Postgres vs. filesystems

Tomas Vondra

pgconf.eu December 12-15, 2023

EDB

Agenda

- Postgres relies on OS filesystems.
    - I/O scheduling, buffered I/O (page cache)
    - Why does it rely on OS, actually?
    - Good or bad? (Dis)advantages? Alternatives?
- evaluation of current (Linux) filesystems
    - ext4, xfs, btrfs, zfs
    - some basic benchmark numbers
    - problems and recommendations
- Future of Postgres I/O (maybe)
    - direct I/O, async I/O (next talk by Andres Freund)

Test cases

- filesystem: ext4, xfs, zfs, btrfs
- LVM vs. btrfs/zfs
- snapshots?
- compression?
- ...

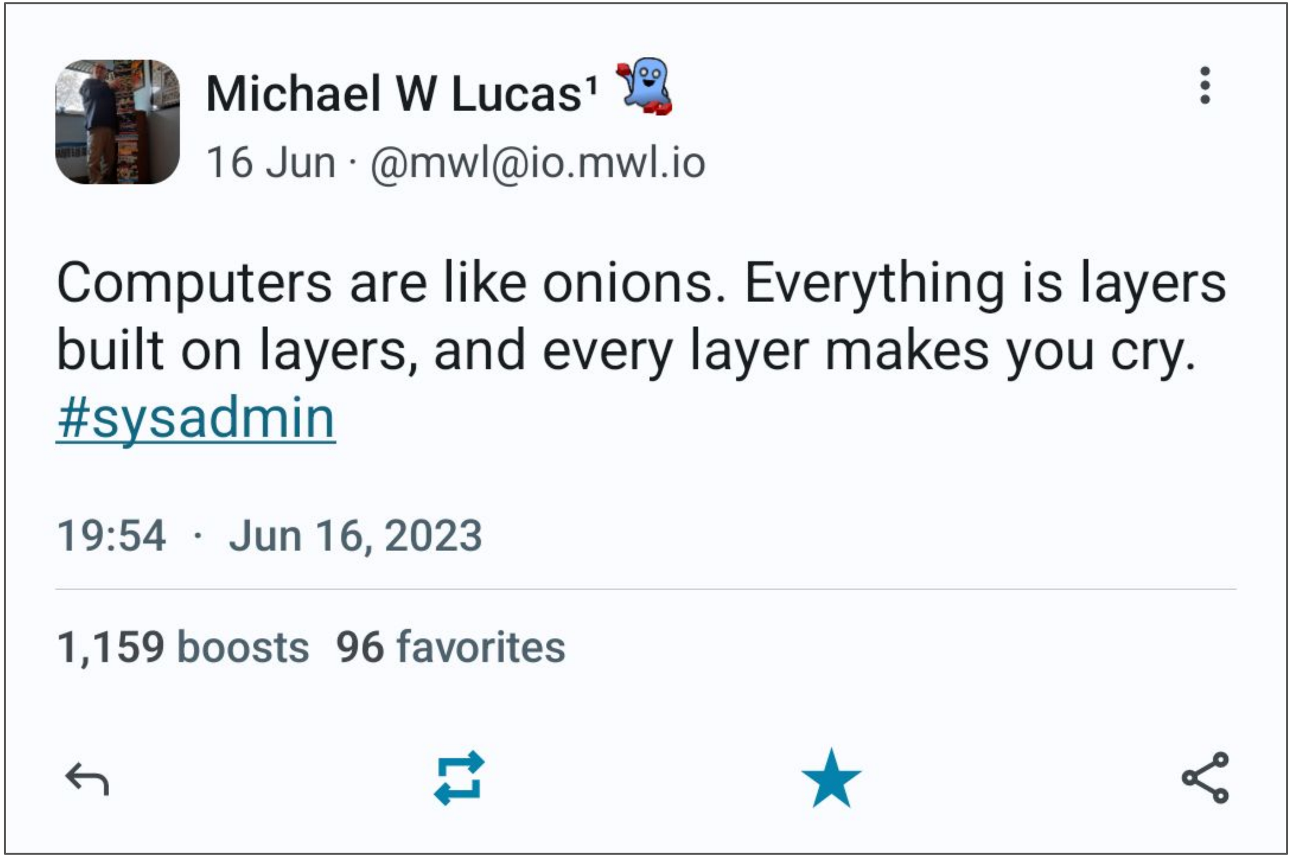
Executive summary

- prefer a mature supported filesystem
  - supported by your distribution & support provider
  - new filesystems are great for research, not for production
- use recent kernels (very important - bugs, ...)
  - numbers will be from 6.3.9
  - bugs, performance improvements, hardware support

Executive summary

- ext4/xfs differences are "relatively small"
  - +10% is nice, but not a go / no-go matter (tuning?)
  - buying better hardware is likely "cheaper"
  - DB tuning easily makes up for this difference
- zfs / btrfs if you actually use advanced stuff
  - but maybe it's simpler to just use LVM ?

Reliance on OS



Michael W Lucas[1]

16 Jun · @mwl@io.mwl.io

Computers are like onions. Everything is layers built on layers, and every layer makes you cry. #sysadmin

