SNIA DEVELOPER CONFERENCE

SDC 24

BY Developers FOR Developers

September 16-18, 2024
Santa Clara, CA

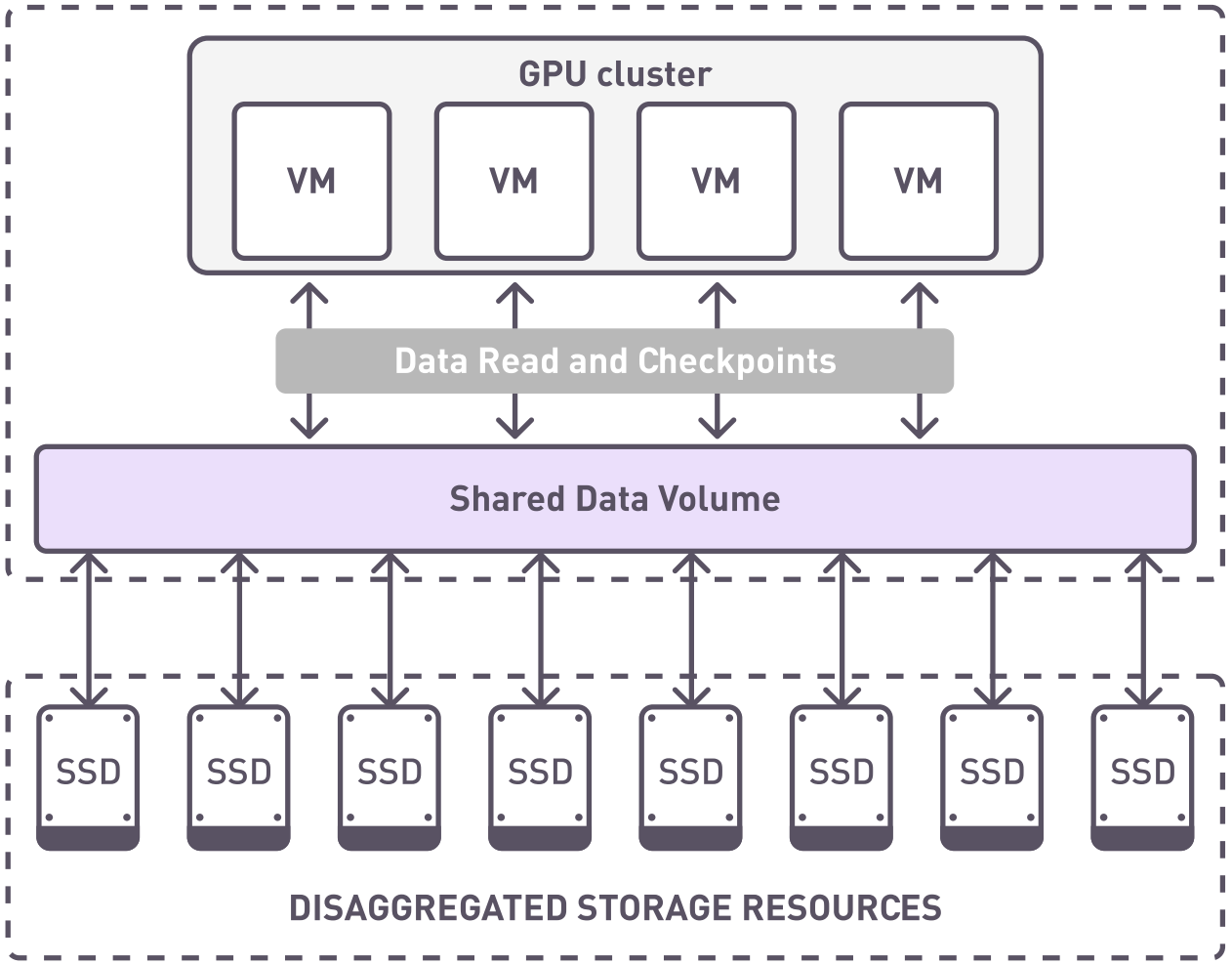
High-Performance Block Volumes in Virtual Cloud Environment

Solutions and Performance Analysis

Presented by Sergey Platonov and Heinrich von Keler

AI workloads in Cloud environments have a performance challenge

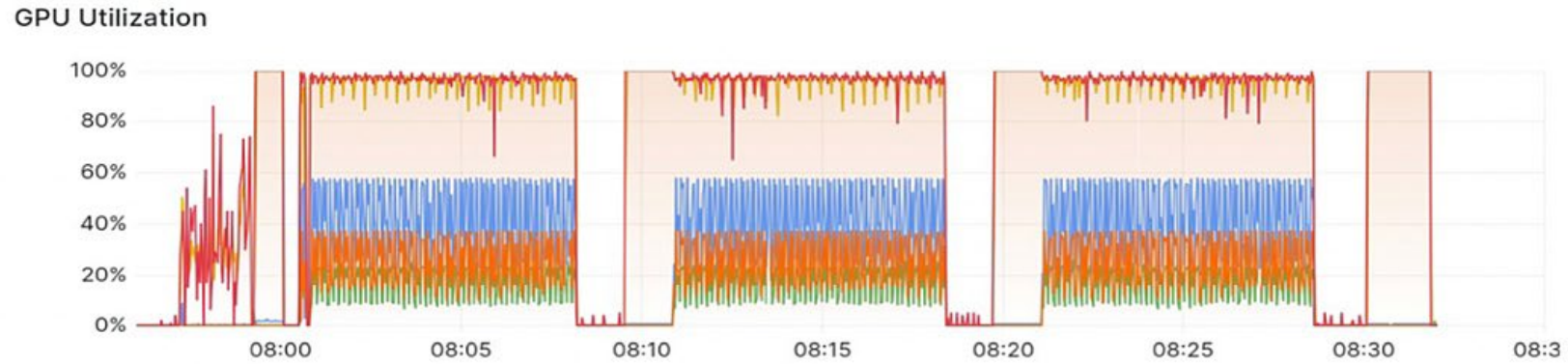
Problem: SDS performance with AI workloads in the Cloud



- **Low Performance:** SDSs often struggle to deliver the required performance for modern Cloud-based AI workloads.
- **Lack of Shared Volumes:** SDS solutions rarely offer high performance shared volume support.

Slow Storage Means GPU Underutilization

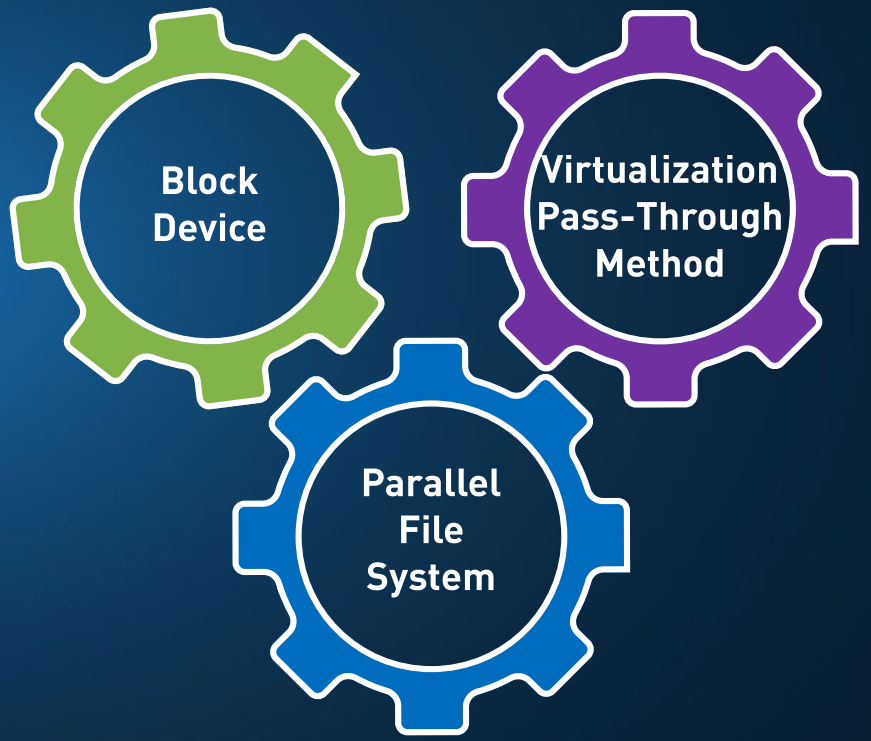
COMPUTE

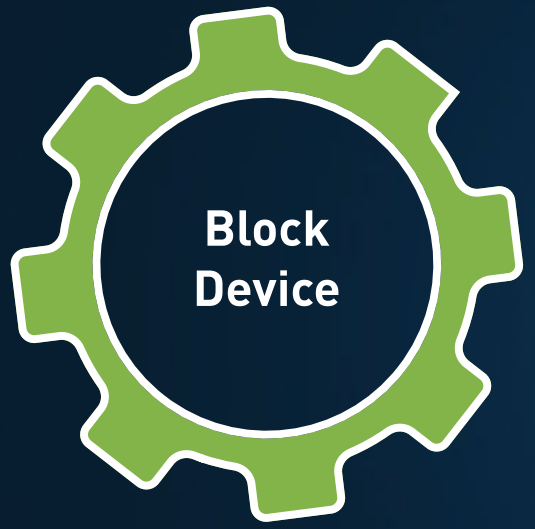


I/O

Our Goals

- Develop performant block device.
- Block device must deliver high-performance when working under a virtualized parallel file system.
- Pass this solution through cloud environments without taking a performance hit with virtualization.
- Deliver random and sequential performance for AI workloads.





xiRAID Opus: optimal solution for block device in Cloud environments

xiRAID Opus (Optimized Performance in User Space)

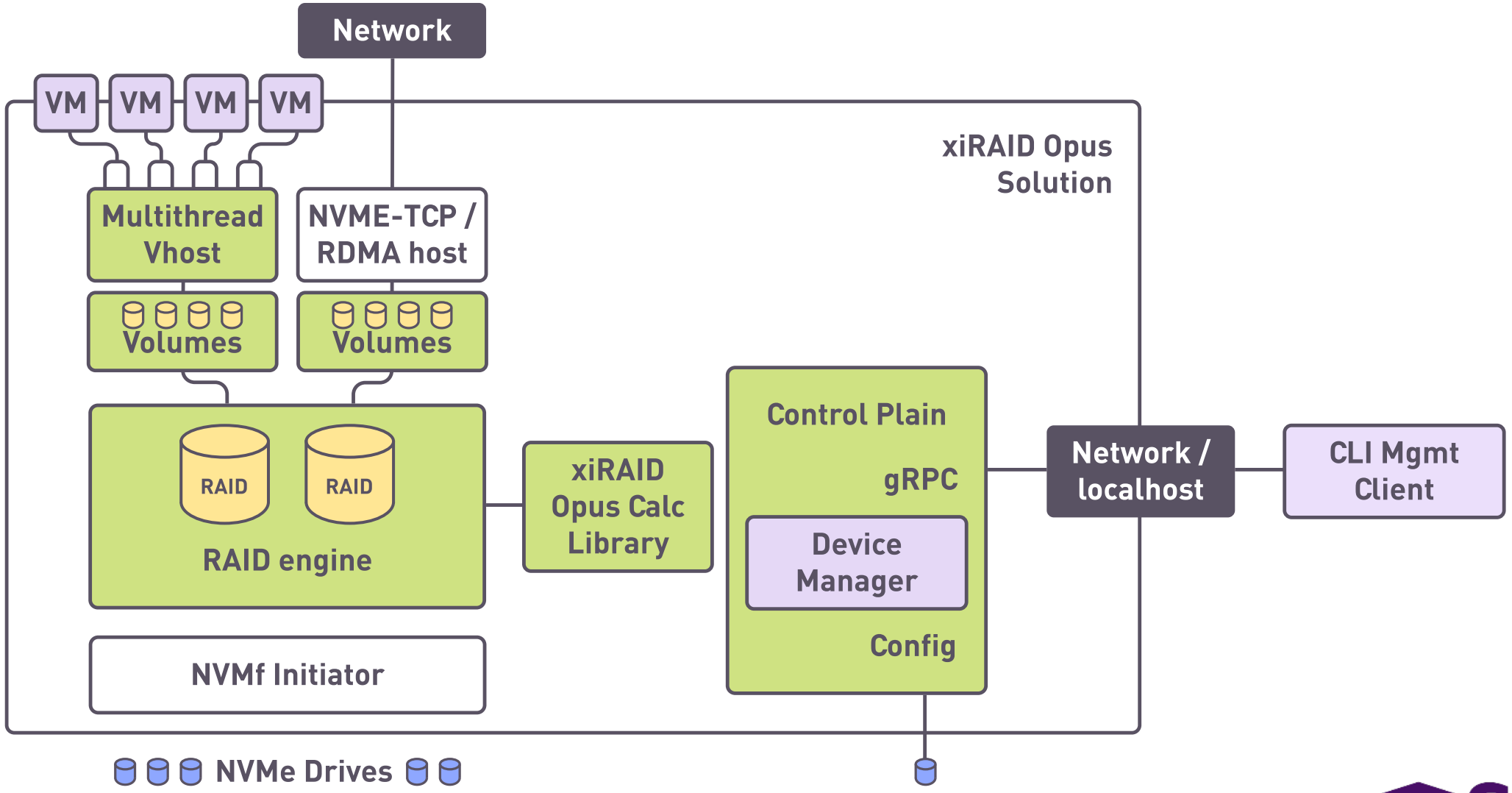
Engine designed for high-performance storage data paths in virtualized environments.

