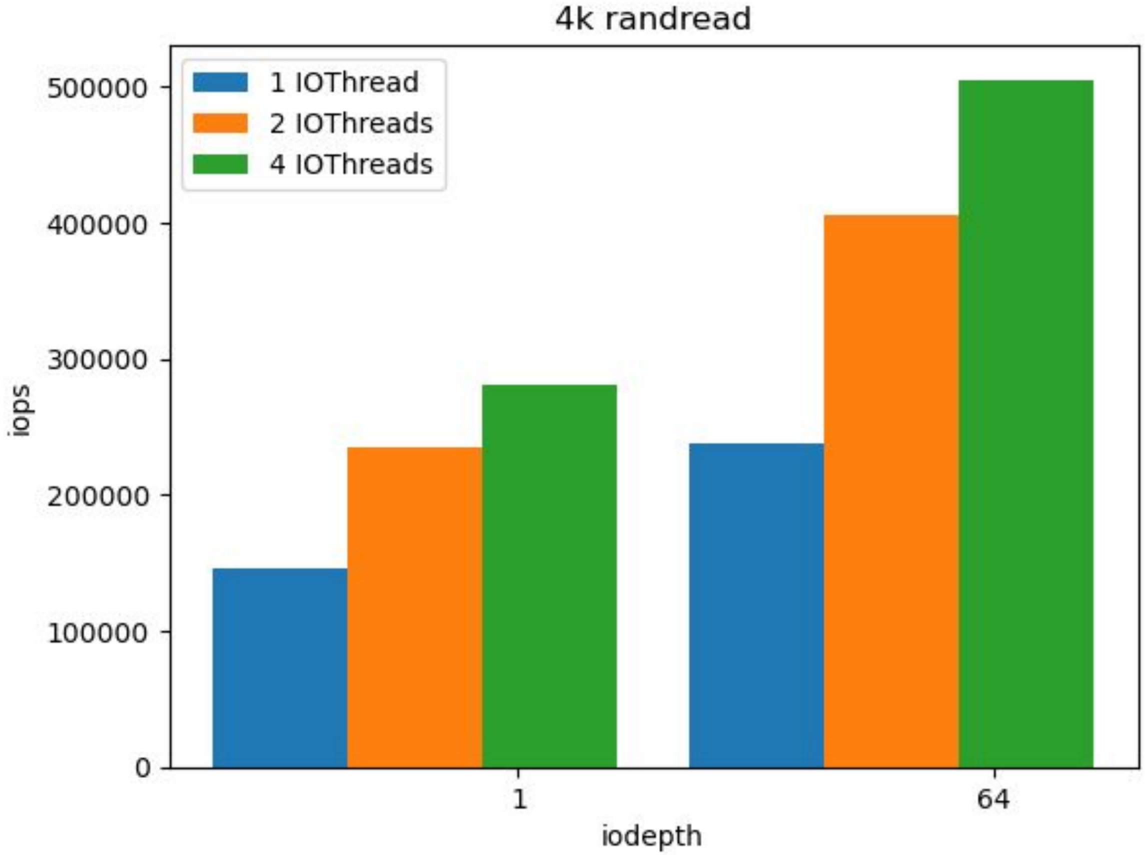
Pinning requires support from management/orchestration tool if VMs live migrate

Measuring IOThread Virtqueue Mapping

```
fio --ioengine=libaio
    --rw=randread --bs=4k
    --numjobs=8 --direct=1
    --cpus_allowed=0-7
 --cpus_allowed_policy=split
```

Intel Optane SSD DC P4800X
Raw host_device (no file system)

