Solid State Drive cache testing with Flexible IO Tester

(original title: Testujeme Solid State Drive cache s Flexible IO Tester)

Adam Kalisz
adamkalisz.eu
adam_kalisz [at] wh2.tu-dresden.de

Outline

1) Overview of the problem
2) Suggested way to a solution
3) About FIO
4) About caching, dm-cache
5) Tests and results

The problem

When we plan for new storage, can we:

- Get an idea about the performance of current storage?
- Plan needed capacity (development) + performance of needed storage?
- Organize a test system?
- Verify the performance on the storage we bought?

A way to solution

- Metrics and relevant data
- Available storage and interconnect technology
- What data, what methods, what tools
- Testing (before changes)
- Verifying (after changes)

Metrics

- Input/ Output (IO) performance
  - Possible, actually used, needed
  - Type (4K, 1M?) and ratio (Read:Write)
- Throughput (MB/s)
- Latency
- Access patterns
  - When, how, how often accessed
  - What data (DB, VM, fileserver, cold storage)

Technology

Storage tiering:

- Fast
  - (Non-Volatile) RAM
  - Solid State Drive
- Slower
  - Hard Disk Drive (Perpendicular MR, Shingled MR)
- Slow access
  - Tape
  - Off-site over the internet

Technology continued

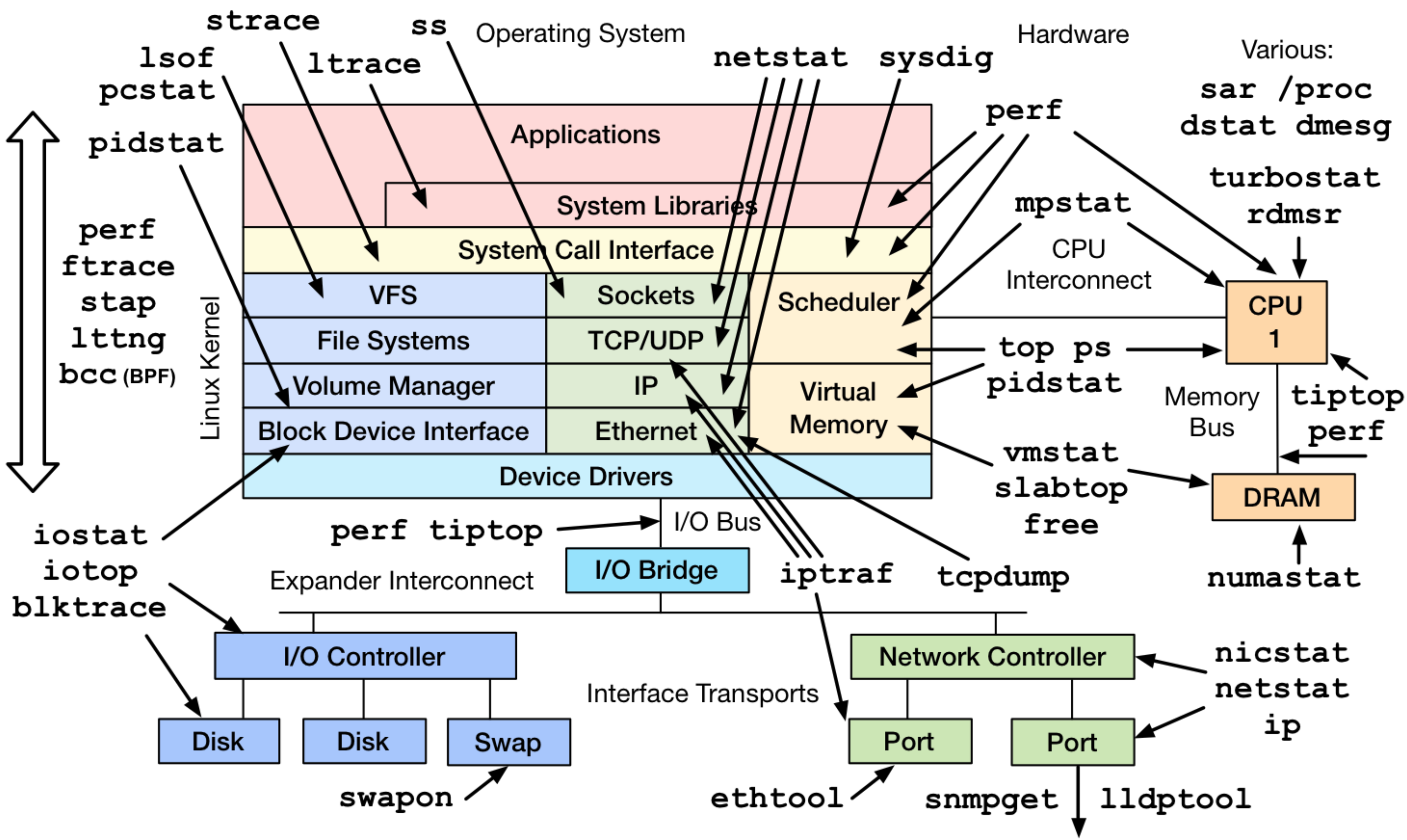
Type of connection:

