Storage Benchmarking

Repeatable & Comparable

Trent Lloyd
Sustaining Engineering @ Canonical (Ubuntu)
@lathiat@fosstodon.org

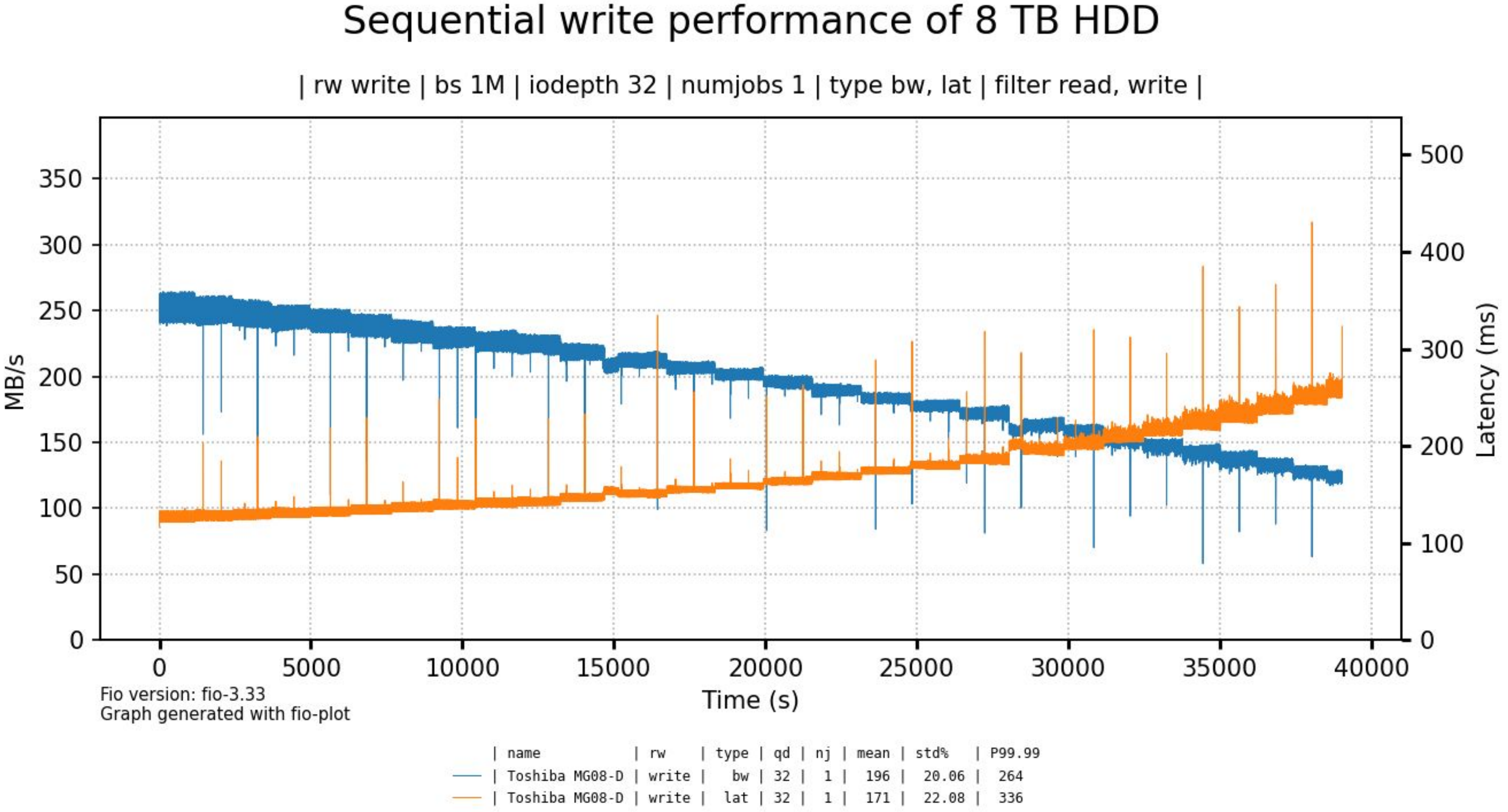
https://fosstodon.org/@lathiat

Feedback/Errata

Summary of Feedback received after the presentation, for those who watch the video or saw the original talk.

- HDDs - Performance can be significantly affected by vibrations, particularly with many drives in a chassis
  - Use rubber mounts if provided, ensure screws are tight, etc.
  - Different HDDs may also handle this better, e.g. "NAS" rated drives versus desktop drives.
  - Video: Shouting in the Datacenter
- Reading from sparse/trimmed storage impossibly fast can help show max bandwidth of the storage path
  - Particularly for SAS/Fibre Channel
  - Not true for Ceph, as the RBD client generates the bytes on the hypervisor itself
  - Zoned / Shingled Magnetic Recording (SMR) HDDs also do this, not just Virtual Storage/SSDs
- Secure Erase on your NVMe drive can be useful
  - **Warning: Erases all data from the drive, absolutely irreversibly. Perform with caution.** Use nvme-cli.
  - Generally clears all of the internal wear levelling and block indirection metadata/state.
  - Helps reset performance to a simple state, has been known and seen by me to sometimes fix "bugs" / resolve fragmentation causing particularly bad performance. Good to do before starting each round of benchmarks.
- Hardware Bottlenecks Diagram
  - Add external FC/SAS enclosures and their link bandwidth to the Hardware Bottlenecks diagram
  - Pinning FIO to one CPU socket, same or different to the storage PCIe links may assist with testing the inter-socket bandwidth. Beware real workloads will add memory bandwidth to this link. Location of NIC can also impact network workloads.
- Add details on best practice configurations for FIO. A couple quick ones:
  - Biggest Note: Always use "--direct" for O_DIRECT/Direct IO when benchmarking devices, avoids host caching
    - However benchmarking without this flag can also be helpful to understand filesystem performance for non-O_DIRECT applications.
  - Test with a variety of block sizes, 4K tests are good for "IOPS" but will need larger sizes for max bandwidth.

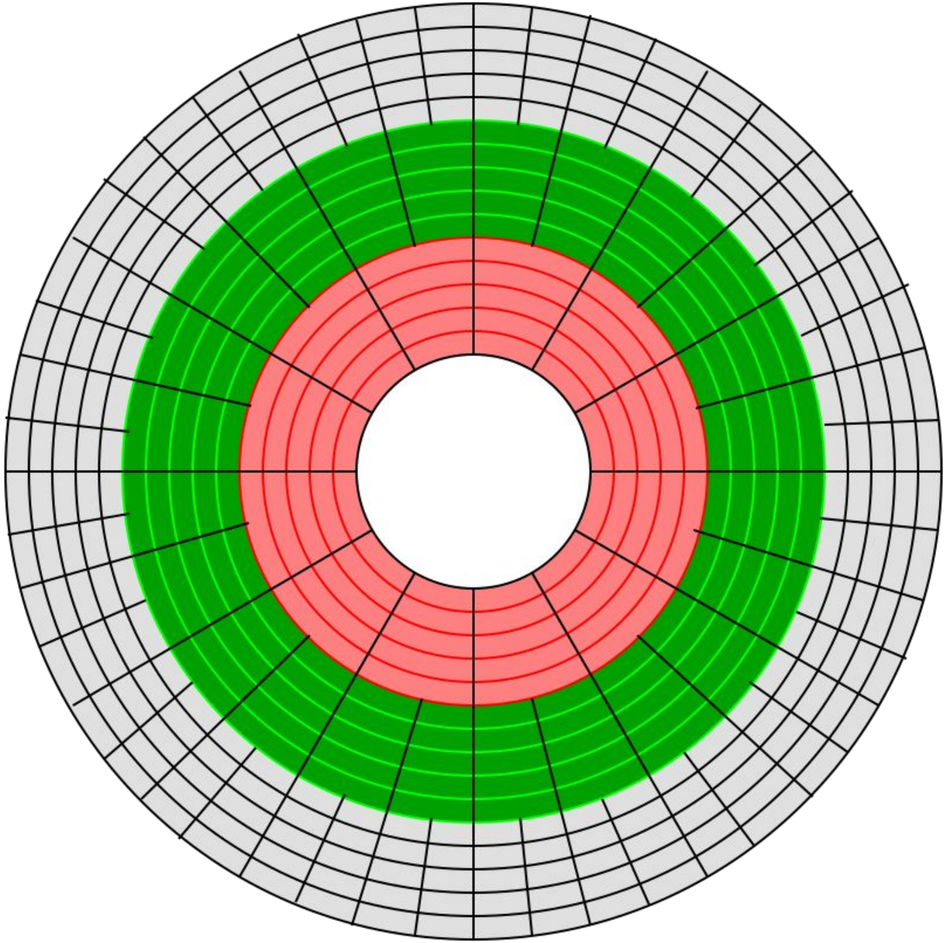
Repeatable & Comparable


Sequential write performance of 8 TB HDD
| rw write | bs 1M | iodepth 32 | numjobs 1 | type bw, lat | filter read, write |
MB/s
Latency (ms)
Time (s)
0
50
100
150
200
250
300
350
0
100
200
300
400
500
0
5000
10000
15000
20000
25000
30000
35000
40000
Fio version: fio-3.33
Graph generated with fio-plot
| name | rw | type | qd | nj | mean | std% | P99.99
| Toshiba MG08-D | write | bw | 32 | 1 | 196 | 20.06 | 264
| Toshiba MG08-D | write | lat | 32 | 1 | 171 | 22.08 | 336