1. Creation of RAID-protected volumes.
2. Provisioning of volumes to VMs.
3. Performance Enhancements:
   - Polling: Reduces latency by actively checking for I/O completions.
   - Zero-Copy: Eliminates unnecessary data copying, increasing throughput.

	Measured single drive performance	2x RAID5 theoretical performance	XiRAID 2x RAID5 performance	Efficiency
4K Random Read (M IOPS)	2,7	65	65	**100%**
4K Random Write (M IOPS)	0,7	8	8	**100%**
Sequential Read (GB/s)	14	336	310	**92%**
Sequential Write (GB/s)	6,75	149	144	**97%**

xiRAID demonstrates world-record performance with 24 Kioxia CM7 PCIe 5 NVMe SSD drives

xiRAID Opus Architecture



Most efficient method of block device delivery in virtualized environments.

Methods for delivery of block device to VMs

VIRTIO
(1 IOT and Multiple IOT)

vhost-user-blk

VDUSE

ublk
(limited virtual environment support)

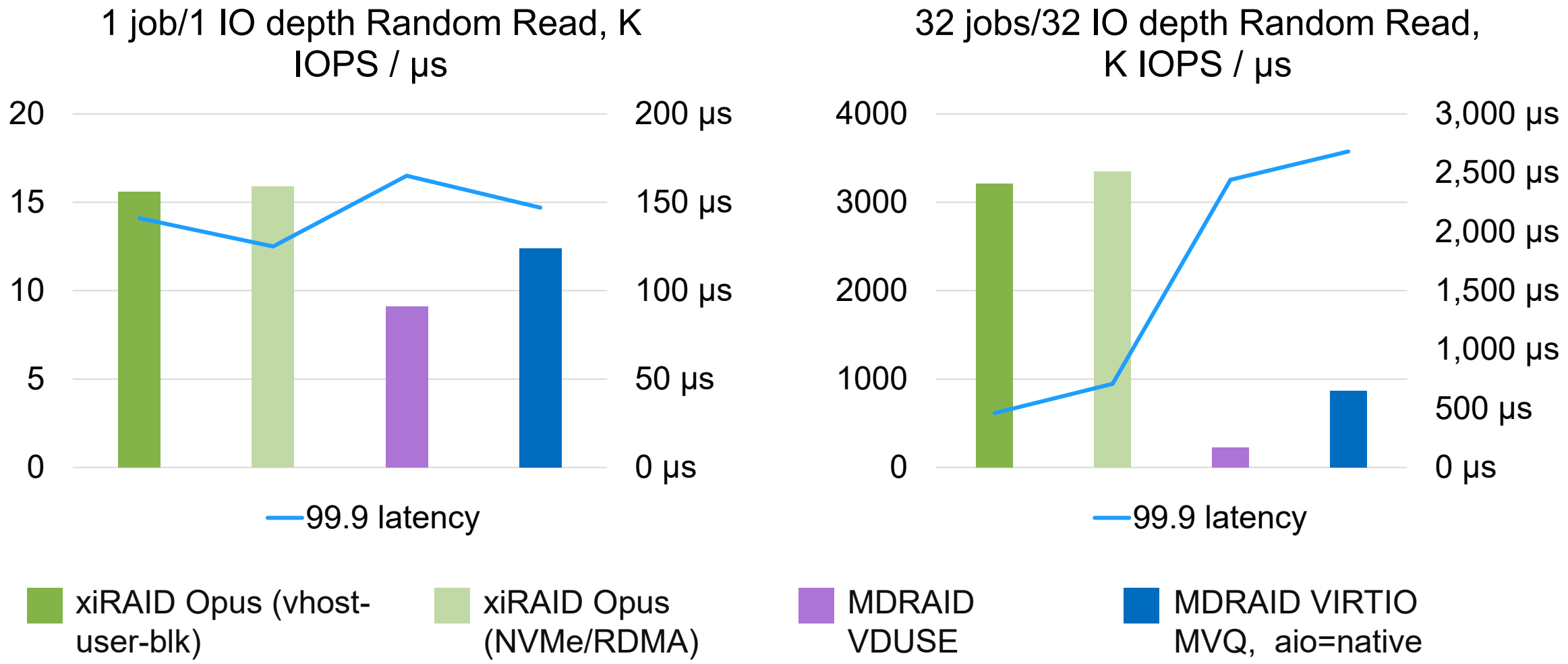
Comparing the technologies

- xiRAID Opus RAID 5 (23+1 drives) vs MDRAID RAID 0 (24 drives)
- Single Virtual Machine: 32 VCPUs, 32GB of RAM
- Rocky Linux 9, kernel-lt(6.10)

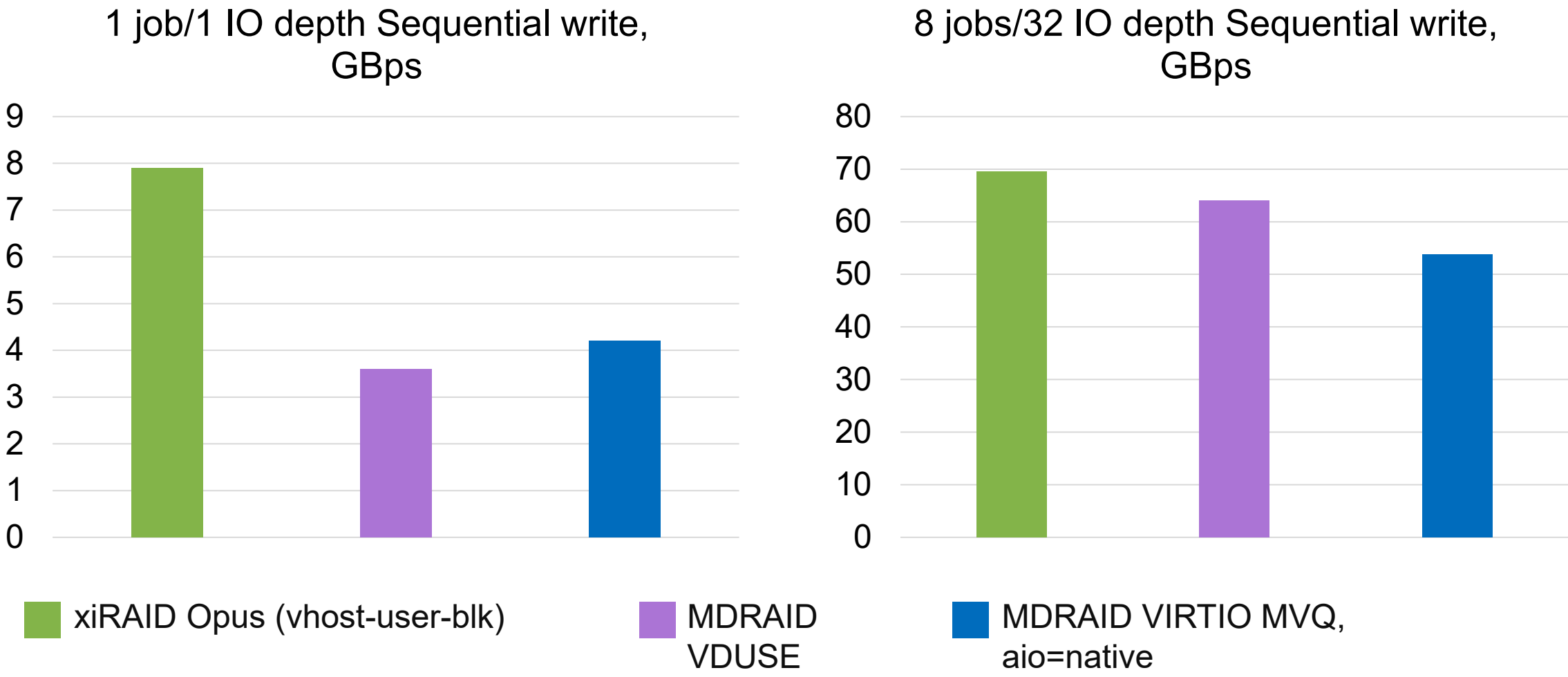
FIO v 3.36

- 4k random reads, AIO, direct_io
- full stripe writes, AIO, direct_io

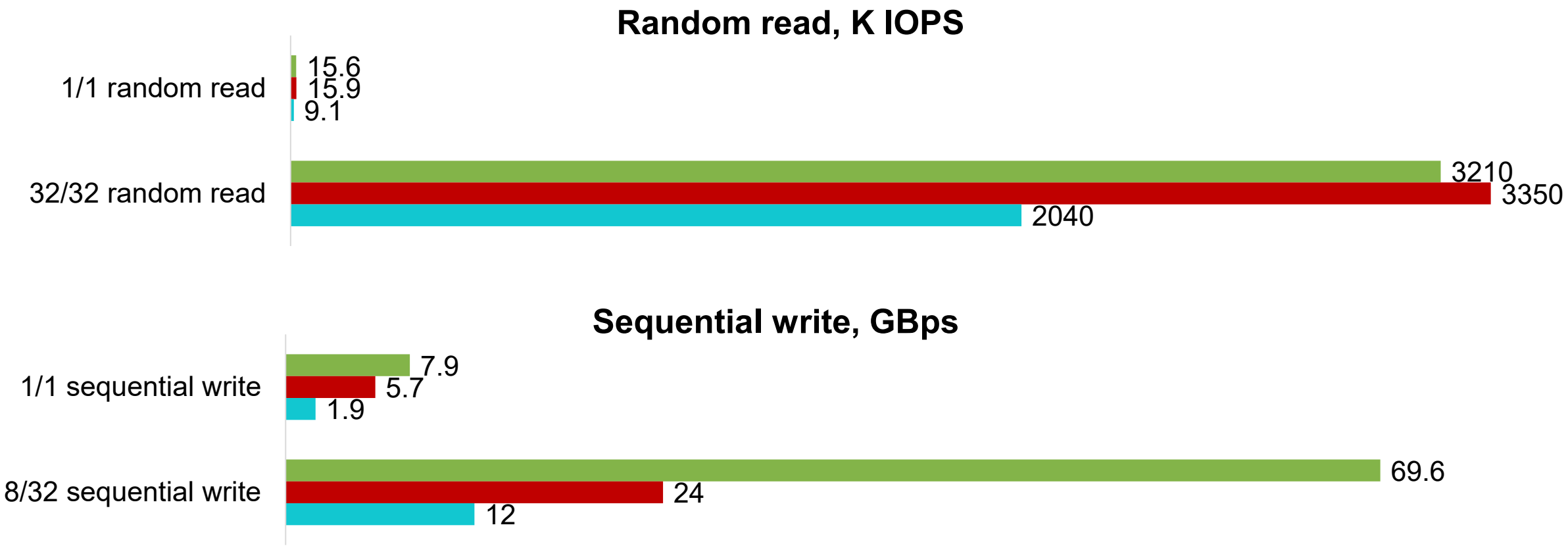
Passing shared block volume to 1 VM - Random read



Passing shared block volume to 1 VM - Sequential write



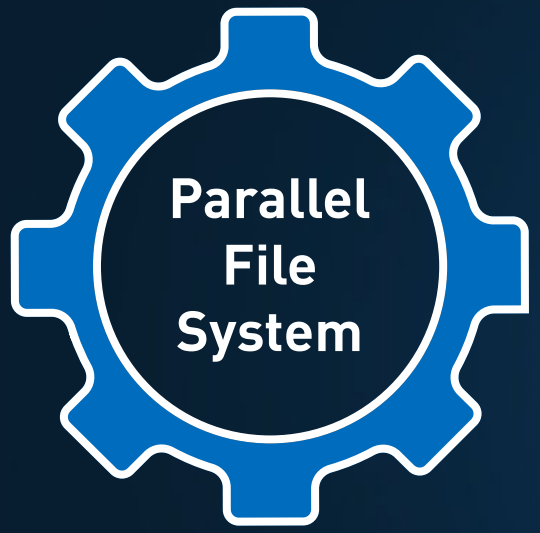
Passing shared block volume: Methods Comparison



Why we chose vhost-usr-blk = Conclusions

xiRAID Opus using our implementation of vhost-usr-blk is the only option to deliver a high-performance block device to VMs.

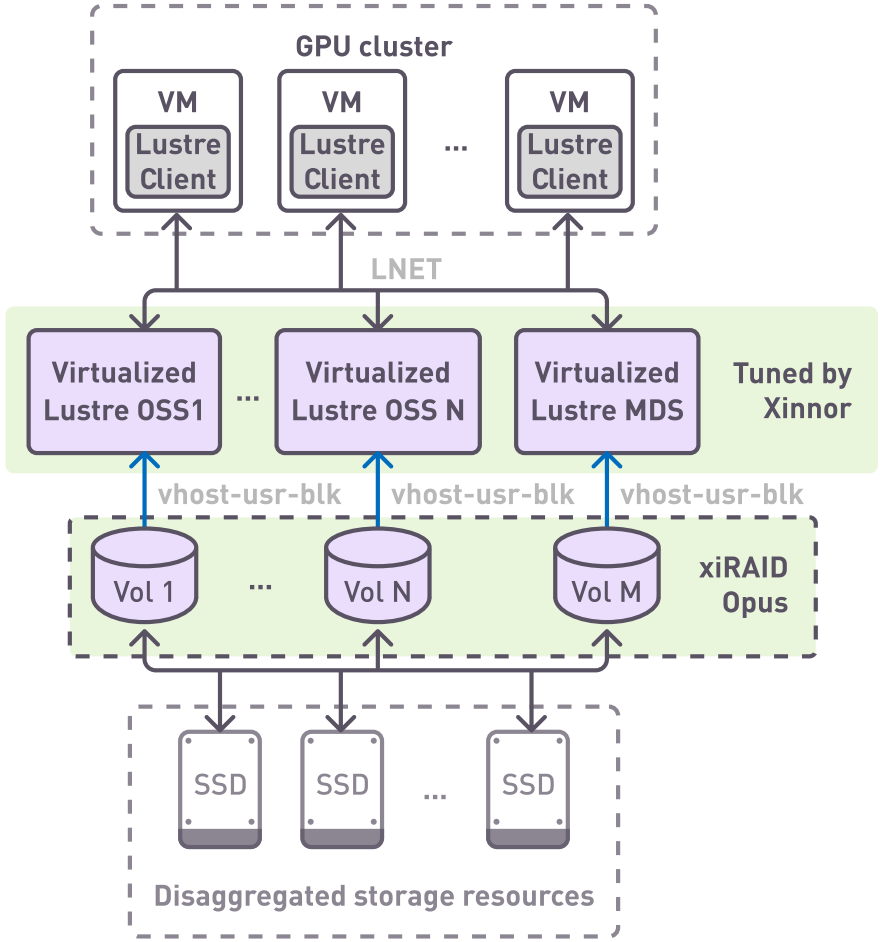
- Developed multithreading interface for VHOST User BLK.
- Reused best practices from SPDK.
- Optimized utilization of specific CPU features from AMD and Intel.
- Optimized performance even for a single virtual machine and a single block device.