19:54 · Jun 16, 2023

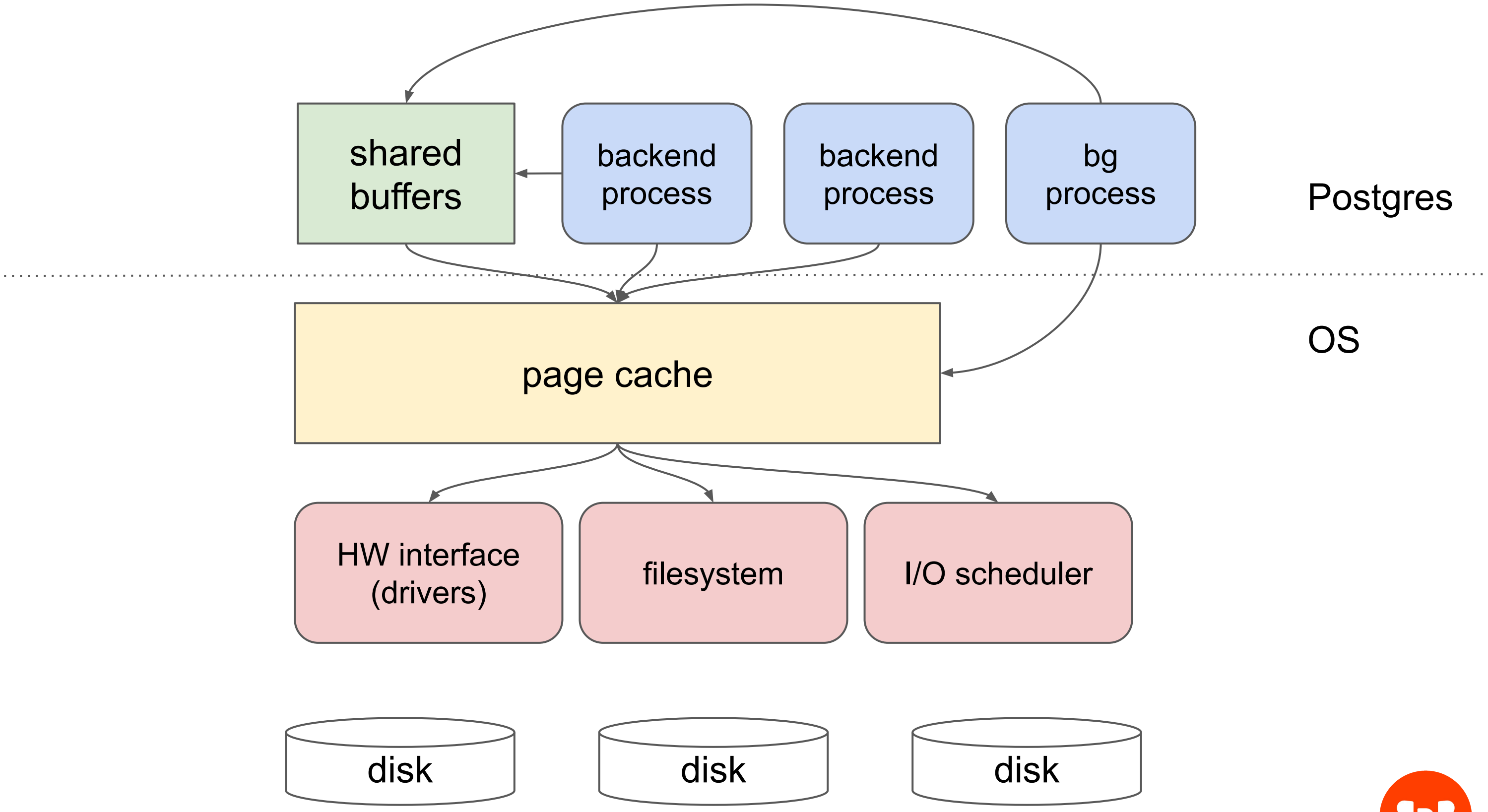
1,159 boosts 96 favorites