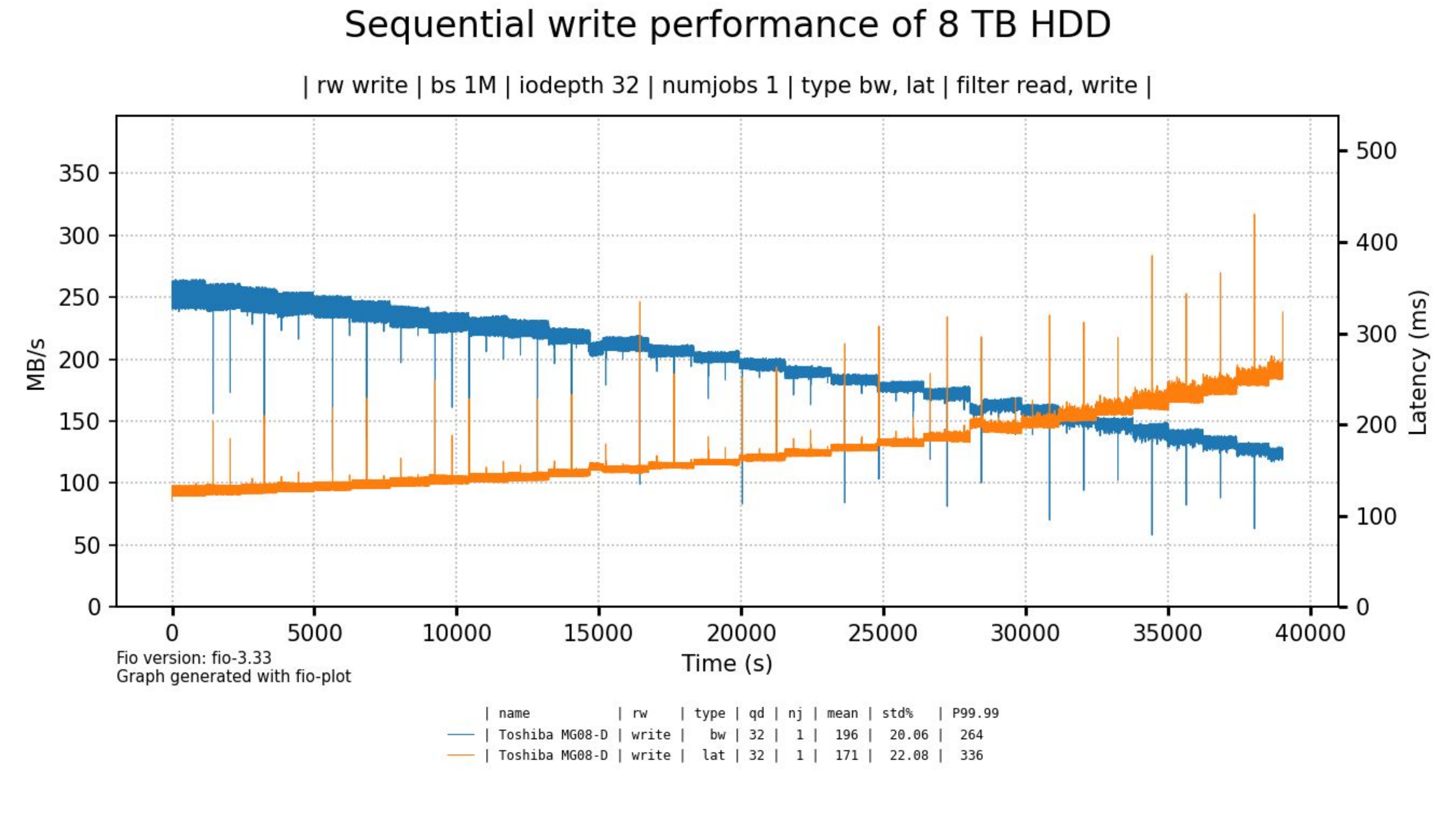
YouTube @bigbluebananabread https://www.youtube.com/shorts/mljraTiKYNY CC0

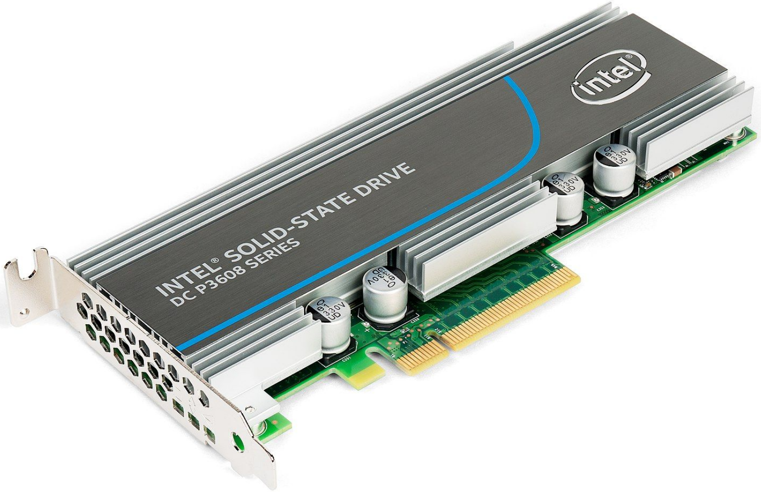
https://commons.wikimedia.org/wiki/File:Laserdisc_CAV.jpg CC-BY-SA-3.0

https://www.reddit.com/r/LaserDisc/comments/jrewab/i_like_how_you_can_see_the_skewed_vertical/