- SAS, SATA, NVMe (local)
- iSCSI, Fibre Channel, InfiniBand (over network)
- Other (API?)

Start by observing

- Use your tools:
  - Top, Sar, tcpdump, BPF, Dtrace, performance monitor etc. pp.
  - http://www.brendangregg.com/
- Ask questions, try to answer them, don’t blame
  - http://www.brendangregg.com/methodology.html
- Establish good overview of access patterns with long time monitoring
  - What data gets accessed how often by whom
  - When are the peaks

Linux Performance Observability Tools



http://www.brendangregg.com/linuxperf.html 2017

Assuming you have observed and gathered your performance data and do want to answer some questions with testing…

Flexible IO Tester (Jens Axboe)

- FIO can use a block device or a directory/ file
- FIO runs on GNU/ Linux, Windows, macOS, BSDs, Illumos/ Solaris etc.
- Tests are specified using *jobs*
- Jobs on command line or in ini-like formated job file(s)
- Optional *global* section and a section for each job
- Job specification can contain simple arithmetic and units or other expressions (range, list, bool, int, str)

FIO Jobs

```
; -- start job file --
[global]
rw=randread
size=128m

[job1]

[job2]

# -- end job file --
```

OR

```
$ fio --name=global --rw=randread --size=128m --name=job1 --name=job2
```

http://fio.readthedocs.io/en/latest/fio_doc.html#job-file-format

Some tips about testing

Maybe you want to learn the tool first and don’t torture you SSD with limited write cycles or slow HDD with the test too early…

Using RAM for testing

- Using *brd* Linux kernel module (rd_size in 1 kB) like:

  `# modprobe brd rd_size=1048576 rd_nr=1`

- Or *tmpfs* (or *ramfs*) and optionally *losetup* (util-linux package)

  `# mount -t tmpfs -o size=1G tmpfs <PATH>`

  `# dd if=/dev/zero of=<PATH>/ramdisk bs=1M \ seek=1024 count=0`

  `# losetup --show /dev/loop0 <PATH>/ramdisk`

- Verify with `free -h` and `ls -l` or `# fdisk -l <DEVICE>`
- Windows, BSDs etc have their own ramdisk creation methods (ImDisk (Chocolatey pkg) or `New-IscsiVirtualDisk -Path ramdisk:… -Size 20MB` in Windows 8.1+ or Windows Server 2012 R2+, md(4) in FreeBSD, rd(4) in OpenBSD etc.)

Some tips

- Use `fio --output-format=terse --minimal` for CSV-like output or json/json+ fileformat for JSON-like output with or without latency buckets
- `echo 1 > /proc/sys/vm/drop_caches` to drop page cache
- When testing with `time` and `dd`, append `&& sync` to dd for really writing everything from the buffer cache to the block device, FIO has the option `buffered=0` and `direct=1` and others like sync for going around buffers
- Think about looking into sar, iostat (sysstat pkg) and ioping
- Start with small tests you can finish in under 30s and go from there, know how to parse the output **before** you start the long measurements
- AWK, sed and grep -B 1 -A 1 and others are very handy with large log files
- gnuplot can plot big data files, LibreOffice Calc should only be used for aggregated (smallish) data series or you will go nuts
- Use `rate_min` to simulate writing to a tape and avoiding “shoe-shining” effect

A few FIO examples

- `fio --readwrite=randrw --size=500M --blocksize=4K,1M --rwmixread=70 --direct=1 --ioengine=libaio --name=test`

  Random IO read (70%) and write of 500 MB in this directory

- `fio --readwrite=write --verify=crc32c-intel --loops=3 --fill_device=1 --blocksize=4M --direct=1 --filename=<DEVICE PATH> --name=test`

  Fill the device with random data using sequential IO write, do it 3x, do verify (harddrive burn in)

My own examples?