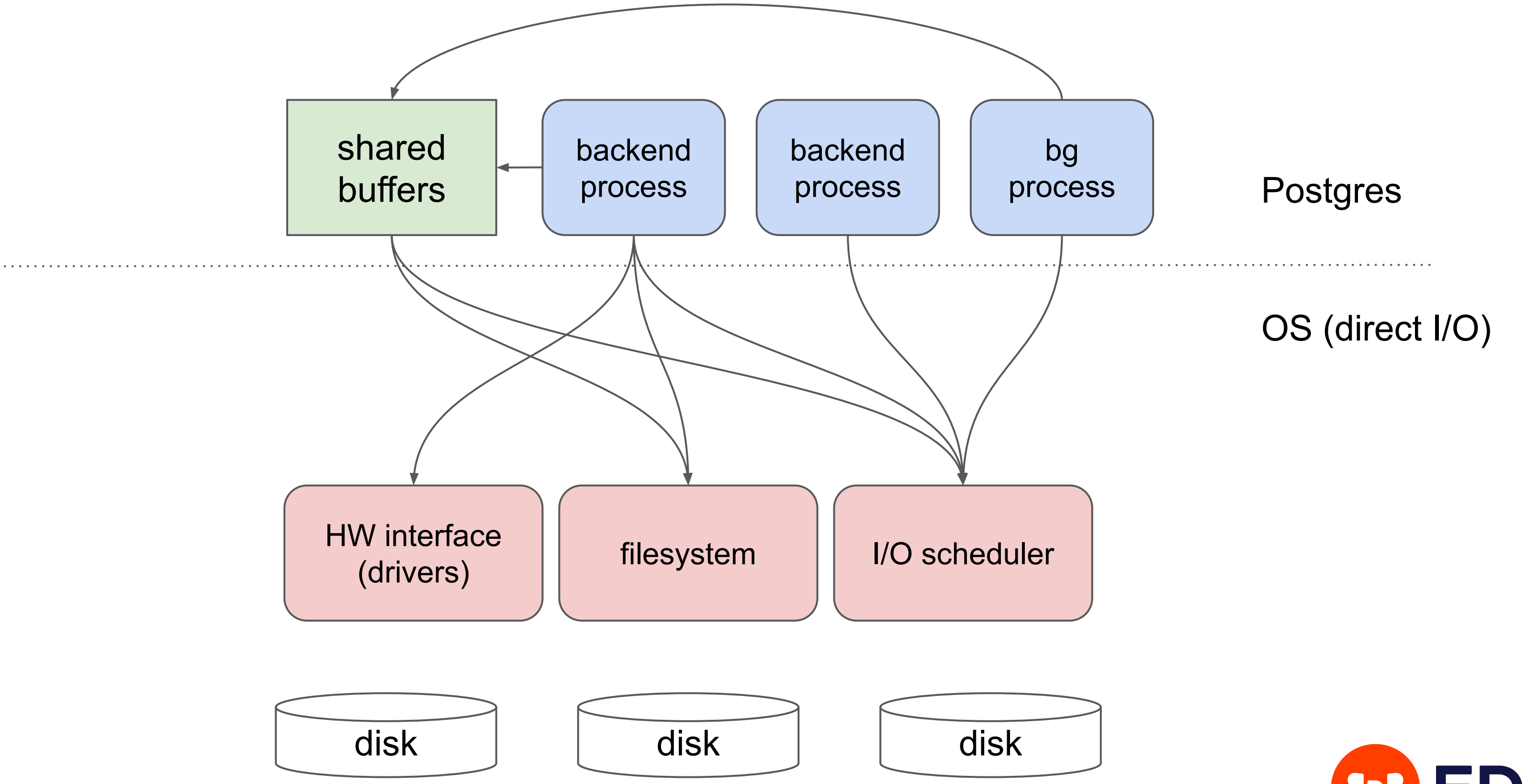
Postgres is a database ...