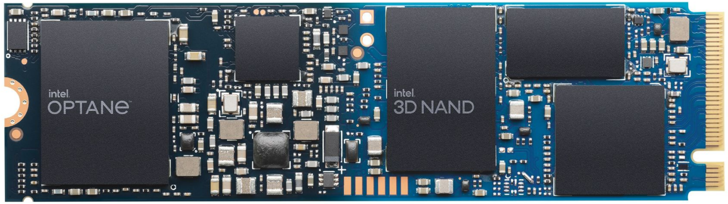


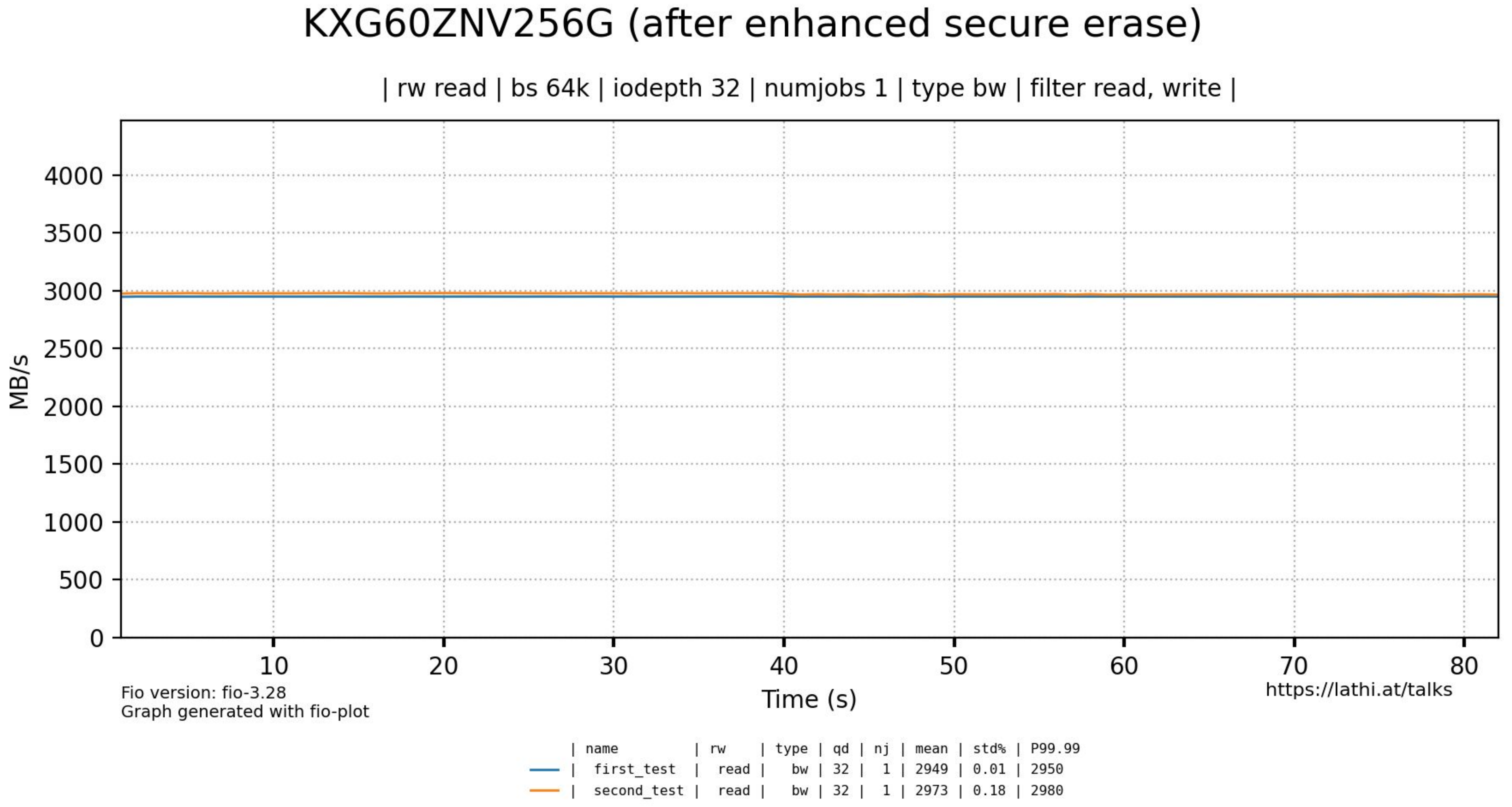
https://en.wikipedia.org/wiki/File:DiskStructure.svg

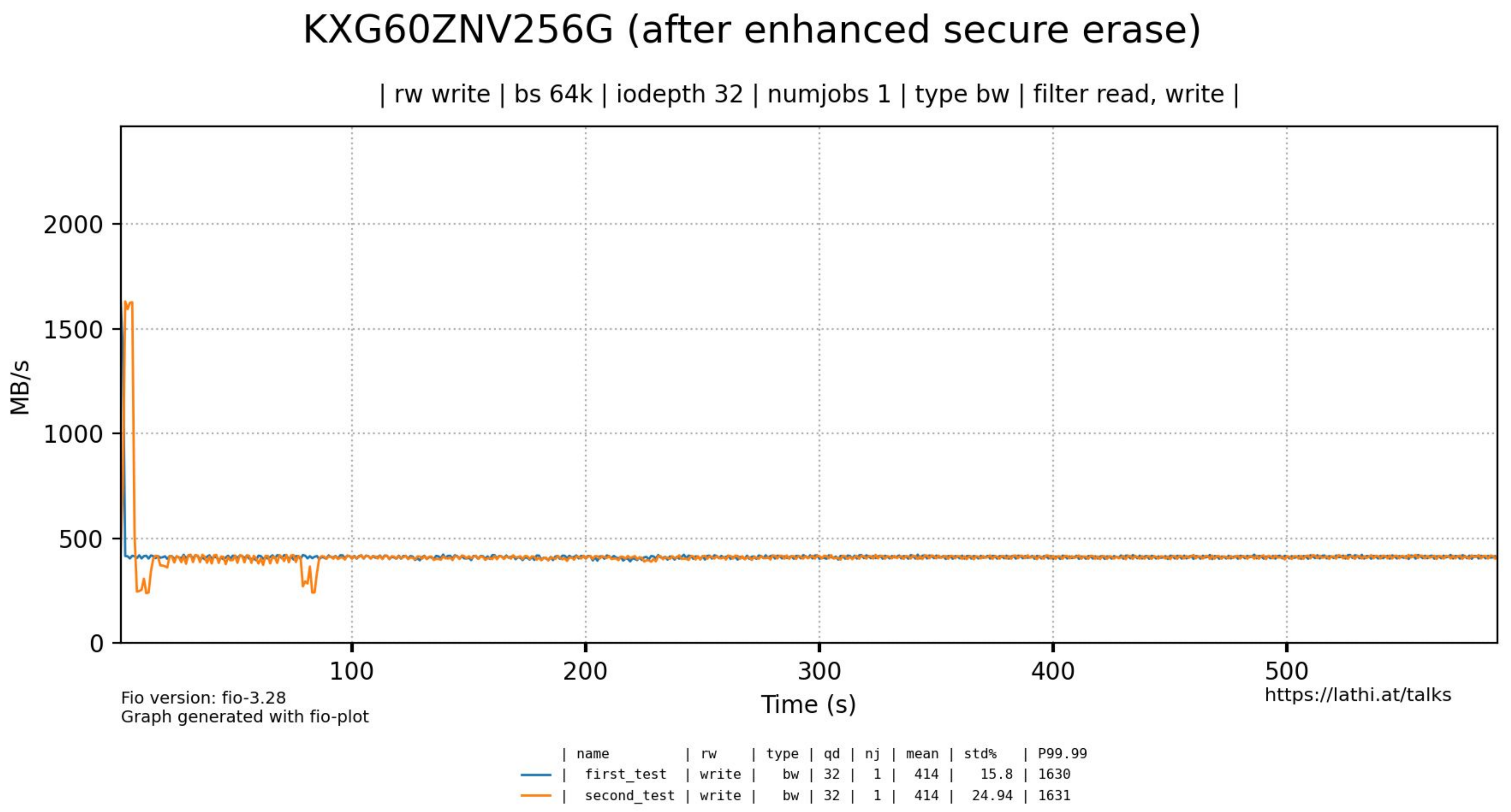

Sequential write performance of 8 TB HDD
| rw write | bs 1M | iodepth 32 | numjobs 1 | type bw, lat | filter read, write |
MB/s
Latency (ms)
Time (s)
0
50
100
150
200
250
300
350
400
500
5000
10000
15000
20000
25000
30000
35000
40000
Fio version: fio-3.33
Graph generated with fio-plot
| name | rw | type | qd | nj | mean | std% | P99.99
| Toshiba MG08-D | write | bw | 32 | 1 | 196 | 20.06 | 264
| Toshiba MG08-D | write | lat | 32 | 1 | 171 | 22.08 | 336