- Try some of the workloads e.g. Anandtech uses for SSD/ HDD/ storage testing
- Observing your workloads is key
- The man page is actually quite good, read it!

References for FIO

- drop_caches: https://www.kernel.org/doc/Documentation/sysctl/vm.txt
- Ramdisk: https://www.kernel.org/doc/Documentation/blockdev/ramdisk.txt
- FIO Output Explained: https://tobert.github.io/post/2014-04-17-fio-output-explained.html
- FIO on Windows: https://tobert.github.io/post/2014-04-28-getting-started-with-fio.html
- Ioping: https://tobert.github.io/post/2015-01-22-ioping.html

Tell me about caches already

- There are many caches
  - Linux VFS has: buffer and page cache, inode and directory cache
  - Hardware caches (harddrives, SSDs, controllers etc.)
- Many possible technologies
  - bcache, flashcache/ EnhanceIO, **dm_cache**
  - ZFS: ZIL on SLOG (~write cache), L2ARC (~read cache)
  - Better HW RAID controllers can have a cache device

Dm-cache and lvmcache

- Merged in Linux Kernel 3.9 (April 2013)
- Vastly improved over time, with kernel 4.2 new stochastic multiqueue policy (smq)
- Included since RHEL 7.1, smq in RHEL 7.2, SLES 12 SP1+ (released in 2015)
- Profits from other work in device mapper, like blk-mq

Setup dm-cache

- Using dmsetup (very mundane)
- Using lvmcache(7) easy, just follow the nice man page
- Origin LV, cache data LV, cache metadata LV
- Metadata LV size min 8 MB, about 1000x smaller than cache data LV, can be on same physical device
- Create cache pool (combine data and metadata)
- Link cache pool and origin LV

Dm-cache FAQ

- You can remove cache pool at any time, the data will be written to the device
- You can use md-raid or dm-cache RAID functionality for cache
- You can change cache mode (default is write through, write back) after setup

```
# lvs -o name,cache_read_hits,\
cache_read_misses <VG>/<LV>
```

```
man 8 lvs, man 7 lvmcache,
# lvs -o help
```

More to the caching theory

- Locality in space and time
- If you use a small amount of data frequently
  - Pareto principle e.g. 80:20 Rule
- Latency, throughput and cost trade-off
- Doesn’t work efficiently when you:
  - read or write all data
  - access data in a random pattern

Test setup

- HPE ProLiant DL360 Gen9 Servers:
  - 2x Intel Xeon E5-2620 v4 (8 Cores, 2,1 GHz Basis, 3 GHz Turbo)
  - 8x 16 GB = 128 GB RAM
- 4x 2,5" 1 TB HDD 7200 rpm in RAID 10 (SAS)
- 2x 128 GB Samsung SSD 850 Pro RAID 1 (SATA) – used as md-raid device with cache on top
- 1x 400 GB Intel SSD DC P3500 (U.2)
- 1x 400 GB Intel SSD DC P3700 (U.2)
- CentOS 7.3 AMD64

More to hardware specs

- RAID 10 HDD (120 IOPS pro HDD):
  - Read: 4x 120 IOPS = 480 IOPS
    → at 4 kB blocks: 1,875 MB/s
  - Write: (4x 120 IOPS)/2 = 240 IOPS
- One Samsung SSD 850 Pro (128 GB):
  - 4k random read (QD1): 10 000 IOPS ~ 39 MB/s
  - 4k random write (QD1): 36 000 IOPS ~ 140,6 MB/s
  - sequential (R, W): 550 MB/s, 470 MB/s
- Intel SSD DC P3700 (400 GB):
  - Max sequential R, W: 2700 MB/s, 1080 MB/s