- storing / accessing data the whole point
- but the low-level stuff is left to the OS
  - OS implements filesystems, provides POSIX interface
- low-level stuff is responsibility of the OS
  - I/O scheduling, caching, sync/async, prefetching (*)
  - handling storage errors (*)

Postgres is a database ...

- is this a good idea?
- historical reasons
  - limited DEV capacity, outside project focus
- would it even be possible to do custom stuff?
  - a lot of supported platforms / different behavior
  - storage hardware changes a lot / quickly
- filesystems do innovate too
  - immediate benefit thanks to that (snapshots, ...)


shared buffers
backend process
backend process
bg process
Postgres
OS
page cache
HW interface (drivers)
filesystem
I/O scheduler
disk
disk
disk


shared buffers
backend process
backend process
bg process
Postgres
OS (direct I/O)
HW interface (drivers)
filesystem
I/O scheduler
disk
disk
disk

Problem #1: error handling

- POSIX is great!
  - but it doesn't guarantee the same behavior everywhere
- what happens after an I/O error during fsync?
- fsync gate (~2018)
  - problems with reporting / handling fsync failures
  - who gets the error with multiple file descriptors?

    (everyone? old/new descriptors?)
  - fs-specific behavior - some throw away the dirty data / mark as clean
  - should be "fine" in new kernels (handled in a no-data-loss way)

Problem #2: lack of visibility

- the OS does great general-purpose scheduling
- the database knows more about the workload, could do better
- example A: it knows what can be done in the background
    - less sensitive I/O, acceptable to delay in favor of user stuff
    - flushing WAL / checkpoints, ...
- example B: prefetching
    - OS has to guess which block will be need next (depends on indexes, ...)
    - we already to explicit posix_fadvise() in a couple places to prefetch async

Basic rule - use recent kernel

- old kernels have all kinds of issues
- bugs
  - fsyncgate (but probably other issues)
  - occasional (performance) regression
- inefficiency
  - general improvements everywhere
  - significant improvements in some filesystems (e.g. BTRFS)

Benchmarks / stress tests

https://github.com/tvondra/fsbench-results

When not under load, all filesystems perform great.

When not under load, all filesystems perform great.