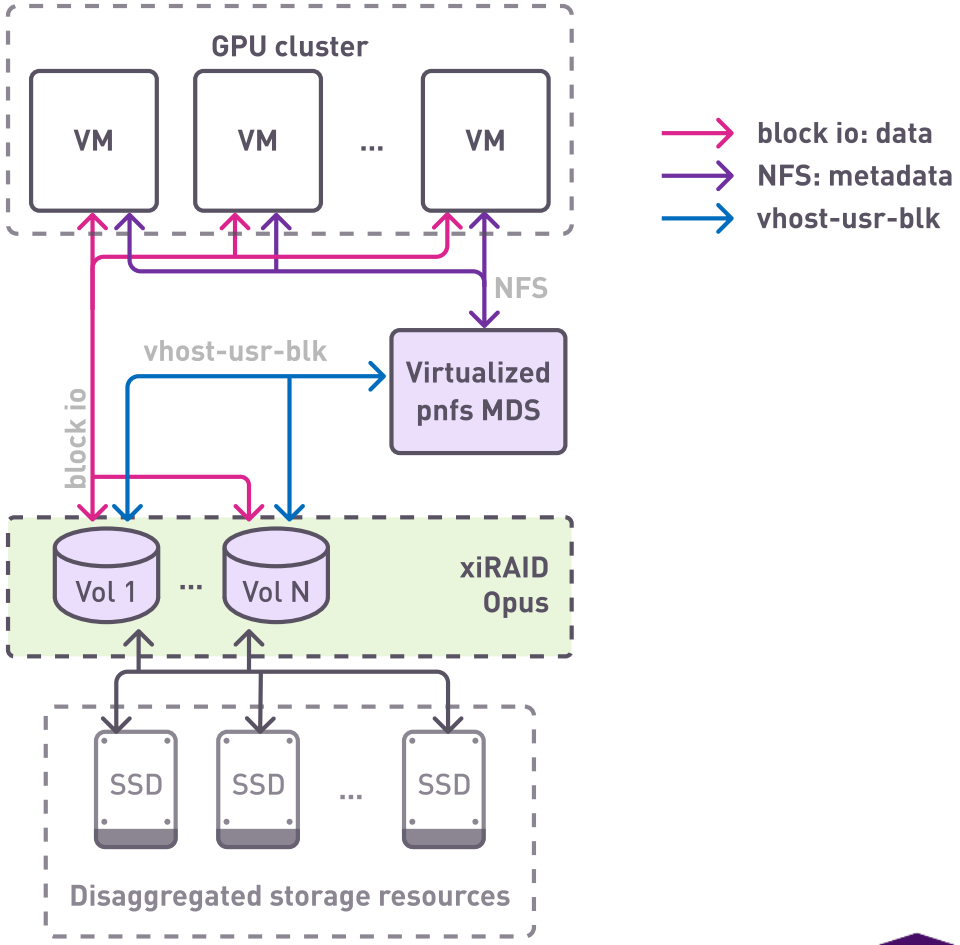
How xiRAID Opus accelerates parallel file systems in cloud environments

Architectures that we are working on

1. Lustre solution

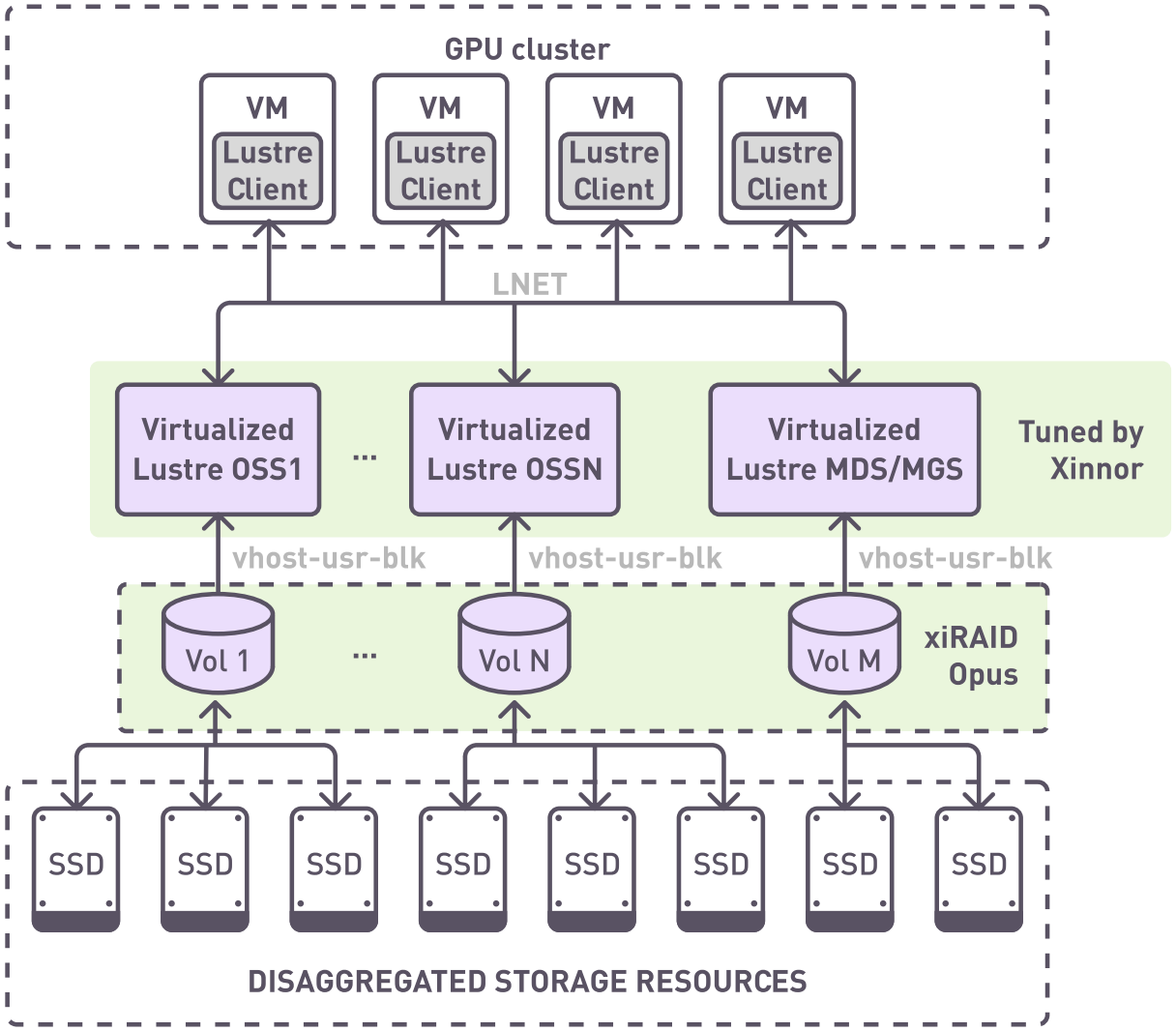


2. pNFS solution

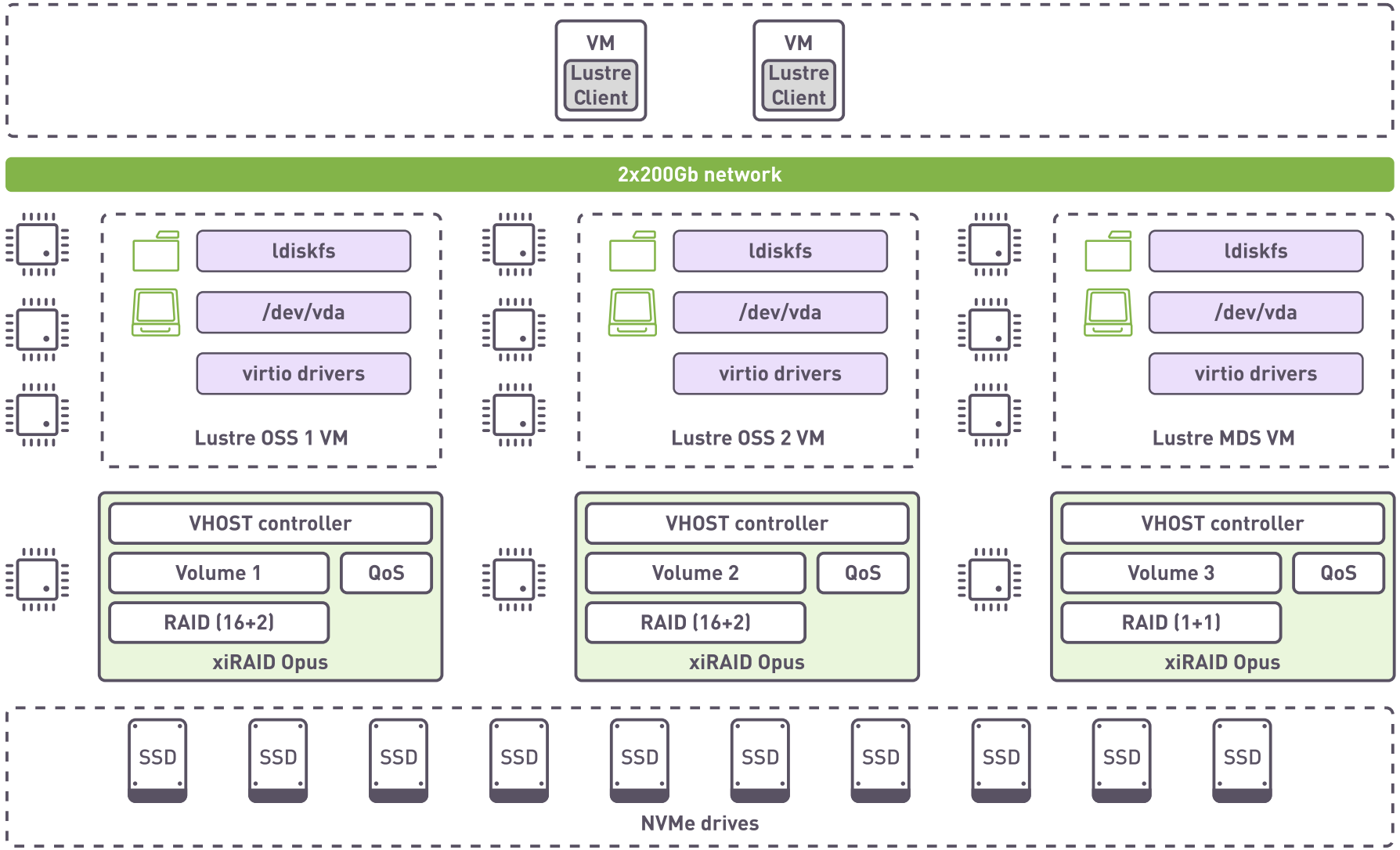


Xinnor Lustre Solution for Cloud Environments

Lustre in Cloud Environments



- Lustre is a well–known FS that is primarily used for HPC workloads
- Provides good scalability and performance for data intensive workloads
- Provides HA over shared storage



Testing Environment Details

CPU: 64-Core Processor per node (AMD 7702P)