Test setup continued

- 10 concurrent Jobs
- 2,5 GB/ job with orig. data

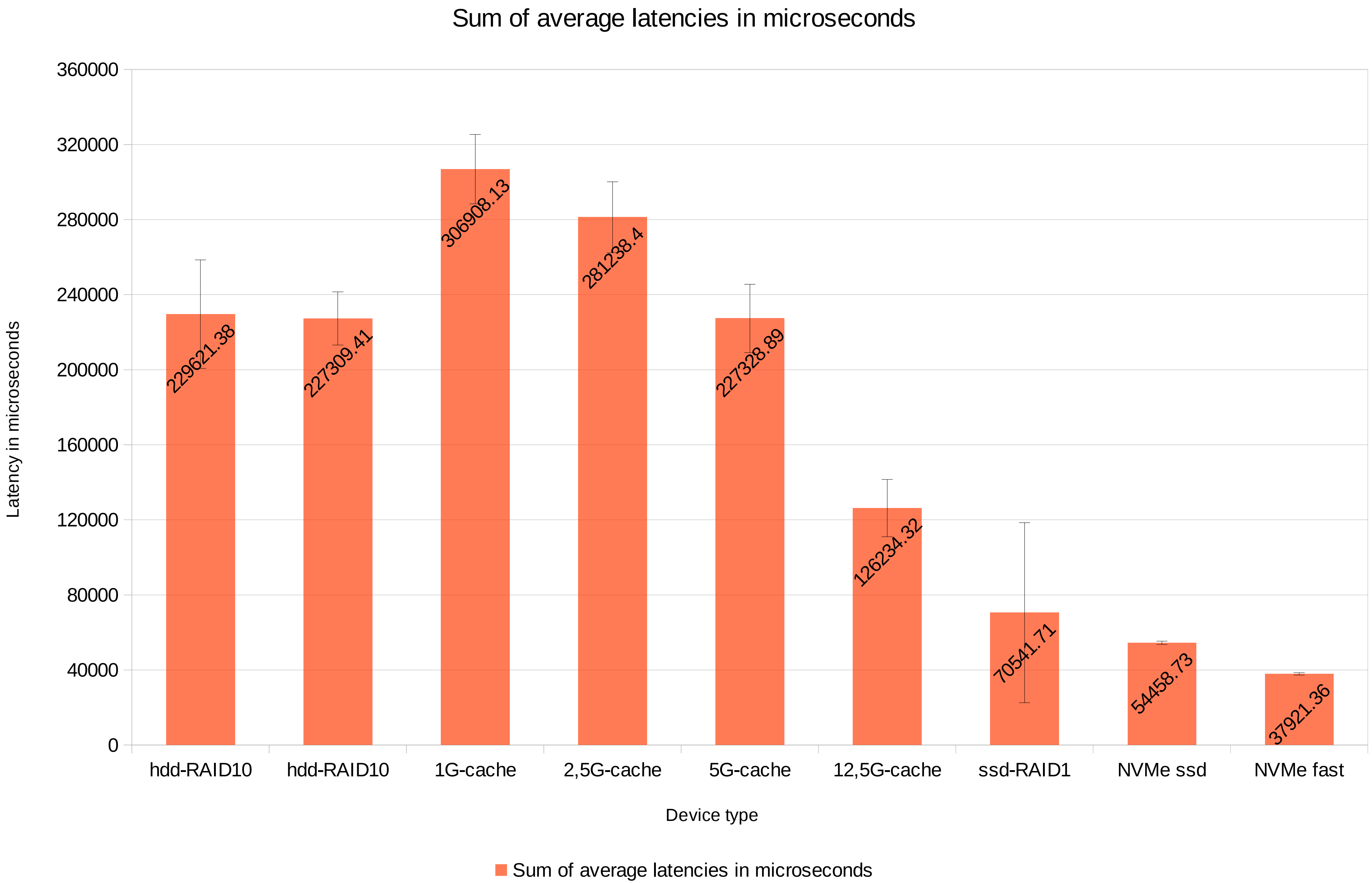
  → 25 GB/ run
- All executed 3x

Jobs	
Random	Sequential
R100:W0	R100:W0
R0:W100	R0:W100
R50:W50	R50:W50
R70:W30	R70:W30
R80:W20	R80:W20