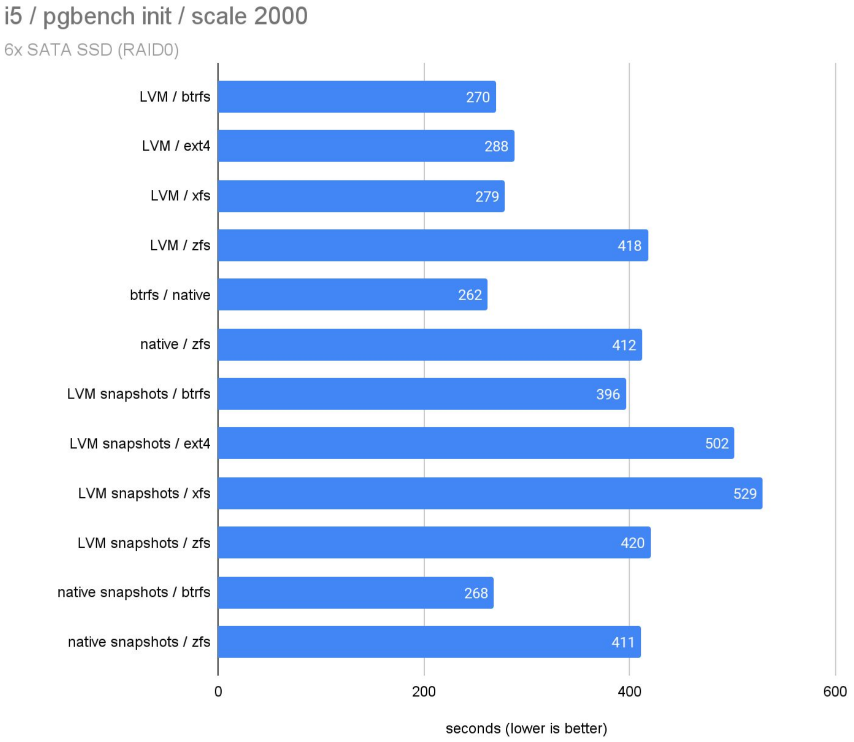
;-)

EDB

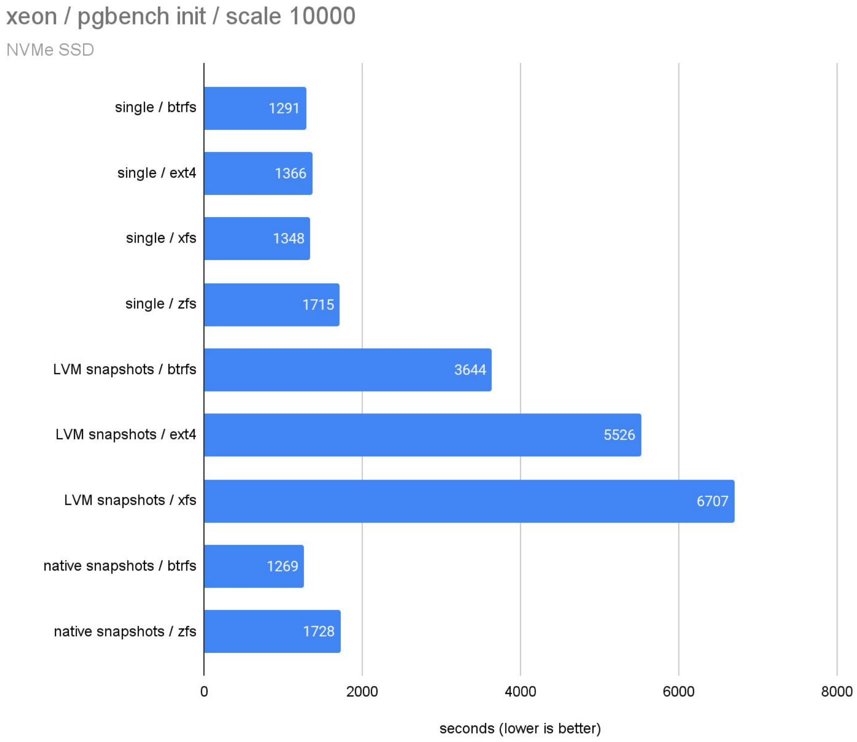
Stress tests are not realistic

- all filesystems have some sort of maintenance / cleanup
    - intended to happen in the background (no disruption)
- stress test = designed to saturate the system
    - do as many transactions as possible
- typical production workload is not 100%
    - aim for ~75% and then consider upgrade
    - makes some of the charts look worse than reality (latency)
- also hardware and configuration-dependent
    - different RAID levels, ZIL/SLOG, ...