Memory: 256 GB RAM per Node

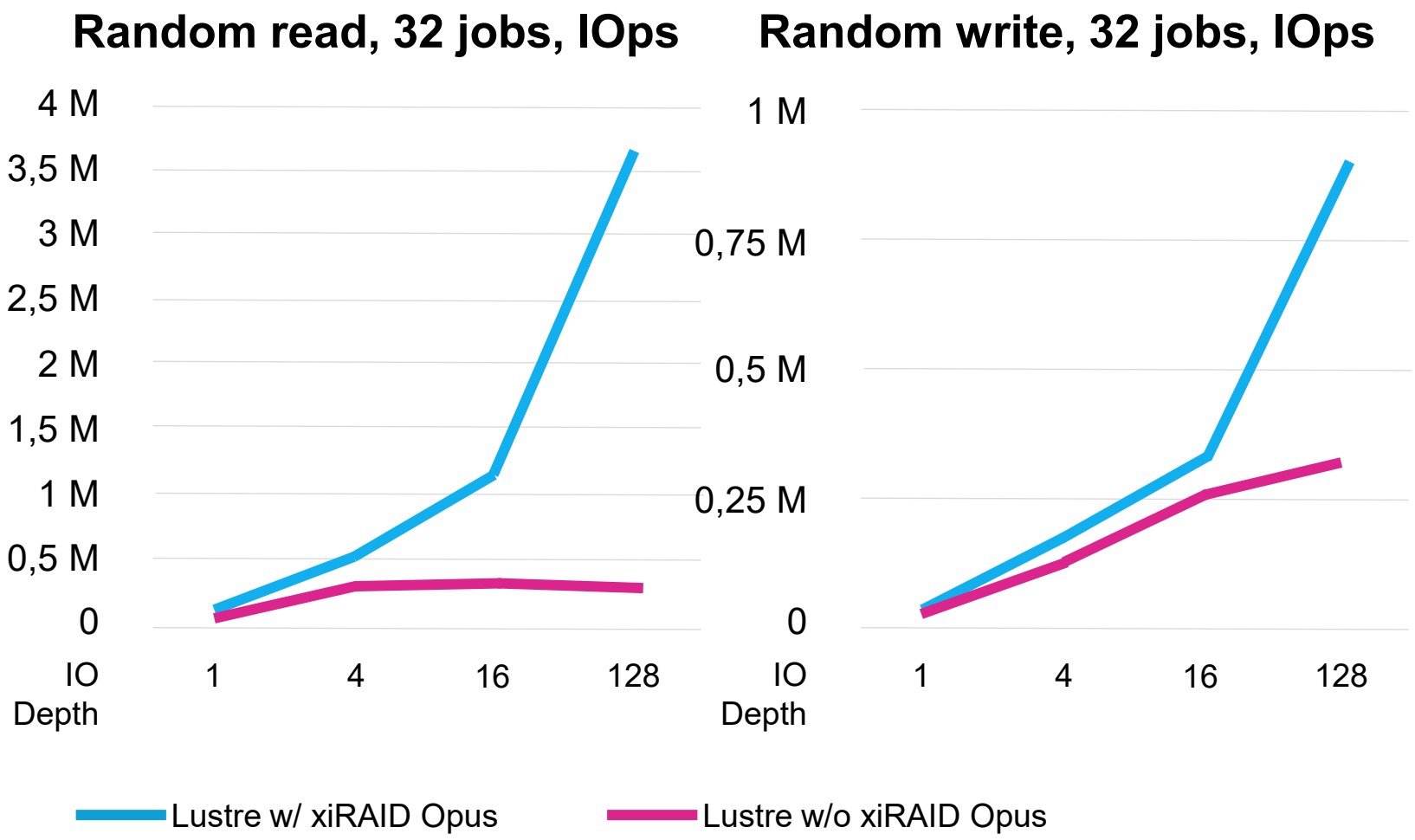
Networking: 1 x MT28908 Family [ConnectX-6] per node

Drives: 24x KIOXIA CM6-R 3.84TB (Gen 4)

Aggregated drive performance per node:

- 9M IOps 4k RR
- 3M IOps 4k RW
- 70 GBps 128k SW, SR

Lustre Solution Performance



Sequential read 1M, 32 jobs:

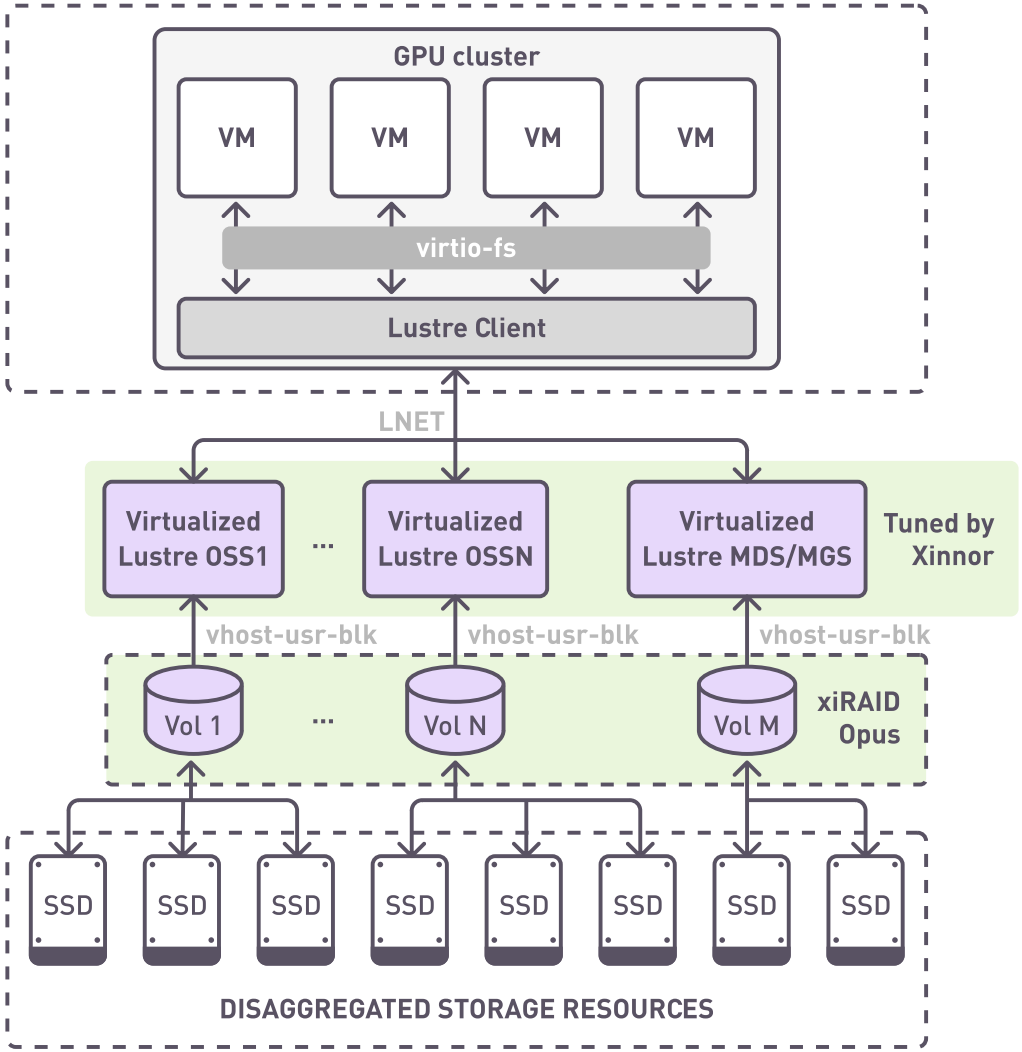
- with xiRAD Opus: 44 GB/s

Sequential write 1M, 32 jobs:

- with xiRAD Opus: 43 GB/s

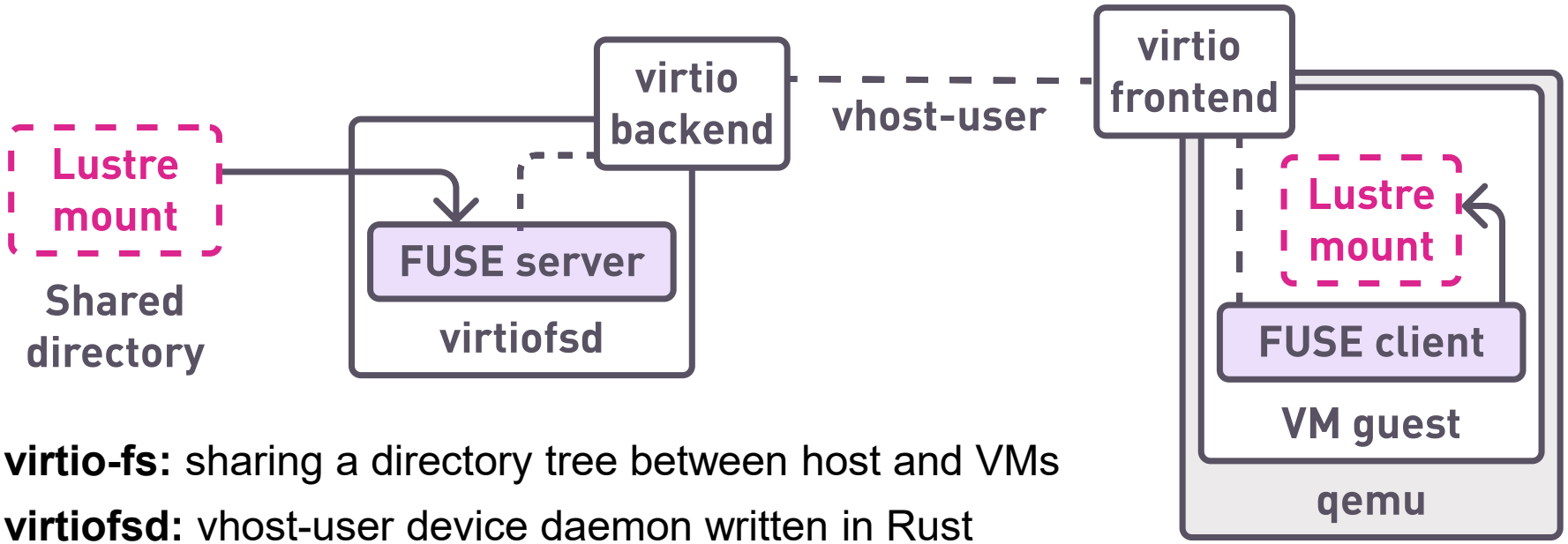
These results can be achieved with multithreaded vhost-user-blk only!

Reducing complexity of Lustre administration



- Virtiofs allows to share mounted FS on the host with VMs
- No client software or specialized networking configuration needed

Virtio FS Upsides = simplicity

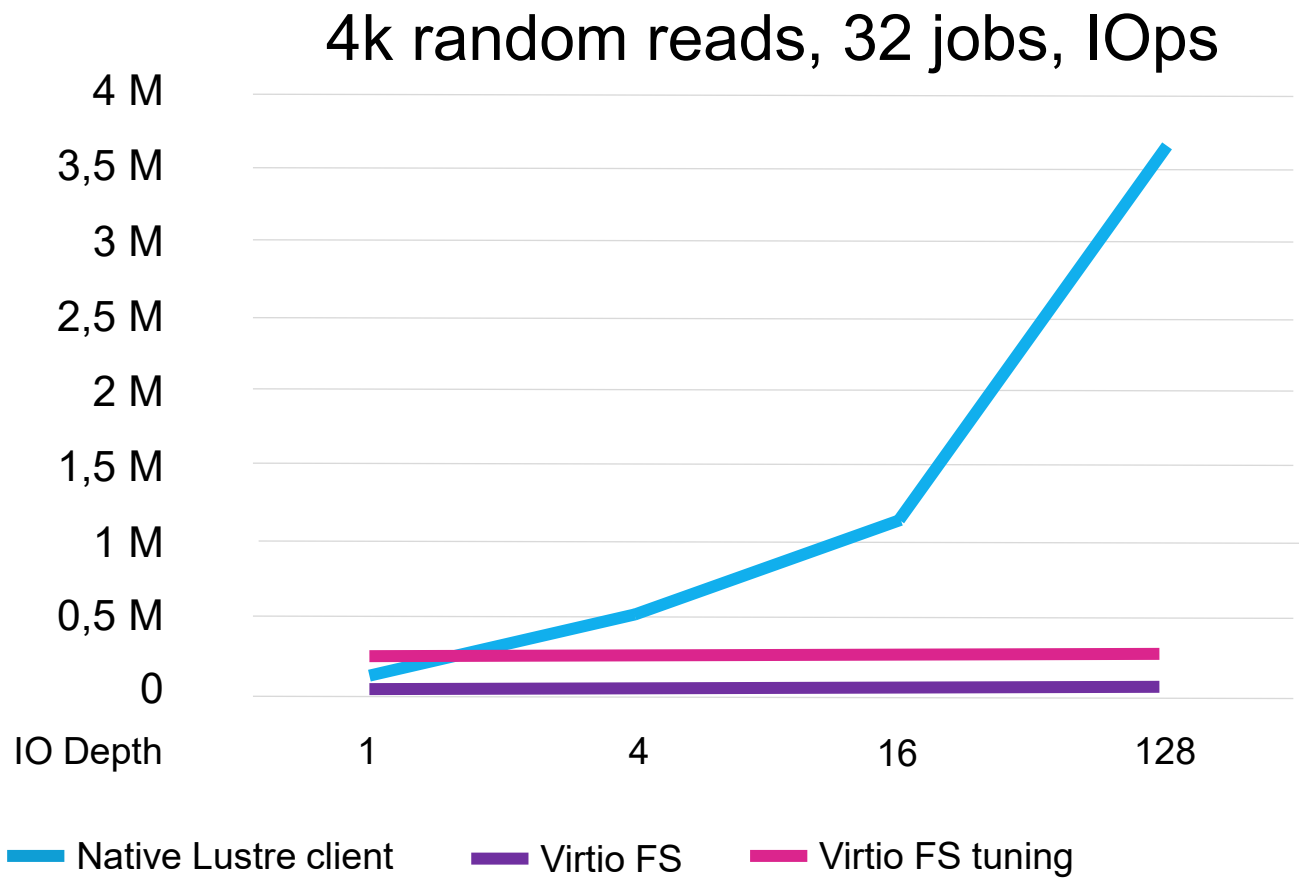


virtio-fs: sharing a directory tree between host and VMs

virtiofsd: vhost-user device daemon written in Rust

We can hide all the complexity of parallel file system setup from the client behind VIRTIOFS!