intel
INTEL® SOLID-STATE DRIVE
DC P3608 SERIES

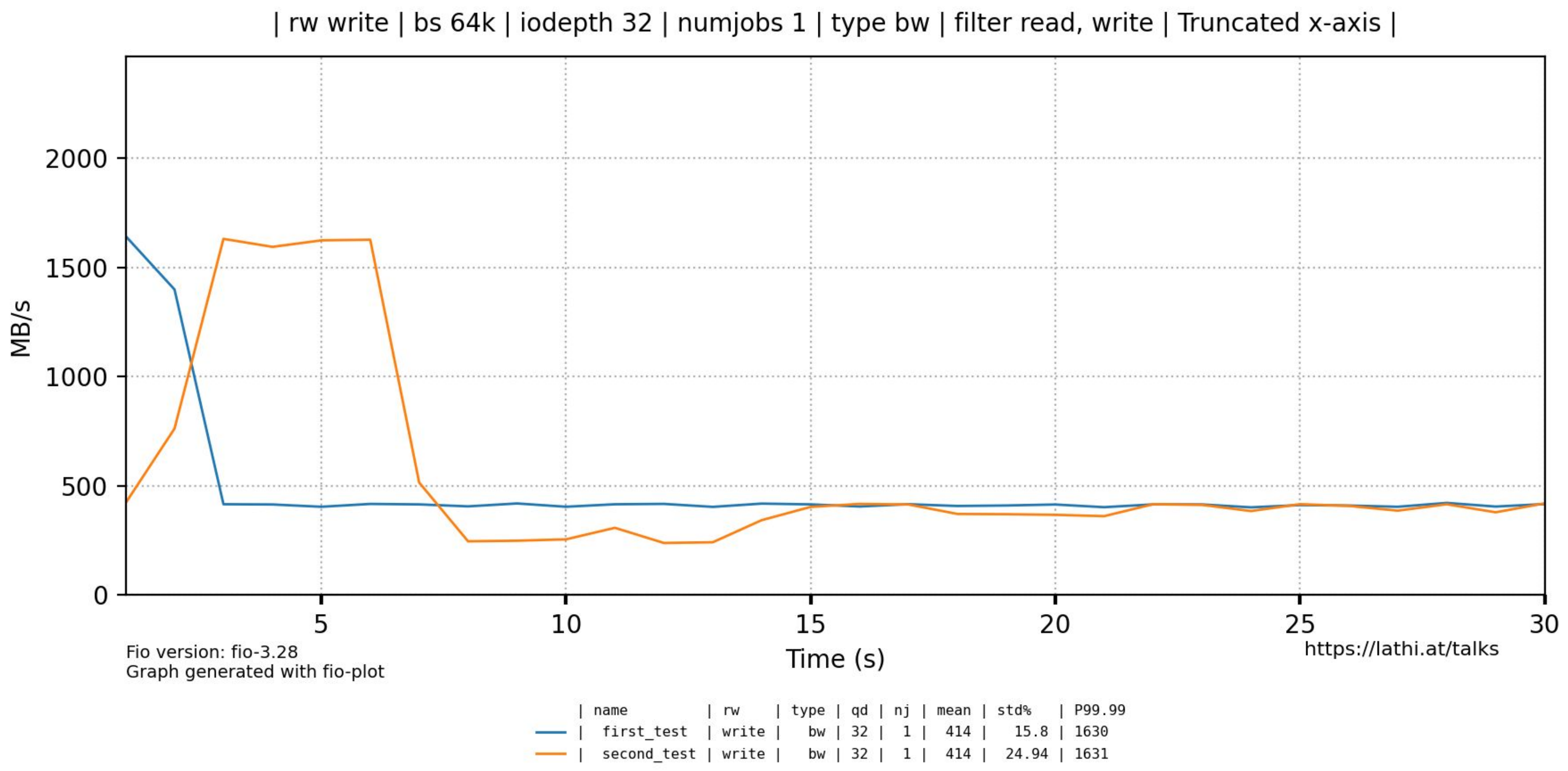

SAMSUNG
PM1653


intel
OPTANE
intel
3D NAND


KXG60ZNV256G (after enhanced secure erase)
| rw read | bs 64k | iodepth 32 | numjobs 1 | type bw | filter read, write |
MB/s
0
500
1000
1500
2000
2500
3000
3500
4000
10
20
30
40
50
60
70
80
Time (s)
Fio version: fio-3.28
Graph generated with fio-plot
https://lathi.at/talks
| name | rw | type | qd | nj | mean | std% | P99.99
| first_test | read | bw | 32 | 1 | 2949 | 0.01 | 2950
| second_test | read | bw | 32 | 1 | 2973 | 0.18 | 2980

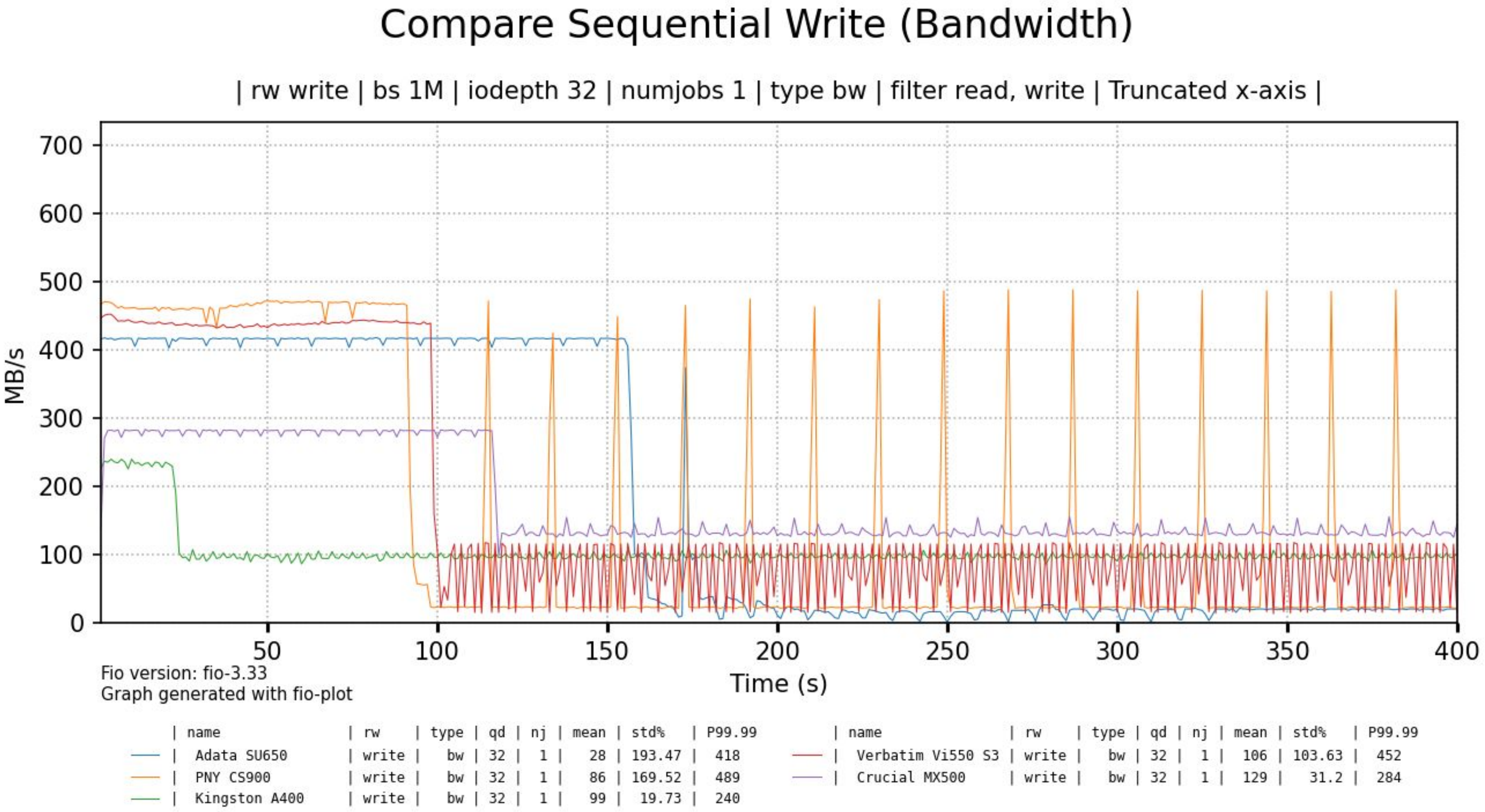

KXG60ZNV256G (after enhanced secure erase)
| rw write | bs 64k | iodepth 32 | numjobs 1 | type bw | filter read, write |
MB/s
0
500
1000
1500
2000
100
200
300
400
500
Time (s)
Fio version: fio-3.28
Graph generated with fio-plot
https://lathi.at/talks
| name | rw | type | qd | nj | mean | std% | P99.99
| first_test | write | bw | 32 | 1 | 414 | 15.8 | 1630
| second_test | write | bw | 32 | 1 | 414 | 24.94 | 1631

KXG60ZNV256G (after enhanced secure erase)



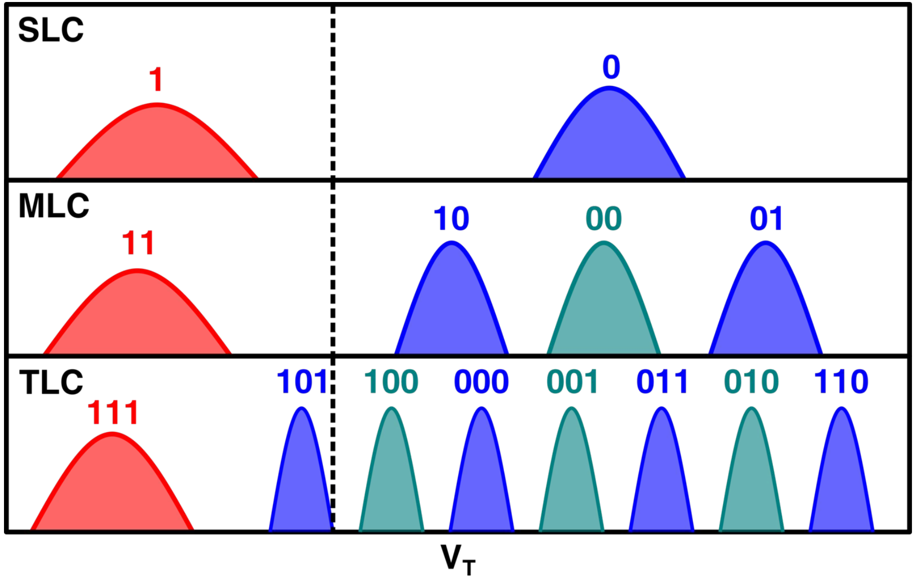
Base Model Number	KXG60ZNV1T02	KXG60ZNV512G	KXG60ZNV256G
SED Model Number	KXG6AZNV1T02	KXG6AZNV512G	KXG6AZNV256G
Capacity	1,024 GB	512 GB	256 GB
Basic Specifications			
Form Factor	M.2 2280-S2 Single-sided		
Interface	PCIe® 3.0, NVMe™ 1.3a		
Maximum Interface Speed	32 GT/s (PCIe® Gen3 x4)		
Flash Memory Type	BiCS FLASH™ TLC		
Performance (Up to)			
Sequential Read	3,180 MB/s	3,100 MB/s	3,050 MB/s
Sequential Write	2,960 MB/s	2,800 MB/s	1,550 MB/s
Random Read	355K IOPS	325K IOPS	270K IOPS
Random Write	365K IOPS	355K IOPS	335K IOPS