4 IOThreads doubles performance



Large block sizes benefit less

Easier for 1 IOThread to saturate disk at large block sizes

Fewer CPU cycles spent in QEMU

Workloads with 64+ KB block sizes may not need multiple IOThreads

Measuring database workloads

Sanjay Rao ran HammerDB on Oracle and MSSQL

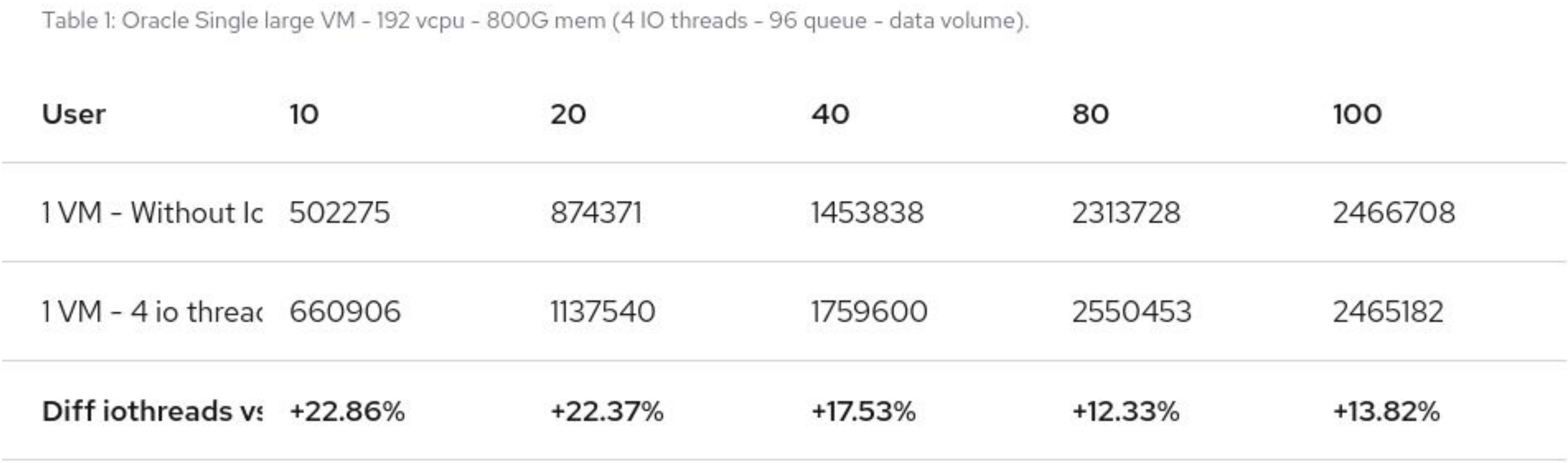


Table 1: Oracle Single large VM - 192 vcpu - 800G mem (4 IO threads - 96 queue - data volume).

User	**10**	**20**	**40**	**80**	**100**
1 VM - Without Ic	502275	874371	1453838	2313728	2466708
1 VM - 4 io threac	660906	1137540	1759600	2550453	2465182
Diff iothreads v	**+22.86%**	**+22.37%**	**+17.53%**	**+12.33%**	**+13.82%**

Starts strong but improvement drops as workload increases

Details:
https://developers.redhat.com/articles/2024/09/10/virtualized-database-io-performance-improvements-rhel-94

Oracle with 8 guests

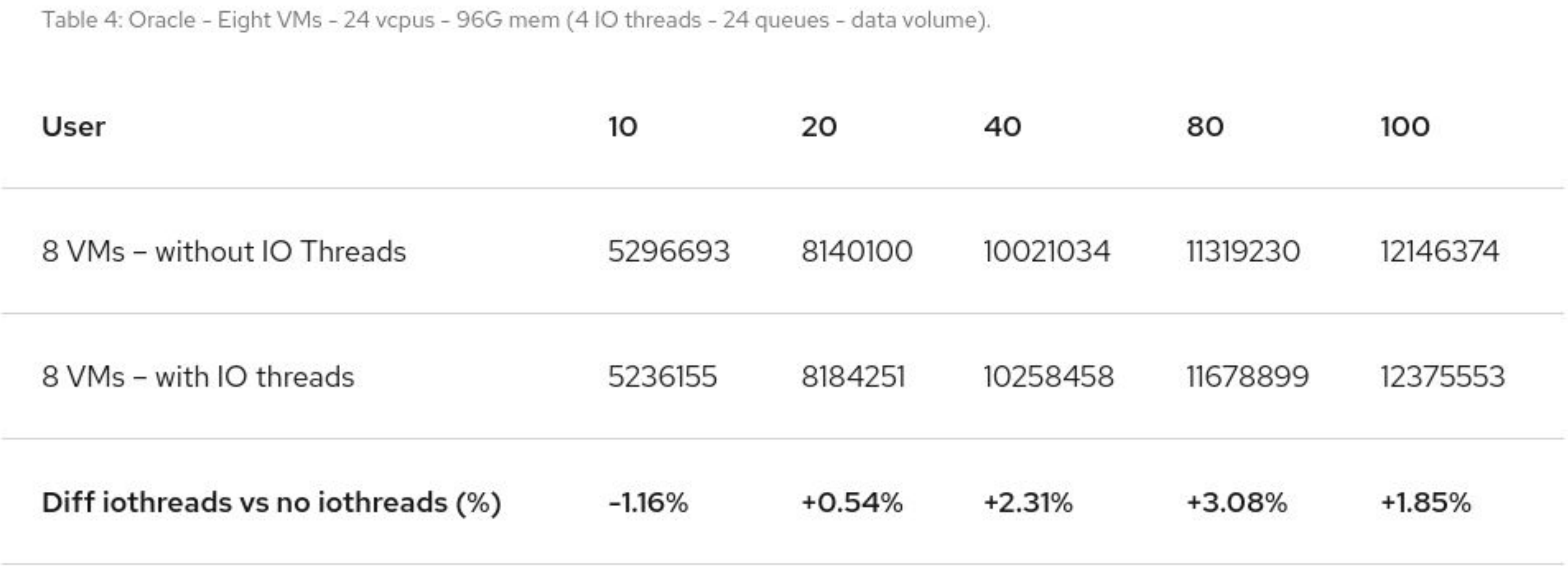


Table 4: Oracle - Eight VMs - 24 vcpus - 96G mem (4 IO threads - 24 queues - data volume).

User	10	20	40	80	100
8 VMs – without IO Threads	5296693	8140100	10021034	11319230	12146374
8 VMs – with IO threads	5236155	8184251	10258458	11678899	12375553
Diff iothreads vs no iothreads (%)	**-1.16%**	**+0.54%**	**+2.31%**	**+3.08%**	**+1.85%**

To benefit you need CPU and disk bandwidth available!

Densely packed hosts won’t benefit much

Future directions

Add support in virtio-scsi

Other VIRTIO device models can also use iothread-vq-mapping infrastructure

Optimize QEMU I/O code path to improve CPU efficiency

Summary

IOThread Virtqueue Mapping improves virtio-blk SMP scalability

Try it on your I/O-intensive workloads!

Integrate it into management/orchestration tools

More info here:

https://developers.redhat.com/articles/2024/07/09/scaling-virtio-blk-disk-io-iothread-virtqueue-mapping