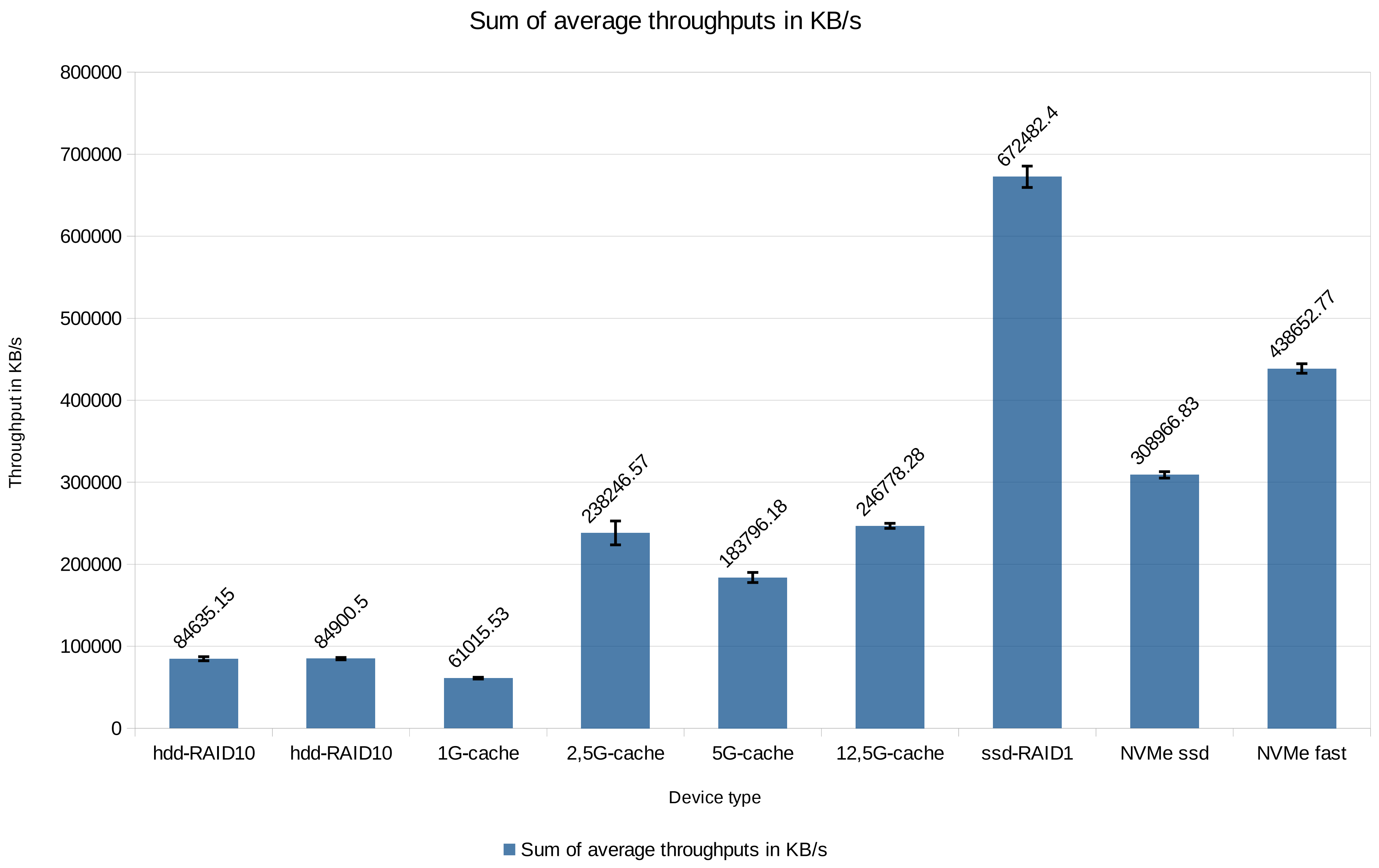
Cache sizes and ratios

- Cache size on RAID1 SATA SSD:
  - 1 GB (ratio 1:25)
  - 2,5 GB (ratio 1:10)
  - 5 GB (ratio 1:5)
  - 12,5 GB (ratio 1:2)
- For comparison:
  - RAID10 HDD
  - RAID1 out of Samsung SSD 850 Pro
  - Intel SSD DC P3500
  - Intel SSD DC P3700