Source: https://apac.kioxia.com/en-apac/business/ssd/client-ssd/xg6.html

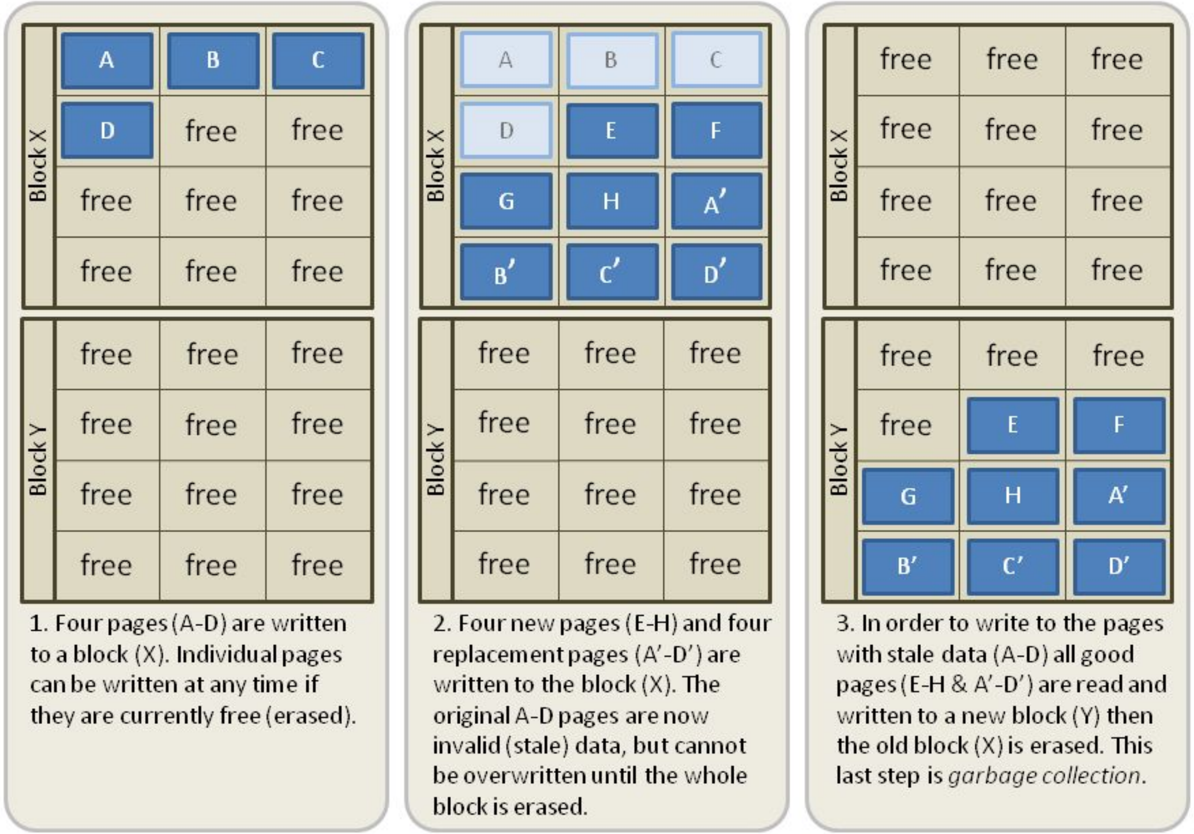

Compare Sequential Write (Bandwidth)
| rw write | bs 1M | iodepth 32 | numjobs 1 | type bw | filter read, write | Truncated x-axis |
MB/s
0
100
200
300
400
500
600
700
50
100
150
200
250
300
350
400
Time (s)
Fio version: fio-3.33
Graph generated with fio-plot
| name | rw | type | qd | nj | mean | std% | P99.99
| Adata SU650 | write | bw | 32 | 1 | 28 | 193.47 | 418
| PNY CS900 | write | bw | 32 | 1 | 86 | 169.52 | 489
| Kingston A400 | write | bw | 32 | 1 | 99 | 19.73 | 240
| name | rw | type | qd | nj | mean | std% | P99.99
| Verbatim Vi550 S3 | write | bw | 32 | 1 | 106 | 103.63 | 452
| Crucial MX500 | write | bw | 32 | 1 | 129 | 31.2 | 284

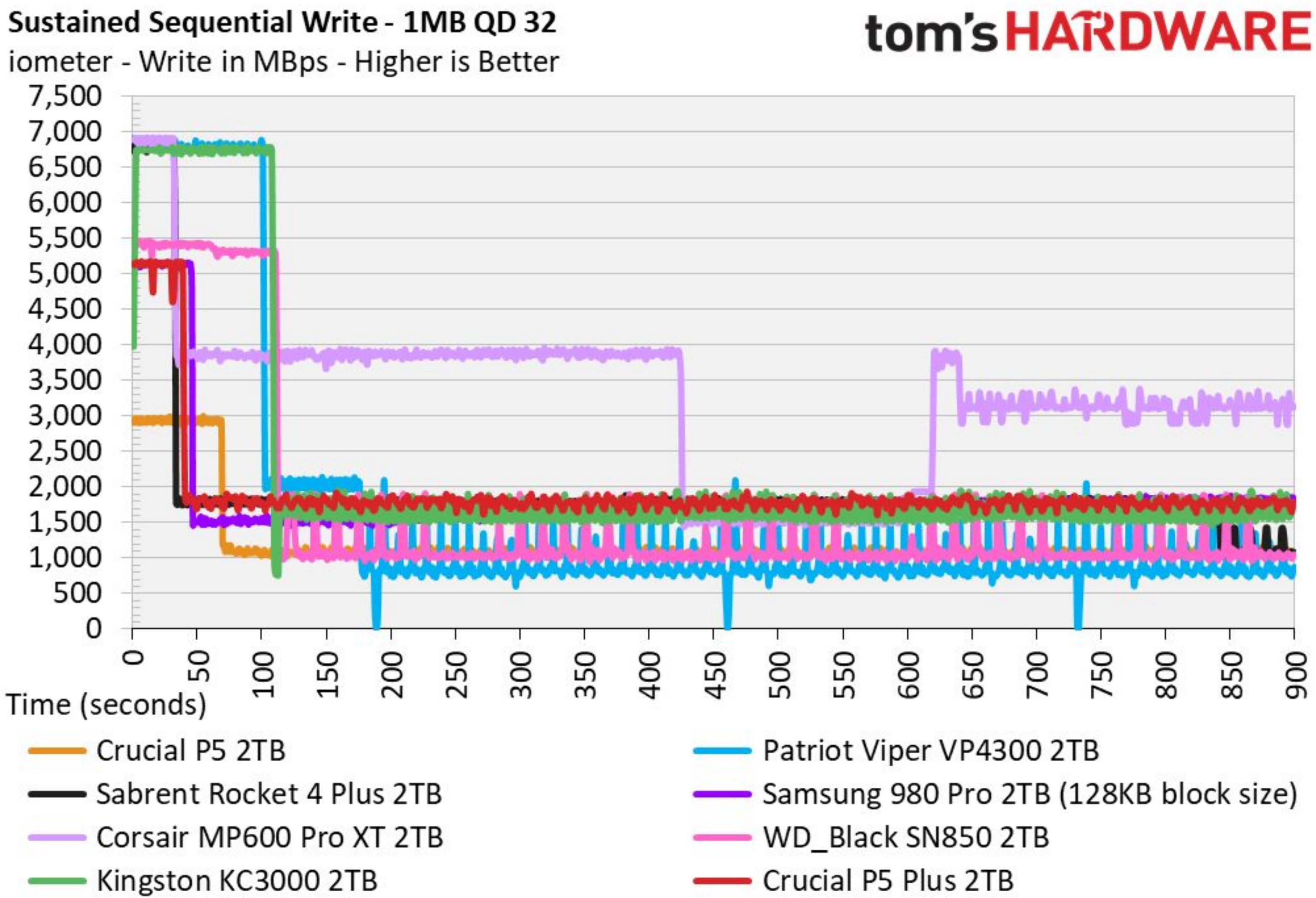
Bits-per-cell

	Bits-per-cell	States	Read Time
SLC	1	1	25us
MLC	2	4	50us
TLC	3	8	100us
MLC	4	16	200us