Xinnor Tuned Virtio FS = performance results



Sequential read 1M, 32 jobs:

- Native Lustre client: 44 GB/s
- Virtio FS: 9 GB/s
- Tuned Virtio FS: 44 GB/s

Sequential write 1M, 32 jobs:

- Native Lustre client: 43 GB/s
- Virtio FS: 7 GB/s
- Tuned Virtio FS: 44 GB/s

Outcomes = Xinnor Lustre Solution can perform

1. Performance:

- Even with only two virtualized OSS Lustre delivers strong results for sequential and random I/O operations (AIO).
- It is essential to have high-performance block devices passed through to the OSS and MDS virtual machines.
- **xiRAID Opus solves this challenge.**

2. Skill requirements:

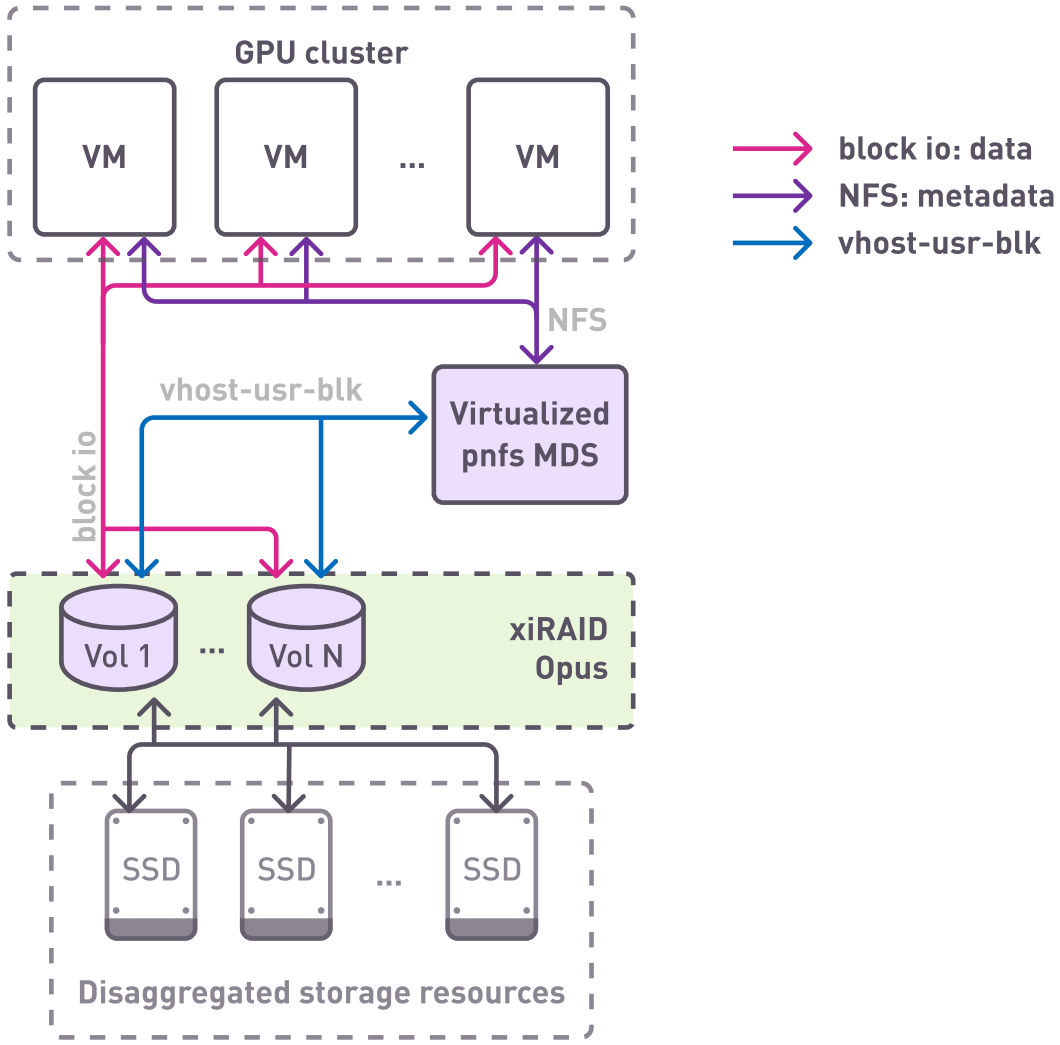
- Requires a high level of expertise to configure the system and client VMs.
- VirtioFS can reduce complexity:
  - For use-cases that require only sequential workload patterns.

3. Xinnor can deliver Lustre solution for Cloud Environments.

Reach out to Xinnor for a POC.

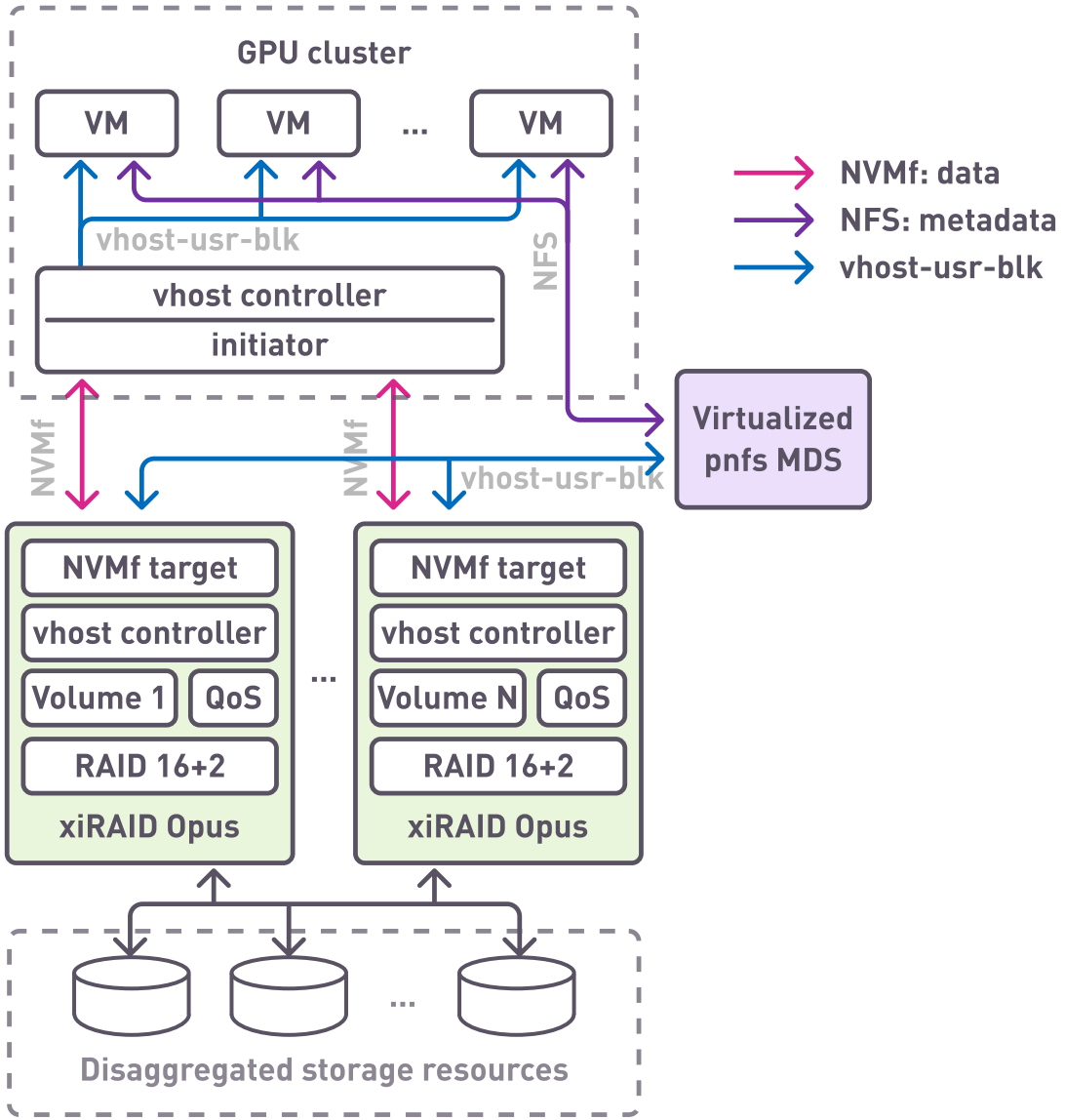
Our vision of the future: pNFS

pNFS Block Layout



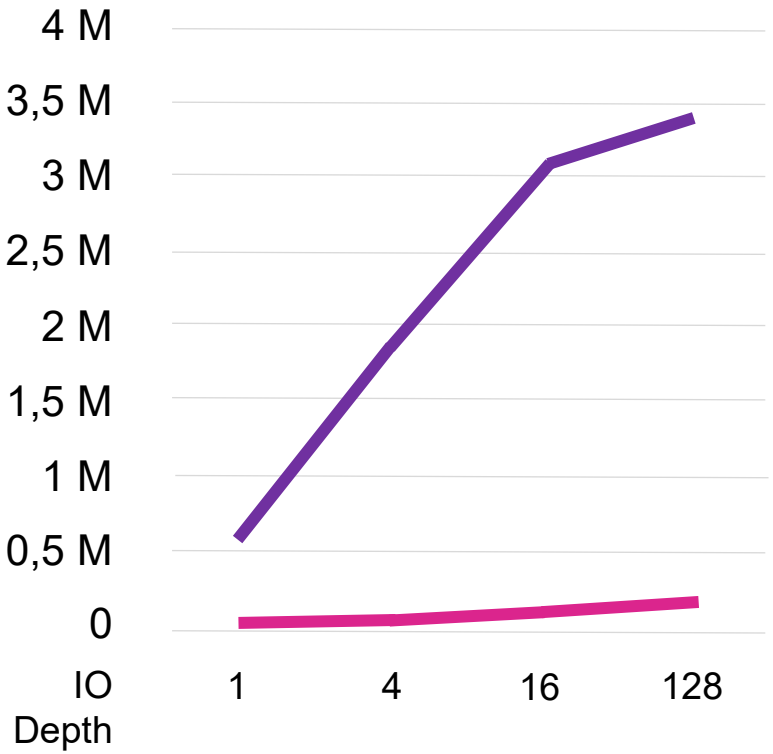
- Allows to build high-performance and scalable data volumes
- Clients reads and writes directly to volumes
- Data and metadata paths are separated
- **No need for third-party client Software**

pNFS Architecture in Cloud Environments

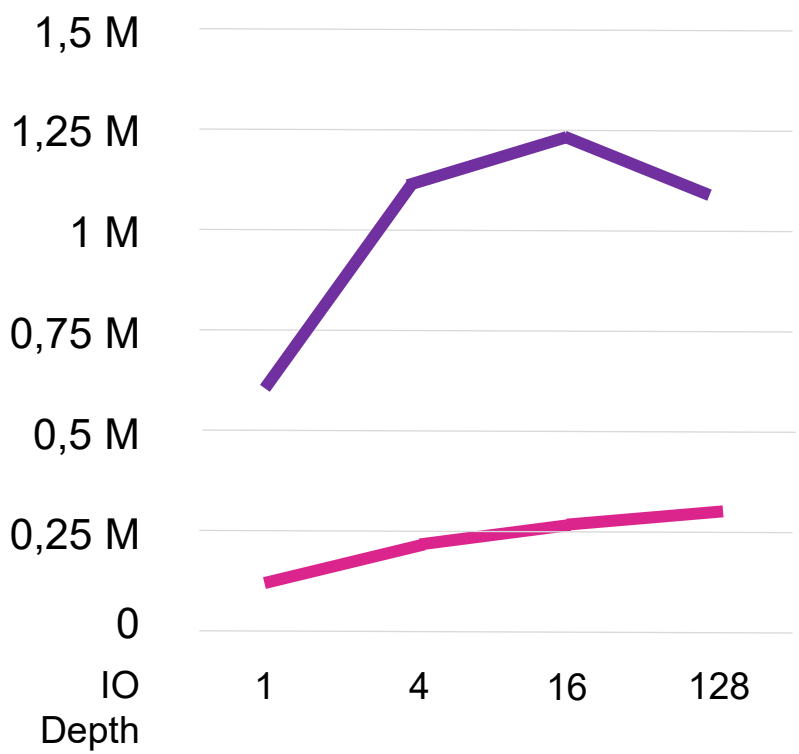


pNFS with and without xiRAID Opus

Random read, 32 jobs, IOps



Random write, 32 jobs, IOps



pNFS w/o xiRAID Opus

pNFS w/ xiRAID Opus

Sequential read 1M, 32 jobs:

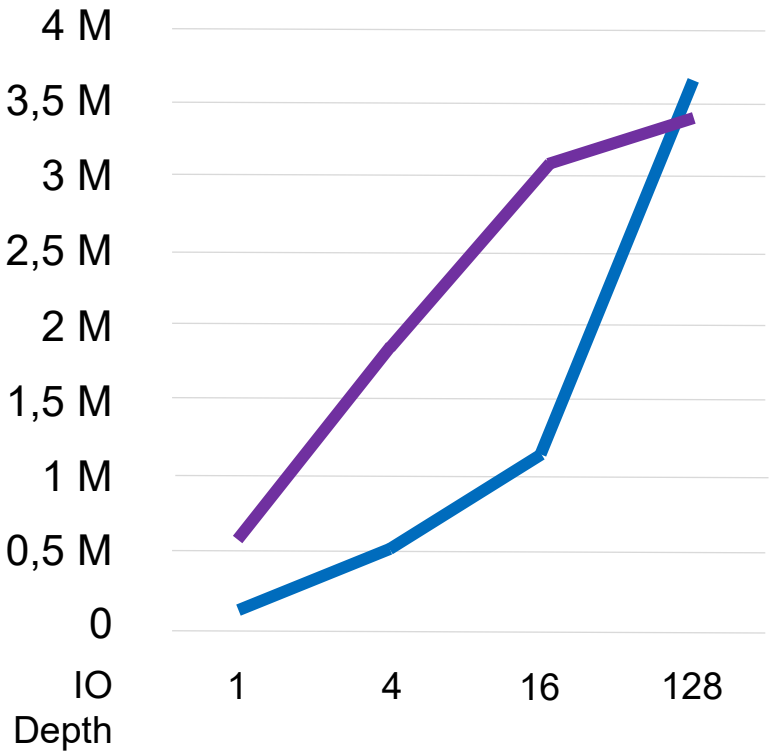
- without xiRAID Opus: 34,8 GB/s
- with xiRAID Opus : 47 GB/s