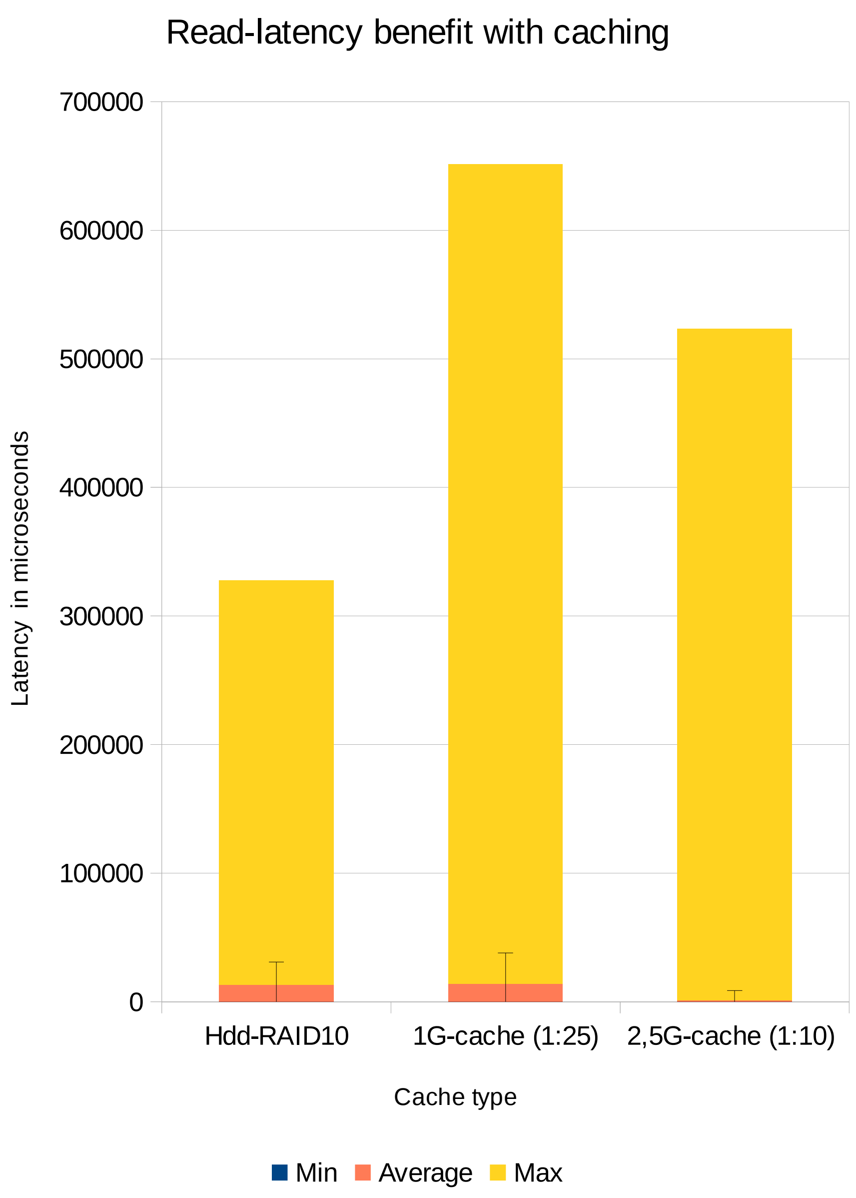
Latency results

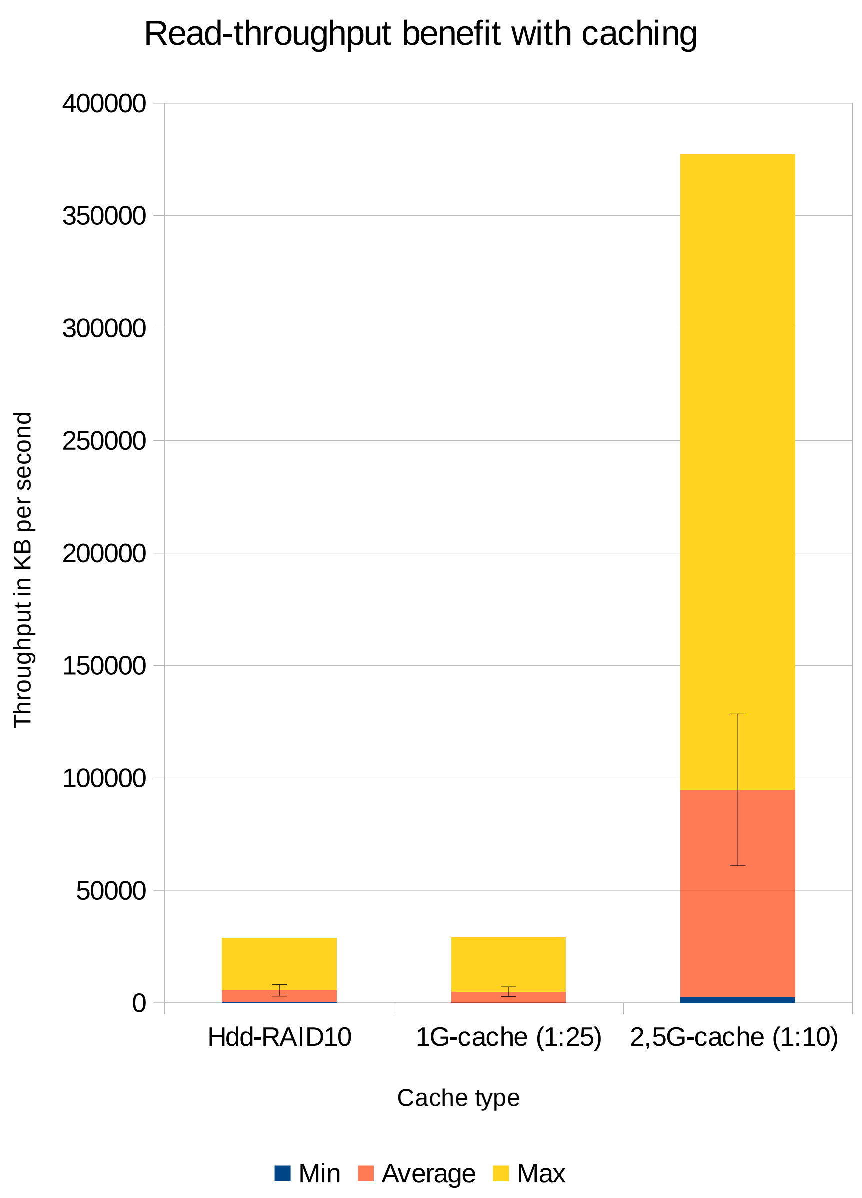


Throughput results

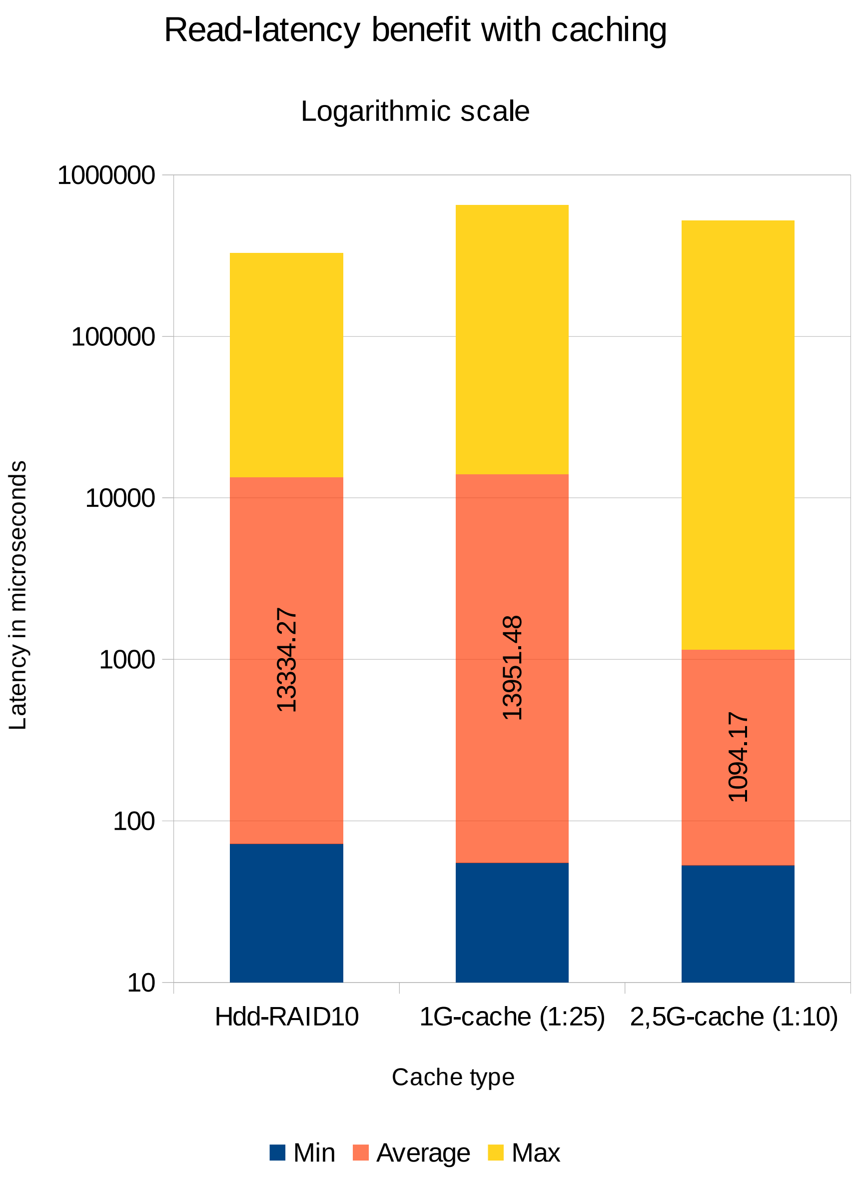


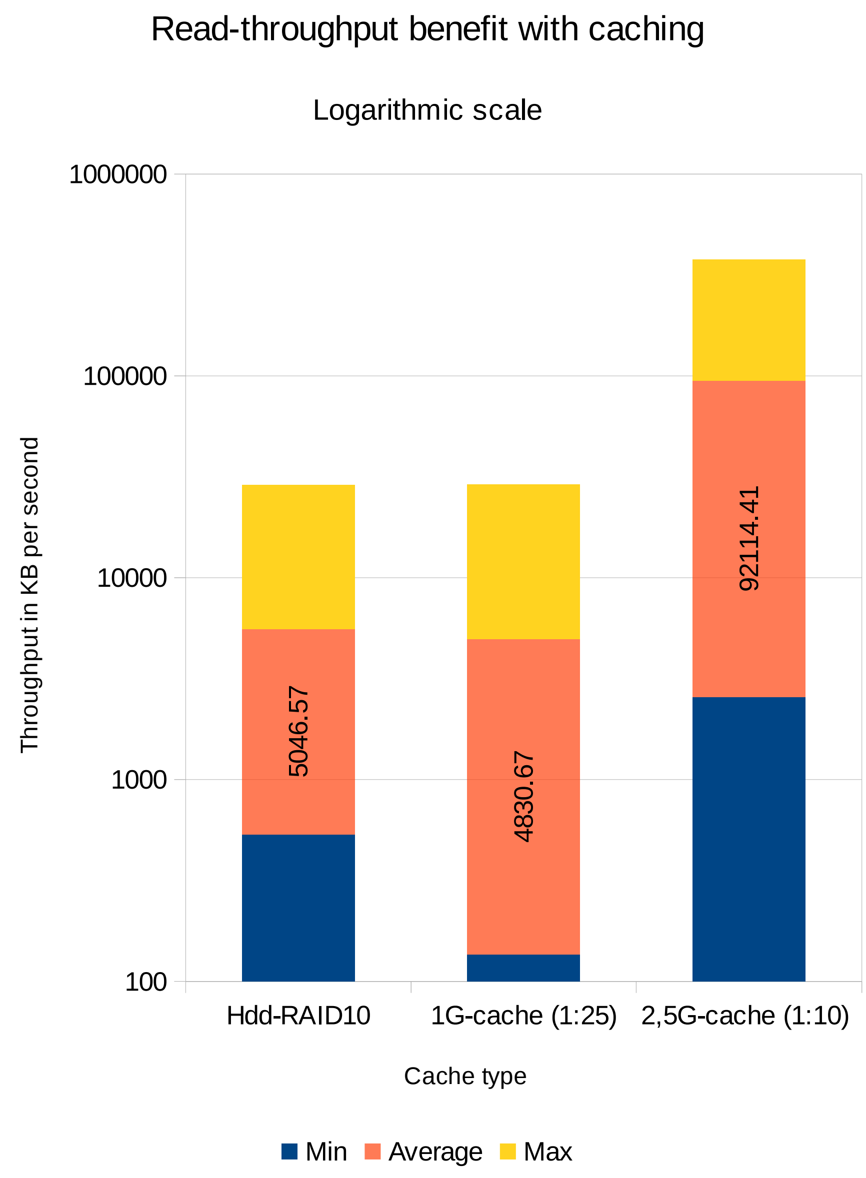
One interesting case





One interesting case (log. scale)





Interpretation

- Performance jump with 1:10 cache ratio
- Different usage of SCSI and NVMe drives
- Higher latency standard deviation with caching

Possible future work

- Test of mostly worst case scenario
- Better testing with larger test scenarios
- Use other access patterns
  - Pareto or zipfian distribution
  - Queue depth
  - Better data analysis
- Test on real data

References for dm-cache

- Cache policies https://www.kernel.org/doc/Documentation/device-mapper/cache-policies.txt
- Dm-cache cache https://www.kernel.org/doc/Documentation/device-mapper/cache.txt
- Dm-cache presentation (Marc Skinner, Q1 2016) https://people.redhat.com/mskinner/rhug/q1.2016/dm-cache.pdf
- Kernel Newbies, kernel 4.2 https://kernelnewbies.org/Linux_4.2#head-4e29dd0a8542b54e319b98f7ed97351dae6211d9
- Kernel Newbies, kernel 3.9 https://kernelnewbies.org/Linux_3.9#head-3dbc54b9324d21f06f55299b0a30d6cb06403529
- My bachelors thesis https://helios.wh2.tu-dresden.de/~adam_kalisz/Bachelorarbeit_AdamKalisz_2017_Druckversion.pdf

Questions and answers