Garbage Collection





Sustained Sequential Write - 1MB QD 32
iometer - Write in MBps - Higher is Better
tom's HARDWARE
7,500
7,000
6,500
6,000
5,500
5,000
4,500
4,000
3,500
3,000
2,500
2,000
1,500
1,000
500
0
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700
750
800
850
900
Time (seconds)
Crucial P5 2TB
Sabrent Rocket 4 Plus 2TB
Corsair MP600 Pro XT 2TB
Kingston KC3000 2TB
Patriot Viper VP4300 2TB
Samsung 980 Pro 2TB (128KB block size)
WD_Black SN850 2TB
Crucial P5 Plus 2TB

Wear Level Management

- Each individual cell has a limited number of program-erase cycles
- SSD firmware ensures each cell is written to evenly
  - Even if we write a full erase-block stripe, can't just rewrite the same block

Overprovisioning

https://download.semiconductor.samsung.com/resources/white-paper/S190311-SAMSUNG-Memory-Over-Provisioning-White-paper.pdf

USE Method

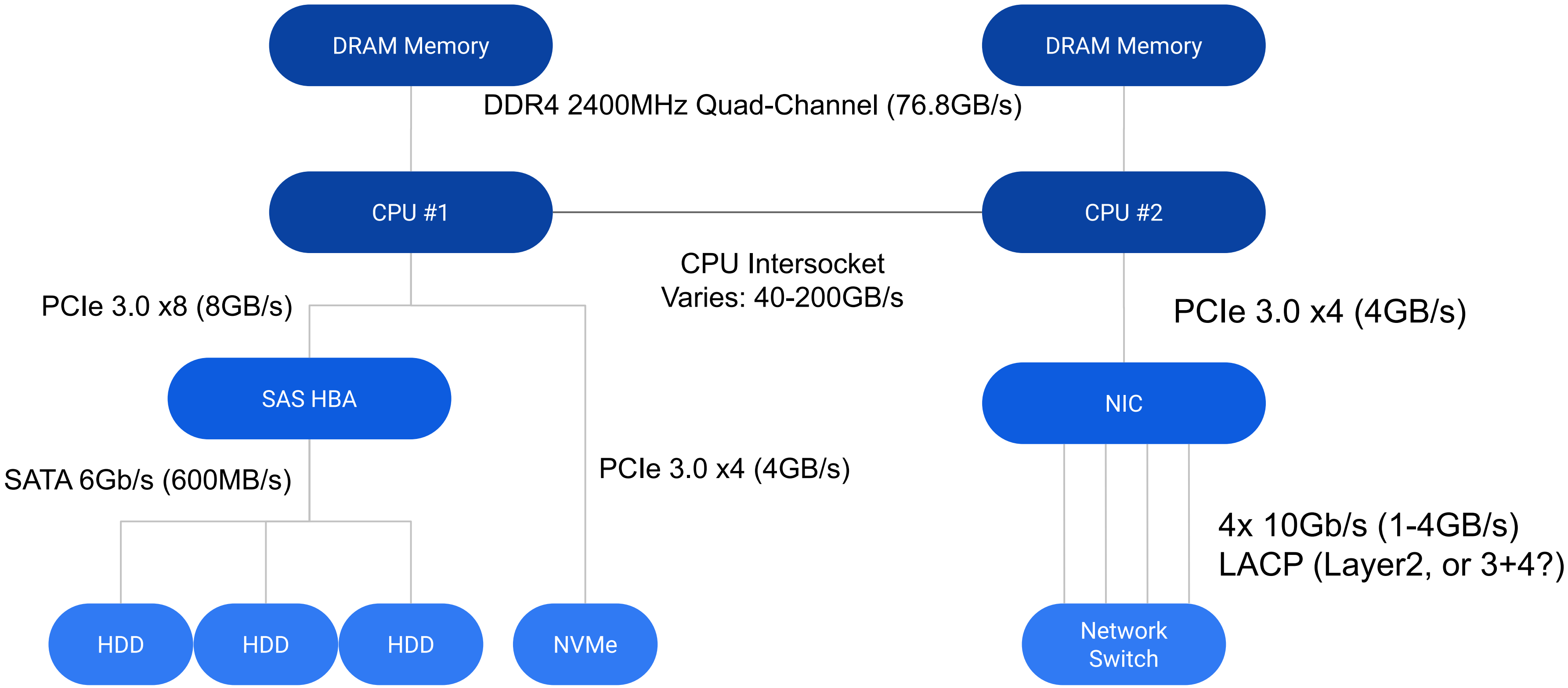
Utilisation

Saturation

Error Counters

More: https://www.brendangregg.com/usemethod.html

Hardware Bottlenecks



Software Bottlenecks

- Saturating a single CPU core (or all of them)
  - Kernel
  - Benchmark Client
- Saturating device queues
  - Hardware
    - Enterprise hardware often supports more queues
    - Some hardware requires more queues enabled in the BIOS
    - Some consumer hardware may only support 1 queue
  - Virtual
    - virtio multi-queue (Block and NIC devices)Benchmark client - Queues Counts

top

```
top - 00:12:14 up 1 day, 19:14,  4 users,  load average: 1.26, 0.80, 0.90
Tasks: 869 total,   3 running, 866 sleeping,   0 stopped,   0 zombie

%Cpu(s):  0.2 us,  6.0 sy,  0.0 ni, 92.7 id,  1.0 wa,  0.0 hi,  0.0 si,  0.0 st

MiB Mem : 128773.1 total,    481.3 free,  31294.7 used,  96997.2 buff/cache
MiB Swap:   8192.0 total,   8191.5 free,      0.5 used.  96423.9 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 775088 root      20   0  218464   7748   2056 R  99.7   0.0   0:25.91 fio
```

top

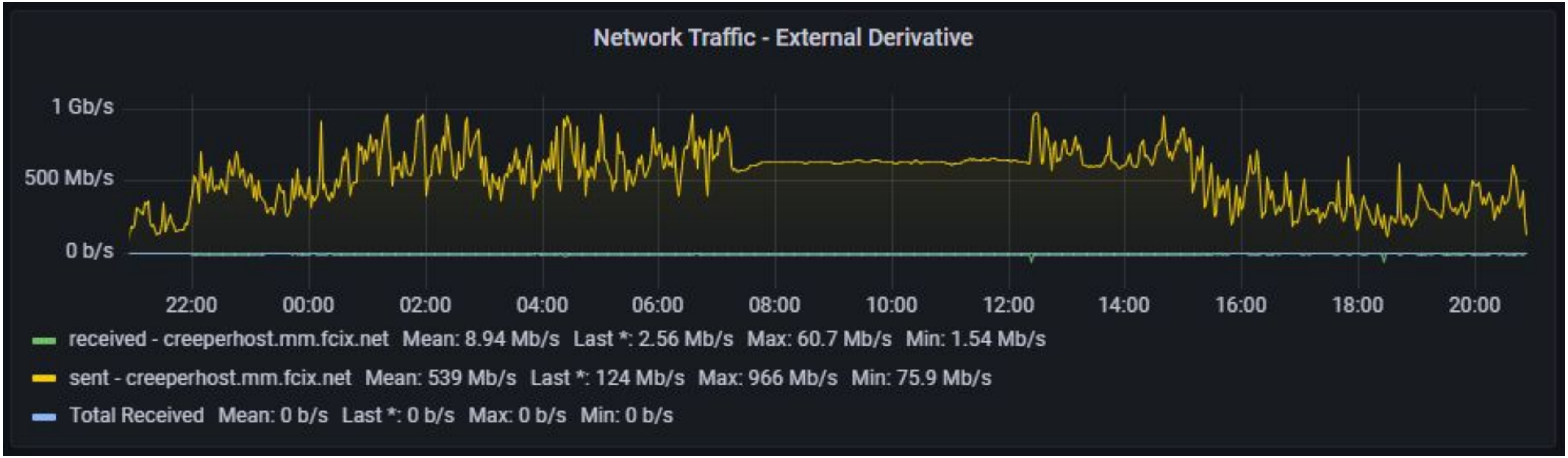
```
top - 00:12:14 up 1 day, 19:14,  4 users,  load average: 1.26, 0.80, 0.90
Tasks: 869 total,   3 running, 866 sleeping,   0 stopped,   0 zombie

%Cpu0  :  0.0 us,  0.7 sy,  0.0 ni, 98.7 id,  0.3 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu1  :  0.0 us, 47.9 sy,  0.0 ni, 22.9 id, 29.2 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  6.3 us, 93.7 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
...
%Cpu31 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 775088 root      20   0  218464   7748   2056 R  99.7   0.0   0:25.91 fio
```