Sequential write 1M, 32 jobs:

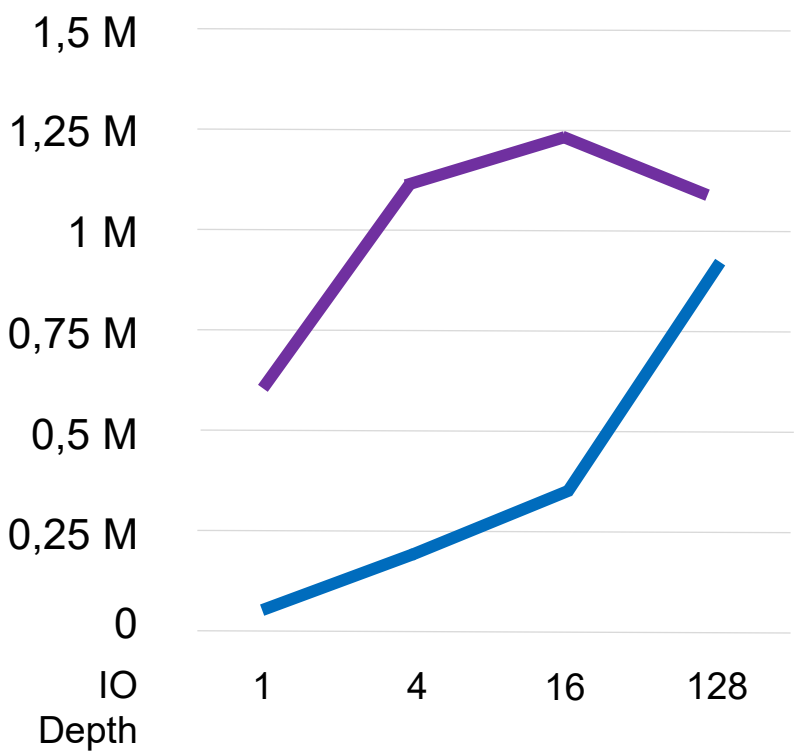
- without xiRAID Opus: 32,7 GB/s
- with xiRAID Opus: 46 GB/s

pNFS vs Lustre - accelerated by Xinnor solutions

Random read, 32 jobs, IOps



Random write, 32 jobs, IOps



Lustre w/ xiRAID Opus

pNFS w/ xiRAID Opus

Sequential read 1M, 32 jobs:

- Lustre: 44 GB/s
- pNFS: 47 GB/s

Sequential write 1M, 32 jobs:

- Lustre: 43 GB/s
- pNFS: 46 GB/s

pNFS in Cloud Environments - Conclusions

Best Option with Proper Configuration

- Supports scalability to tens or even hundreds of gigabytes per second with minimal resource usage.

High Performance

- Sequential and random Small Block operations, with minimal latency due to direct interaction with the block device.

No 3rd party client software required

Drawbacks:

- Open Source MDS is not production-ready and can only be used for POCs.

We are looking for partners to make pNFS production ready.

Conclusions

Xinnor has solutions for AI in Cloud Environments

xiRAID Opus and Xinnor Lustre Solution are both ready to be deployed in Cloud environments as a high-performance solutions for AI workloads.

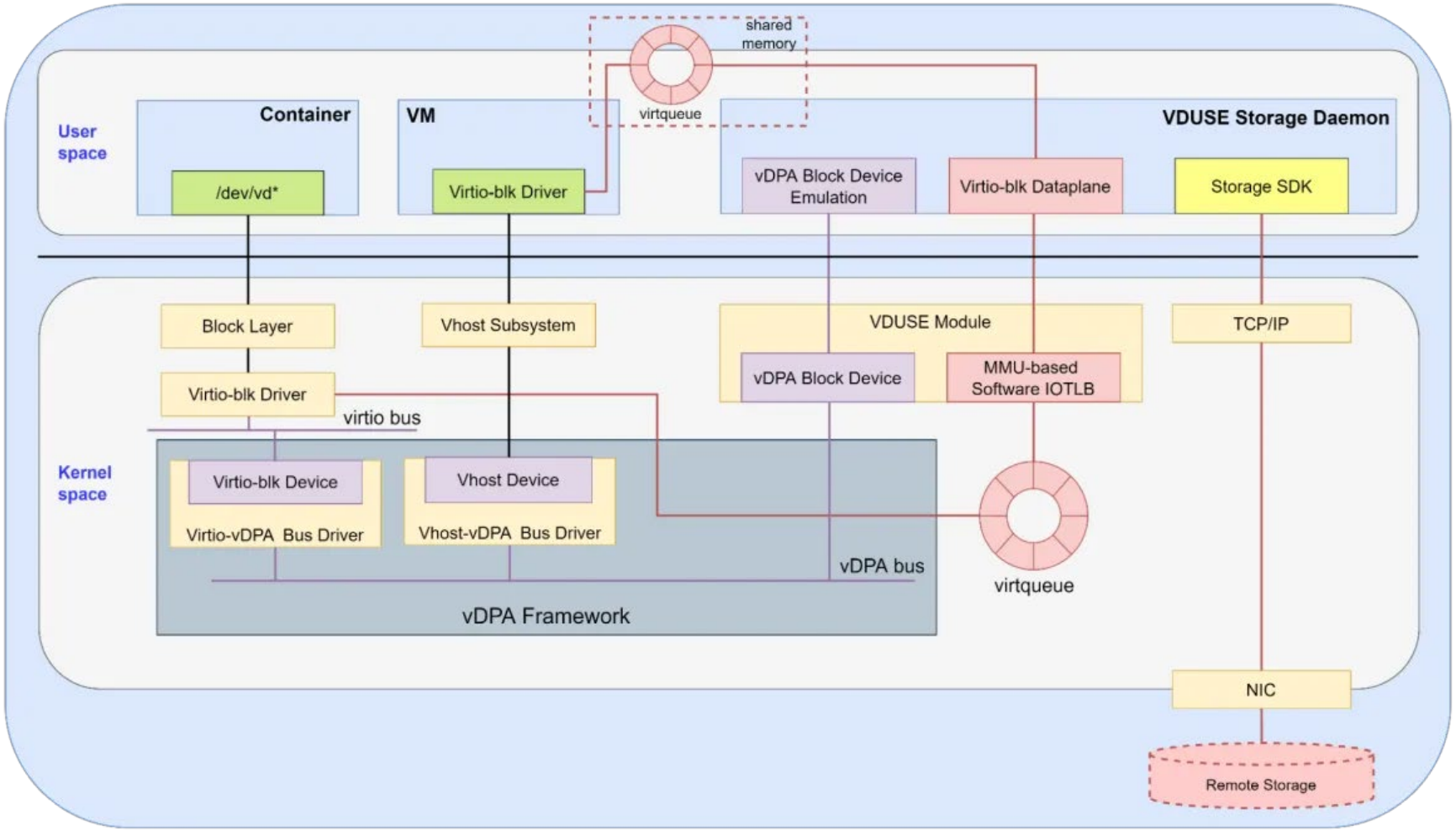
- Lustre Enables High-Intensity AI Workloads
  - New versions excel in Asynchronous Small Block I/O performance

- The VHOST-User-BLK interface, especially with Multi-IO Thread support, allows the direct passthrough of high-performance block volumes into virtual machines.

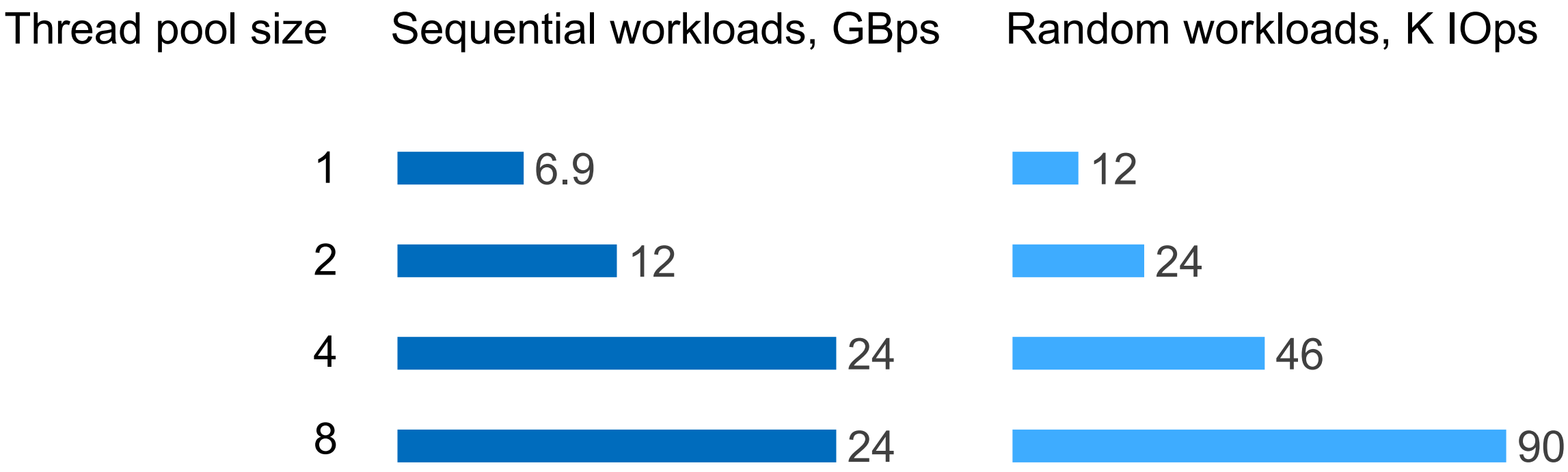
Backup

Backup

Backup



VirtioFS requires additional CPU cores for each VM



Outcomes of VirtioFS addition to Lustre

Simplified Configuration:

- VirtioFS eliminates the need for client-side settings within virtual machines.

Performance Considerations:

- Tuning of VirtioFS on the host side is required.
- Dedicating extra CPU cores that are isolated from the ones used by the virtual machines is necessary.
- Increased cost of data access.

Random I/O Performance:

- Poor small random I/O performance.

VirtIO FS is suitable for workloads that handle large volumes of data, where frequent Small IO operations are not required.

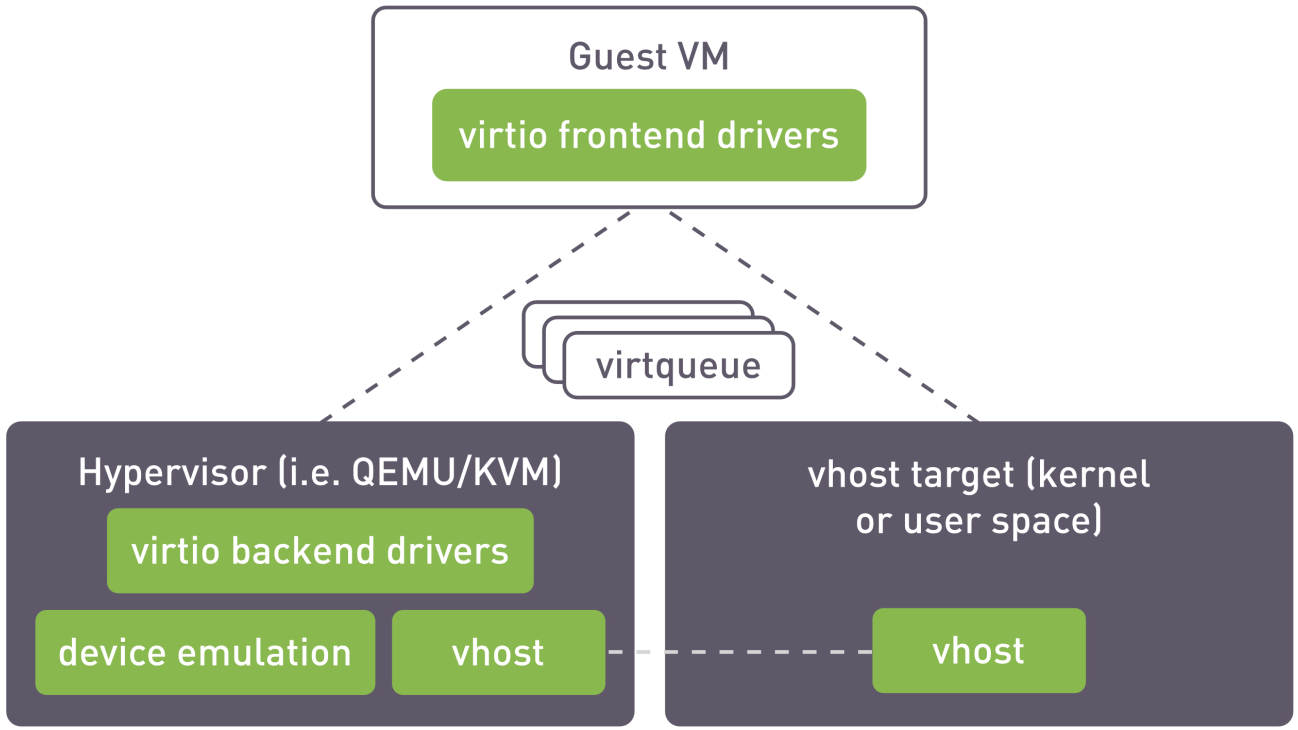
However, to achieve high performance, properly tuned Lustre OSS/OST and a high-performance backend are essential to efficiently handle the workload.

vhost-usr-blk

- Local block I/O interface
- Userpace
- Zero-copy (shared memory)
- Notifications and polling

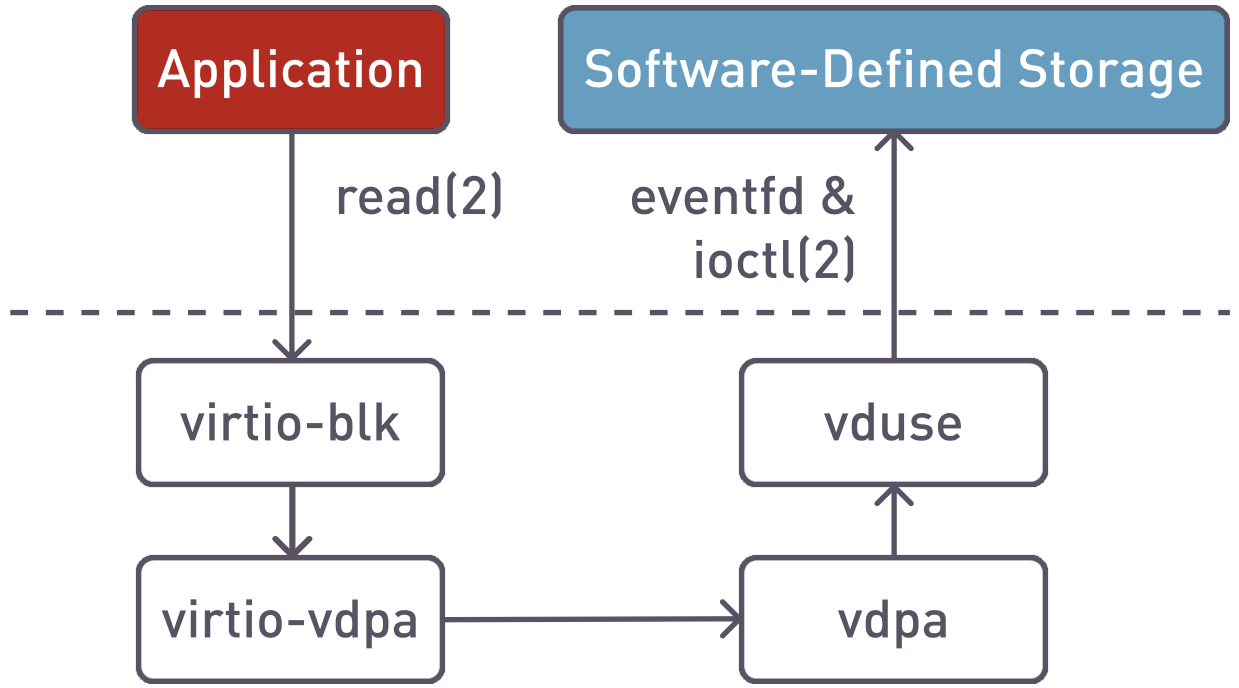
Linux, BSD, and macOS

Implementations started in 2017



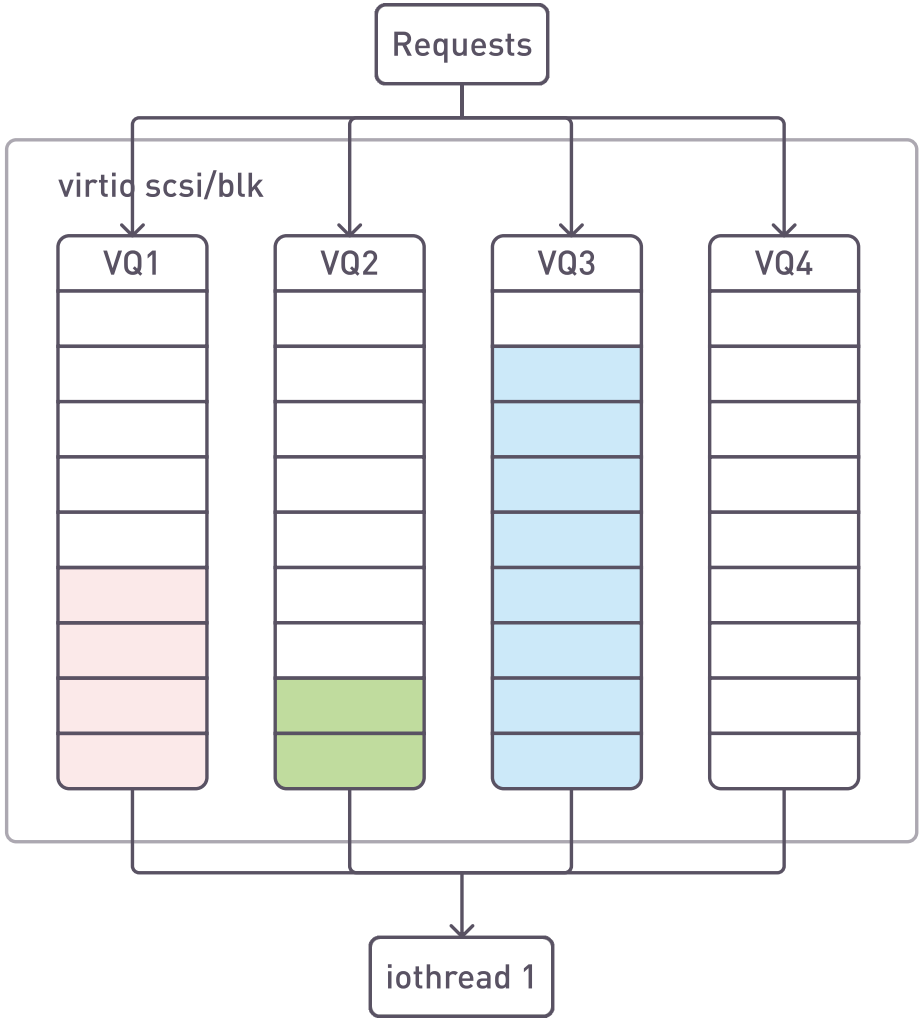
VDUSE: a software-defined datapath for virtio

- vDPA device in userspace (VDUSE) is an emerging approach for providing software-defined storage and networking services to virtual machine (VM) and container workloads.
- The vDPA (virtio data path acceleration) kernel subsystem is the engine behind VDUSE.

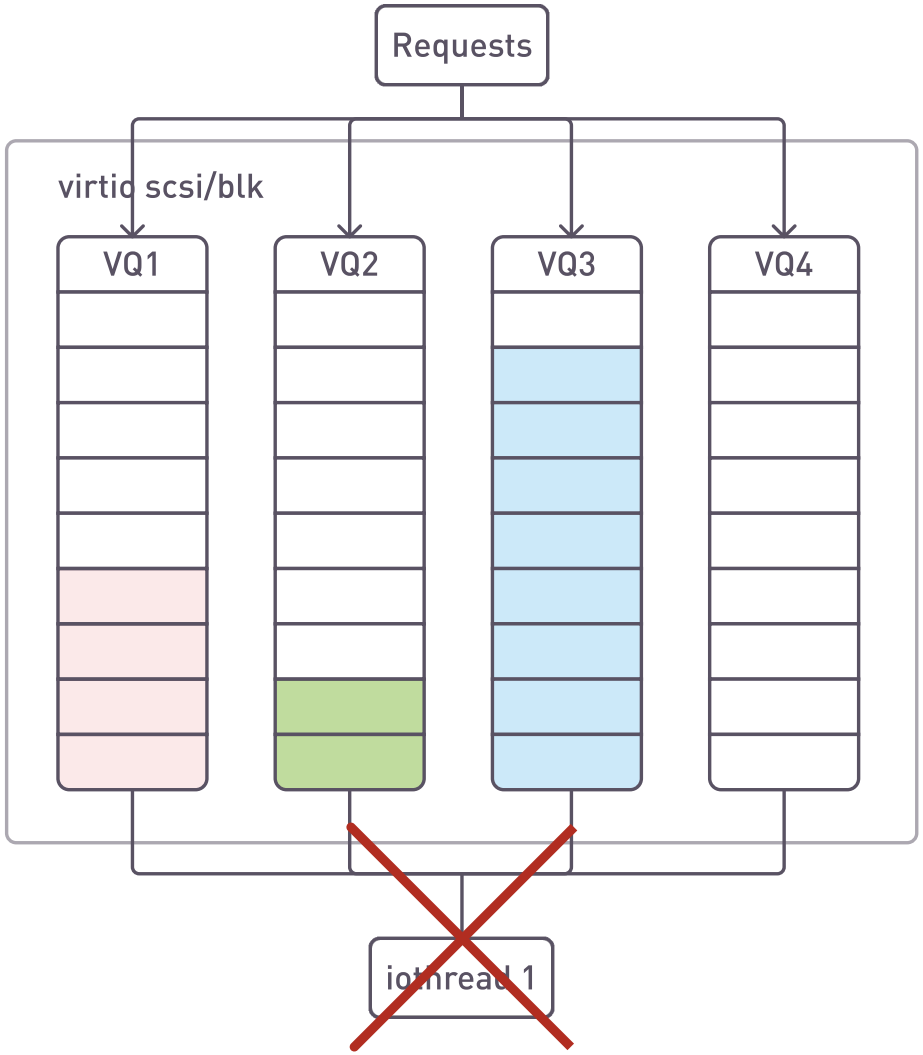


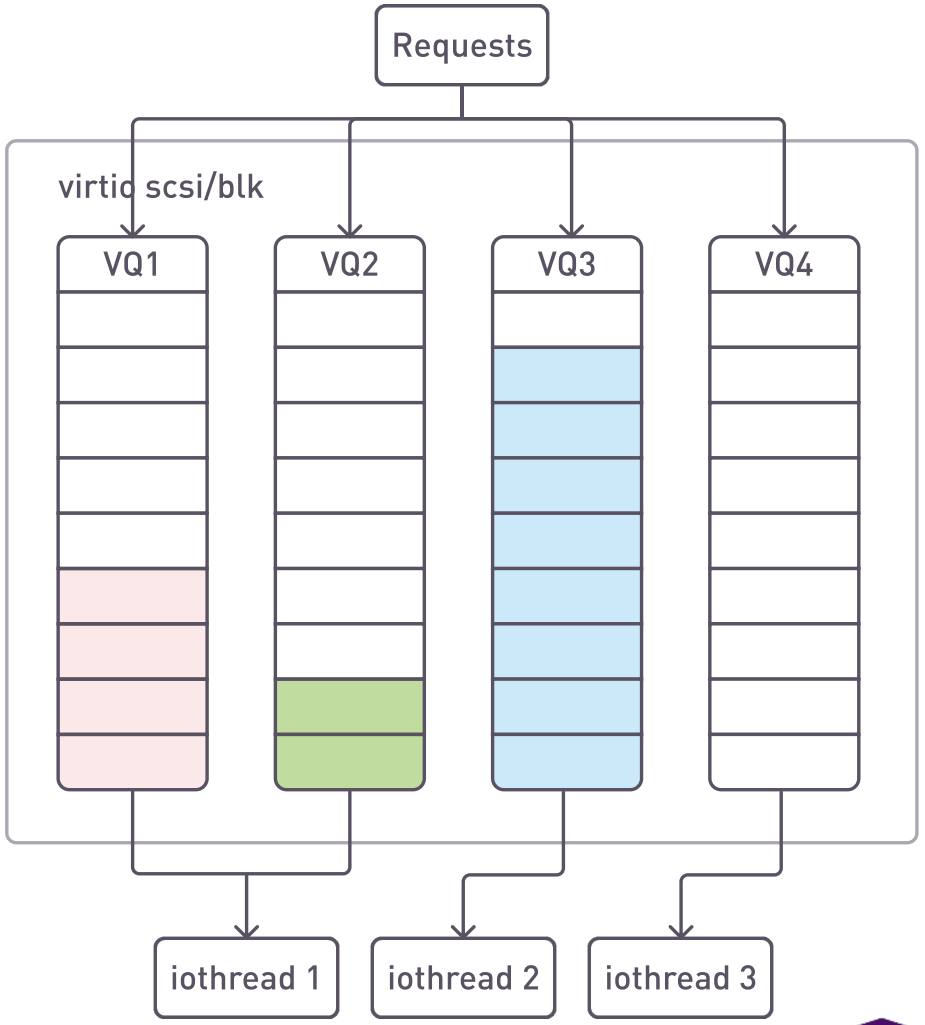
VDUSE enables you to easily implement a software-emulated vDPA device in userspace to serve both VM and container workloads.