CODINGFLYBOY.MM.FCIX.NET
CONTACT: MIRROR@FCIX.NET

Device Queues - MicroMirror Project



https://fosstodon.org/@lathiat/110228551076334949

Device Queues - MicroMirror Project

https://fosstodon.org/@lathiat/110228551076334949

Receive/Transmit Packet Steering (CPU Mask)

```
echo f > /sys/class/net/IFACE/rx-0/rps_cpus
echo f > /sys/class/net/IFACE/rx-0/xps_cpus
```

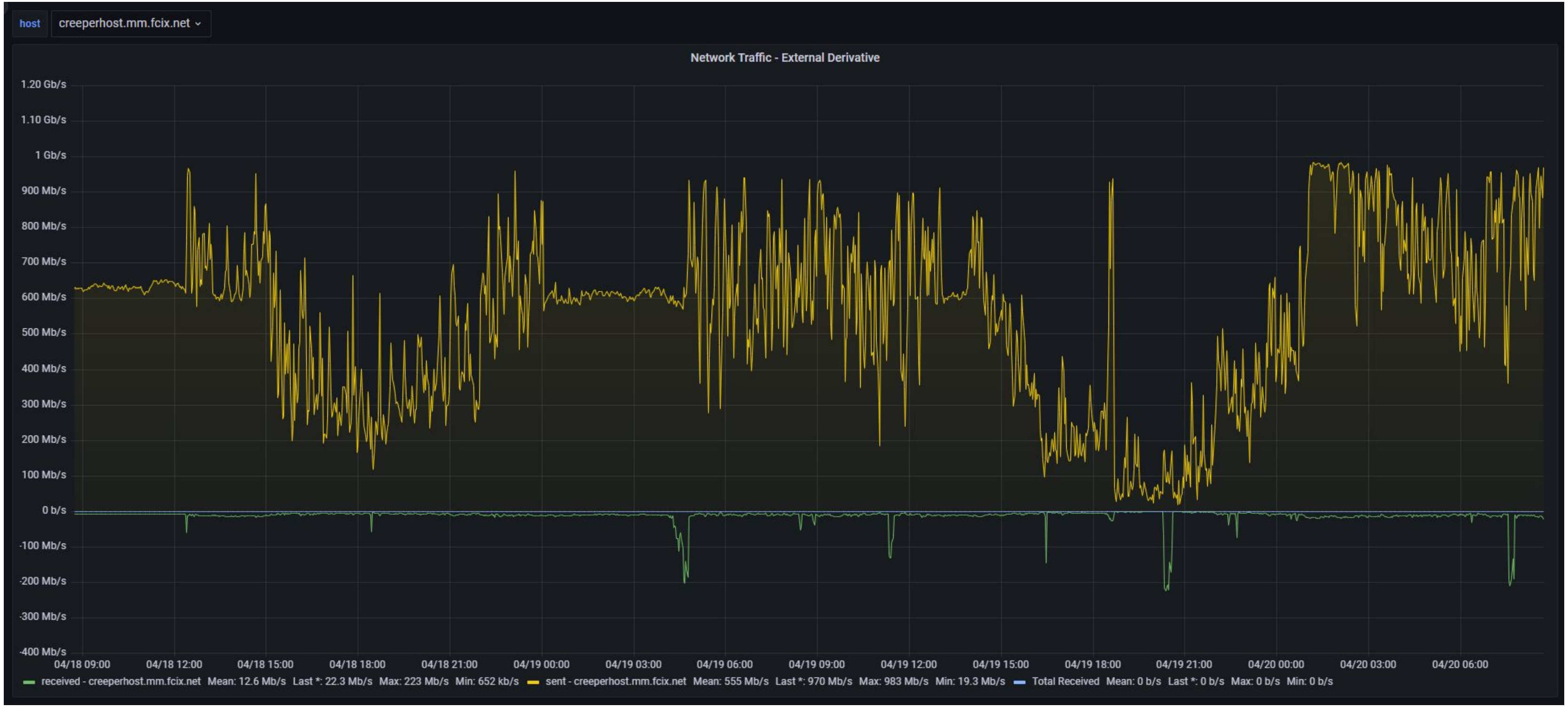
Device Queues - MicroMirror Project

mirror@creeperhost:~

```
 26   1  73   0   0| 592k    0 |2183k   73M|   0     0 |2949  1669
 26   2  72   0   0| 256k    0 |2189k   73M|   0     0 |3046  1754
 26   1  73   0   0|   0     0 |2186k   73M|   0     0 |3027  1822
 26   1  72   0   0| 128k    0 |2181k   73M|   0     0 |3149  1888
 33   3  63   1   0| 812k    0 |2151k   72M|   0     0 |3763  2372
 26   1  73   0   0| 128k    0 |2169k   72M|   0     0 |3191  1988
 26   1  72   0   0|   0  8192B|2184k   73M|   0     0 |3069  1926
 26   2  72   0   0| 856k    0 |2188k   73M|   0     0 |3131  2018
 26   2  72   0   0| 340k    0 |2186k   73M|   0     0 |3169  2011
 26   1  72   0   0|  11M    0 |2202k   75M|   0     0 |2990  1873
 26   1  72   0   0|1792k    0 |2164k   72M|   0     0 |3171  1913
 26   1  73   0   0|1413k  127k|2164k   72M|   0     0 |3093  1860
 31   1  67   0   0|9490k    0 |2482k   86M|   0     0 |  11k 1894
 47   2  51   0   0|  14M   12k|3436k  117M|   0     0 |  18k 1873
 54   5  40   1   0|9341k    0 |3452k  117M|   0     0 |  16k 2279
 47   2  51   0   0|9187k    0 |3478k  117M|   0     0 |  17k 1990
 47   2  51   0   0|9618k   16k|3457k  117M|   0     0 |  17k 1796
 46   2  52   0   0|  22M    0 |3465k  117M|   0     0 |  24k 1904
----total-usage---- -dsk/total- -net/total- ---paging-- ---system--
usr sys idl wai stl| read  writ| recv  send|  in   out | int   csw
 46   2  51   0   0|  22M    0 |3432k  117M|   0     0 |  24k 1970
 47   1  51   0   0|  12M    0 |3400k  117M|   0     0 |  18k 1662
 47   2  51   0   0|6141k    0 |3442k  116M|   0     0 |  17k 1546
 46   2  52   0   0|  14M 4100B|3454k  117M|   0     0 |  18k 1703
```

https://fosstodon.org/@lathiat/110228551076334949

Device Queues - MicroMirror Project



https://fosstodon.org/@lathiat/110228551076334949

Device Queues - Cisco UCS Ceph Cluster

```
$ grep eth5 /proc/interrupts
eth5-rx-0
eth5-tx-0
eth5-err
eth5-notify
```

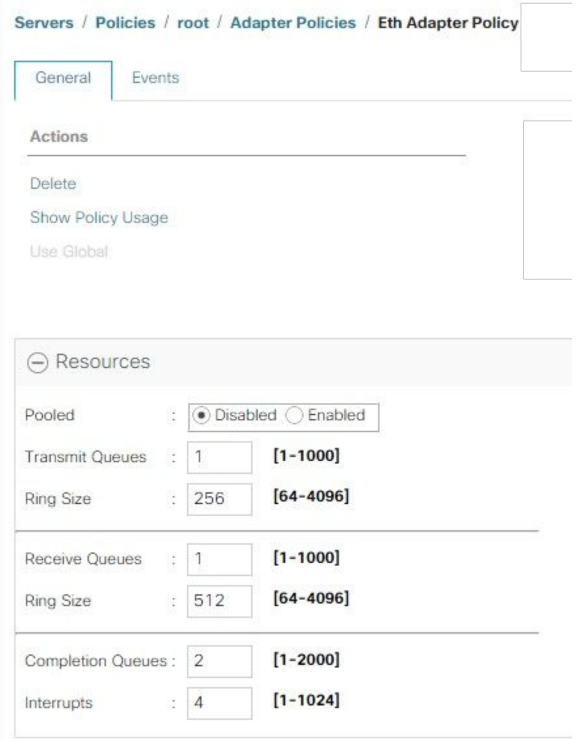
https://toreanderson.github.io/2015/10/08/cisco-ucs-multi-queue-nics-and-rss.html