VirtIO - True multi-threading in QEMU 9

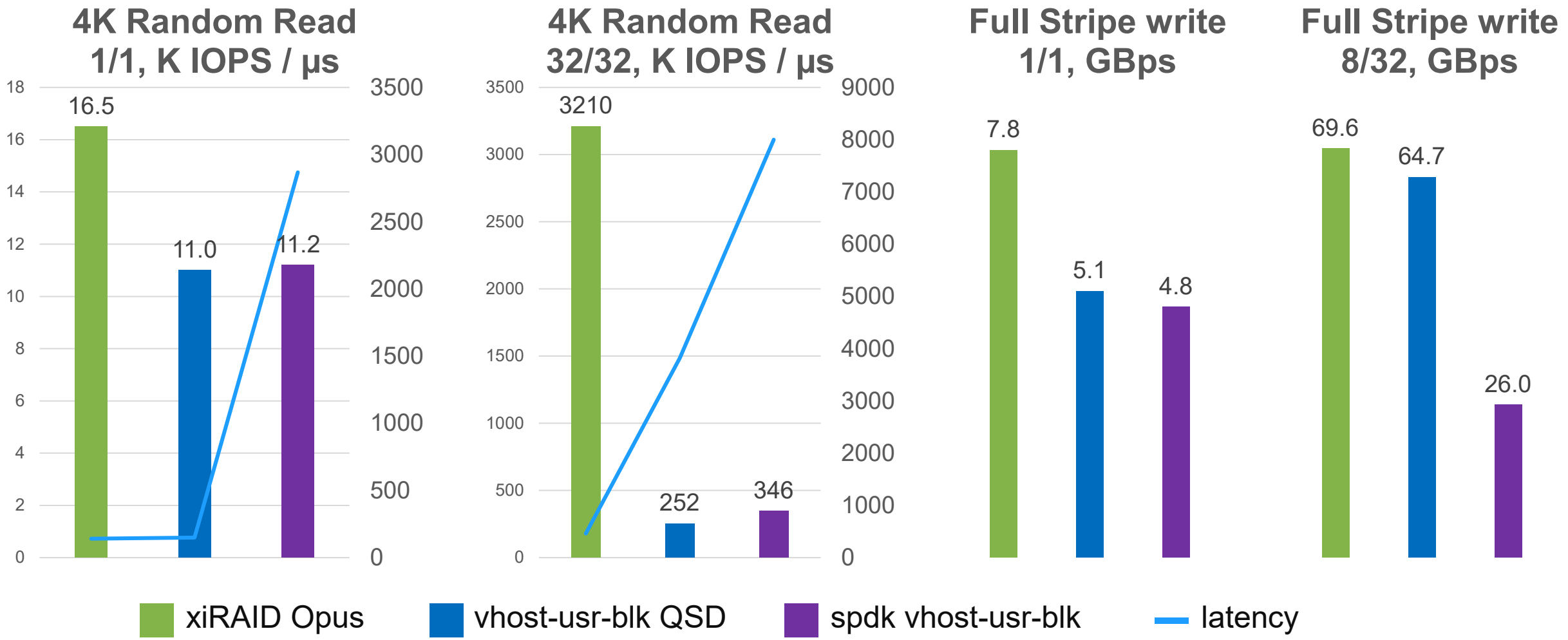


VirtIO - True multi-threading in QEMU 9





Vhost-usr-blk Opus vs SPDK vs QSD implemetations



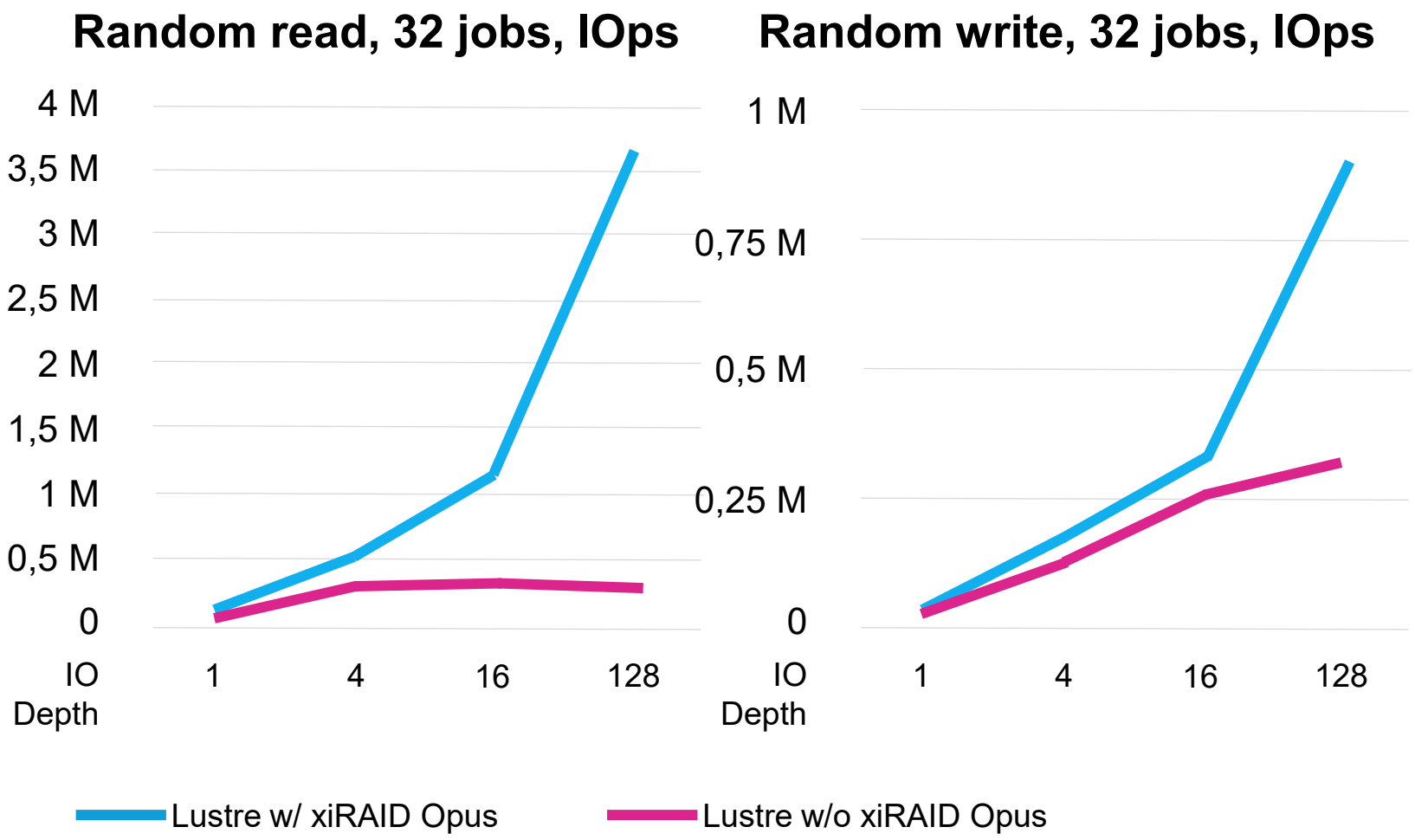
VIRTIO vs IO_URING PT

Workload	VIRTIO, MIOT, AIO=native		Io_uring PT	
Random read 1J/1IOD	12.4k IOps	147 µs 99.9 lat	12.7k IOps	133 µs 99.9 lat
Random read 32J/32IOD	870k IOps	2681 µs 99.9 lat	920k IOps	1647 µs 99.9 lat

vDUSE, mdraid vs sigle drive vfio_pci

Workload	VDUSE 1 drive, vfio_pci		VDUSE, mdraid 0	
Random read 1J/1IOD	10.2k IOps	145 µs 99.9 lat	9.1k IOps	165 µs 99.9 lat
Random read 32J/32IOD	70k IOps	435 µs 99.9 lat	225k IOps	2440 µs 99.9 lat

Lustre Solution Performance



Sequential read 1M, 32 jobs:

- without xiRAID Opus: 44 GB/s
- with xiRAD Opus: 44 GB/s

Sequential write 1M, 32 jobs:

- without xiRAID Opus: 44 GB/s
- with xiRAD Opus: 43 GB/s

These results can be achieved with multithreaded vhost-user-blk only!

fio configurations 1

-name=randtest -bs=4k –ioengine=libaio –direct=1 –iodepth={1,32} –numjobs={1,32} –norandommap –filename=/dev/vda –group_reporting -rw={randread,randwrite} -timebased=1 –runtime=600

-name=randtest_iou -bs=4k –ioengine=io_uring –fixedbufs=1 –hipri=1 –registerfiles=1 –direct=1 –iodepth={1,32} –numjobs={1,32} –norandommap –filename=/dev/vda –group_reporting -rw={randread,randwrite} -timebased=1 –runtime=600

-name=seqRW -bs={1024k, 1472k, 1536k} –ioengine=libaio –direct=1 –iodepth={1,32} –numjobs={1,8} –offset_increment=10% –filename=/dev/vda –group_reporting -rw={read,write} -timebased=1 –runtime=600

fio configurations 2

- -name=randtest -bs=4k –ioengine=libaio –direct=1 –iodepth={1, 8, 16, 128} –numjobs={16} –norandommap -directory=/mount –group_reporting -rw={randread,randwrite} -timebased=1 –runtime=600 –size=50G
- -name=seqRW -bs={1024k, 1472k, 1536k} –ioengine=libaio –direct=1 –iodepth={1,32} –numjobs={1,8} -directory=/mount -group_reporting -rw={read,write} -timebased=1 –runtime=600 –size=50G

QSD settings

```
taskset -ac 32-63  qemu-storage-daemon  --blockdev driver=raw,node-name=md0,file.driver=host_device,file.filename=/dev/md0,cache.direct=on  --export type=vduse-blk,id=md0-export,node-name=md0,writable=on,name=vduse-0,num-queues=32,queue-size=256
```

```
taskset -ac 32-63  qemu-storage-daemon --blockdev driver=nvme,node-name=md0,driver=nvme,device=0000:01:00.0,namespace=2 --export type=vduse-blk,id=md0-export,node-name=md0,writable=on,name=vduse-0,num-queues=32,queue-size=256
```


virtiofs run parameters

```
taskset -ac 32-63 /usr/libexec/virtiofsd --socket-path=/tmp/vhostqemu.sock -o source=/virtiofs/ --cache=metadata --allow-direct-io --thread-pool-size=16 &
```


VM run parameters 1

```
taskset -ca 0-31  /usr/libexec/qemu-kvm  \
-enable-kvm -cpu host\
-m 32G -object memory-backend-file,id=mem,size=32G,mem-path=/dev/shm,share=on -numa node,memdev=mem \
-chardev socket,id=char0,path=/tmp/vhostqemu.sock2 -device vhost-user-fs-pci,queue-size=1024,chardev=char0,tag=myfs \
-smp 32 \
-hda ${VMDISK_QCOW2} \
-netdev user,id=net0,net=192.168.0.0/24,dhcpstart=192.168.0.9 \
-device virtio-net-pci,netdev=net0  -vnc 0.0.0.0:2 --nographic
```


VM run parameters 2

taskset -a -c 0-31 /usr/local/bin/qemu-system-x86_64 \

-enable-kvm -cpu host \

-m 32G -object memory-backend-file,id=mem,size=32G,mem-path=/dev/shm,share=on -numa node,memdev=mem \

-smp 32 \

-hda ${VMDISK_QCOW2} -drive if=none,id=drive0,cache=none,aio=native,format=raw,file=/dev/md127 \

-device '{"driver":"virtio-blk-pci","drive":"drive0","iothread-vq-mapping":[{"iothread":"my0","vqs":[0,1]},{"iothread":"my1","vqs":[2,3]},{"iothread":"my2","vqs":[4,5]},{"iothread":"my3","vqs":[6,7]},{"iothread":"my4","vqs":[8,9]},{"iothread":"my5","vqs":[10,11]},{"iothread":"my6","vqs":[12,13]},{"iothread":"my7","vqs":[14,15]},{"iothread":"my8","vqs":[16,17]},{"iothread":"my9","vqs":[18,19]},{"iothread":"my10","vqs":[20,21]},{"iothread":"my11","vqs":[22,23]},{"iothread":"my12","vqs":[24,25]},{"iothread":"my13","vqs":[26,27]},{"iothread":"my14","vqs":[28,29]},{"iothread":"my15","vqs":[30,31]}]}' \

-netdev tap,id=net0,ifname=tap0,script=no,downscript=no -object iothread,id=my0 -object iothread,id=my1 -object iothread,id=my2 -object iothread,id=my3 -object iothread,id=my4 -object iothread,id=my5 -object iothread,id=my6

-object iothread,id=my7 -object iothread,id=my8 -object iothread,id=my9 -object iothread,id=my10 -object iothread,id=my11 -object iothread,id=my12 -object iothread,id=my13 -object iothread,id=my14 -object iothread,id=my15 \-device virtio-net-pci,netdev=net0 -vnc 0.0.0.0:1

VM run parameters 3

```
taskset -ac 0-31 /usr/local/bin/qemu-system-x86_64  \
-enable-kvm -cpu host -m 32G -smp 32 \
-m 32G -object memory-backend-file,id=mem,size=32G,mem-path=/dev/hugepages,share=on -numa node,memdev=mem \
-hda ${VMDISK_QCOW2} \
-vnc 0.0.0.0:2 \
-device vhost-vdpa-device-pci,vhostdev=/dev/vhost-vdpa-0,queue-size=256
```


VM run parameters 4

```
taskset -a -c 0-31 /usr/libexec/qemu-kvm \
-enable-kvm -cpu host\
-m 32G -object memory-backend-file,id=mem,size=32G,mem-path=/dev/hugepages,share=on -numa node,memdev=mem  \
-smp 32 \
-hda ${VMDISK_QCOW2} -chardev socket,id=char1,path=/opt/xiraid/bin/xnr_conf/sock/r1 -device vhost-user-blk-pci,id=blk0,chardev=char1,num-queues=32,queue-size=256 \
-netdev socket,id=net0,connect=127.0.0.1:1234 \
-device virtio-net-pci,netdev=net0 \
-netdev user,id=net1,net=192.168.0.0/24,dhcpstart=192.168.0.9 \
-device virtio-net-pci,netdev=net1 \
 -vnc 0.0.0.0:4 --nographic
```