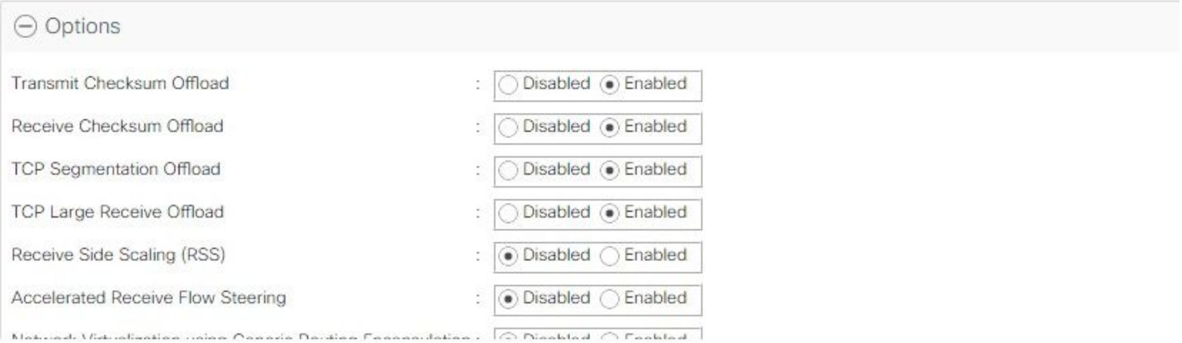
Device Queues - Cisco UCS Ceph Cluster

```
$ grep enp6s0 /proc/interrupts
enp6s0-rx-0
enp6s0-tx-0
enp6s0-err
enp6s0-notify
```

https://toreanderson.github.io/2015/10/08/cisco-ucs-multi-queue-nics-and-rss.html

Device Queues - Cisco UCS Ceph Cluster





https://toreanderson.github.io/2015/10/08/cisco-ucs-multi-queue-nics-and-rss.html

Less known USE Tools

Find new tools
sosreport
https://www.brendangregg.com/USEmethod/use-rosetta.html

Non-hardware statistics
sar
dstat

Physical Hardware Layout / Link Speeds
lstopo --of ascii
lspci -vv

Network Errors
netstat -s
ethtool -S

Read-Modify-Write

- Various layers have a minimum write size which varies
- Any smaller write must first read the rest of it, modify it, then write it back
- Causes
  - Checksums/ECC/Parity (HDDs, RAID5, ZFS)
  - Compression (ZFS)
  - Snapshots (LVM, Virtual Disks, ZFS)
- Trap: Partition Alignment

Read-Modify-Write

- Hard Disks - Hidden CRC
  - Traditionally: 512B: Smaller writes rejected
  - Most modern drives: 4096B (4KiB)
    - **512e** emulation: Transparent Read-Modify-Write
    - **4Kn** native mode: Smaller writes rejected
- Linux Page Cache - 4KiB
  - All writes go to the page cache unless bypassed
  - Small writes will read the rest of the 4KiB block before returning
    - Even for non-blocking/asynchronous I/O!
- ZFS - Checksums, Compression, Snapshots
  - Files: Dynamic per-file based on the size of the first write
    - Minimum: **ashift**, 8=512KiB, 12=4KiB, 13=8KiB
    - Maximum: **recordsize** (Default: 128KiB)
  - zvol: Fixed at volblocksize - default 16KiB
- Ceph - 4KiB for Checksums, 16-64KiB for Compression

Read-Modify-Write: Non-4K aligned writes

- Non-4K aligned writes
  - Linux Guests: Always fine
  - Windows Guests: Will write 512-byte aligned by default
    - Unless virtual disk has physical_block_size=4096 hint (usually doesn't)
    - Ceph intentionally does not cache data even briefly. This will happen even if you read and immediately write the data.
  - Fixing I/O performance for Windows guests in OpenStack Ceph clouds: https://www.youtube.com/watch?v=_vfGcsvnn6U
- Mis-informed etcd fio test
  - https://www.ibm.com/cloud/blog/using-fio-to-tell-whether-your-storage-is-fast-enough-for-etcd (now removed)
  - `fio --rw=write --ioengine=sync --fdatasync=1 --size=22m --bs=2300 --name=mytest1`

Sparse Allocation / Thin Provisioning

- Writes
  - Penalty typically incurred when you first write to an area of a disk
- Reads
  - Impossibly fast to read all-zero data that isn't actually on-disk

- Discard/Trim may happen at random, sparse-ifying your thick provision
  - Many things are nervous to trim
    - Many early implementations performed badly, wouldn't queue or corrupted data
  - By default most Linux FS don't trim on the fly
  - But fstrim is scheduled weekly
  - LUKS encryption disables trim by default to prevent information-leak

Working Set Size

- Write a realistic total amount of data to each drive/the entire cluster
- Issue reads and writes to a realistic percentage of the total storage
    - fio
        - --io_size
        - zipf distribution

Why? Caching

- Ensures cache hit rates approximate what you'll see in production
- Small benchmarks (10s of GB) often fit in every possible cache
  - Memory caching
    - Metadata: indirect block allocation maps
    - Data itself
  - SSD Caching (bcache, lvmcache, storage tiering, etc)
    - May not trigger writeback
    - May never read/write to the actual backing HDD

Benchmarking Workload

- Tool Selection
  - **Bad:** hdparm, dd, cp, rados bench
  - **Good**: fio (with the right config)
- Parallelism
  - Parallel I/O submission
  - Multiple VMs on Multiple Hosts
    - Multiple Client IPs, MACs
    - Multiple Network Links
- Filesystem vs Raw Block
  - ext4 inode table lazy initialization
  - Metadata Overheads
  - Lock Contention

Ceph Specifics

- Thin Provisioning-type behaviour
  - RBD images are thin provisioned by default
  - Space for both metadata & data is thin provisioned within the OSD itself
- Per-client concurrency limits, eg. objecter_inflight_ops, objecter_inflight
- Periodic RocksDB compaction
- Working Set Size
  - Deep Scrub reads the entire dataset once a week
  - 8TB / 1 week = 13MB/s
  - Significant background load
  - Each 4MB of an RBD = 1 object = 1 PG = 1 OSD
- Dynamic scaling - will impact benchmarks, especially right after deploy/creation
  - PG Autoscaler
  - PG Balancer
  - RadosGW Bucket Sharding
- Erasure Coding = worst case Read-Modify-Write with network latency

"Tuning"

- Always question & validate tuning guides
  - Many tune all sorts of crazy values
  - Usually aiming to achieve some maximum throughput for marketing
  - Almost **always** at the expense of **latency** and **crash safety**
  - A bigger buffer value is not always faster

"Tuning"

- Don't assume changing a config option dynamically **actually** works
  - You may need to re-connect the TCP connection
    - Examples: net.ipv4.{tcp_rmem,tcp_wmem}
  - You may need to re-create the storage file
    - Examples: zfs compression, recordsize
  - You may need to restart the OSD
  - You may need to stop/start the VM

Simple guidelines

- Benchmark raw block device instead of a filesystem
- Pre-condition/pre-write the entire virtual disk size at the start
  - Eliminates any thin provisioning overhead or speedups
- Fill the underlying storage to a reasonable percentage (60-80%)
  - Minimises variance from SSD SLC-caching/Garbage Collection
  - Reproduce production deep-scrub impact in Ceph
  - Avoid filling filesystems above 90% to avoid abnormal fragmentation
- Working Set Size
  - Avoid writing to only a small portion of a virtual disk
  - Write to the disk at random or use a distribution like zipf
  - Benchmark a range of sizes and watch the effect
- Benchmark duration
  - Benchmark long enough to push any SSDs past the bi-modal speed cut-off
- Switch back and forth between the two compared options
  - Ensure the performance change is repeatable in both directions
- Research and determine the limiting factor, improve it, try again

Why do we need to do all of these things when we don't actually do them for the production workloads?

Questions

lathiat.net/talks
twitter.com/lathiat
@lathiat@fosstodon.org
linkedin.com/in/lathiat

trent.lloyd@canonical.com

https://fosstodon.org/@lathiat