Bulk load

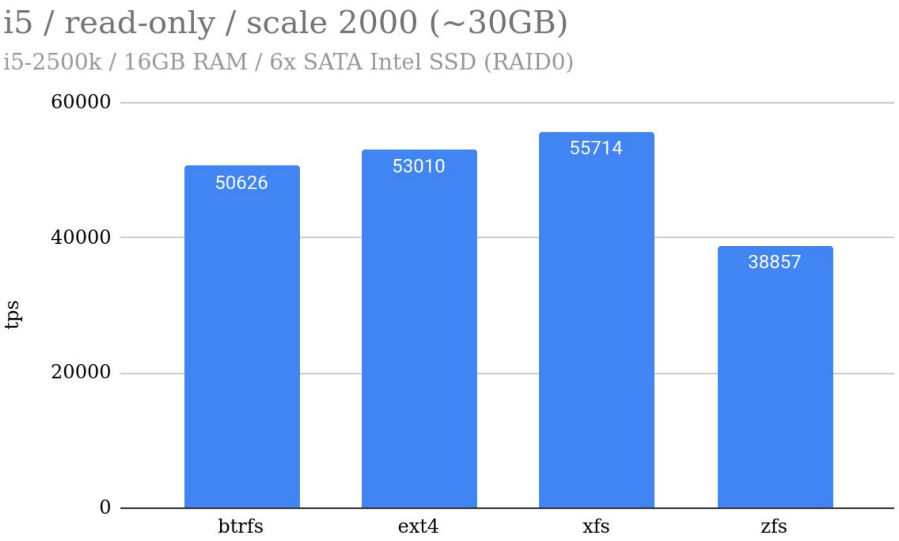


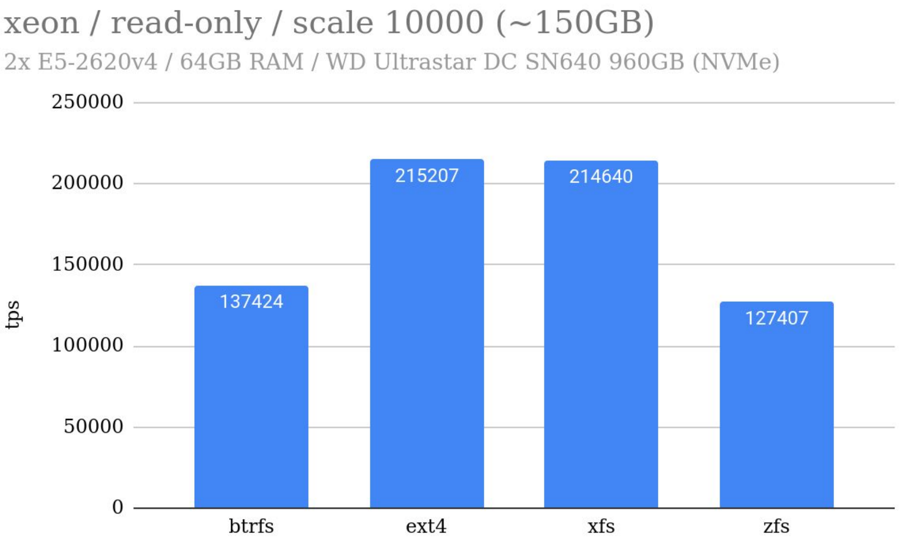
i5 / pgbench init / scale 2000
6x SATA SSD (RAID0)
LVM / btrfs 270
LVM / ext4 288
LVM / xfs 279
LVM / zfs 418
btrfs / native 262
native / zfs 412
LVM snapshots / btrfs 396
LVM snapshots / ext4 502
LVM snapshots / xfs 529
LVM snapshots / zfs 420
native snapshots / btrfs 268
native snapshots / zfs 411
0
200
400
600
seconds (lower is better)



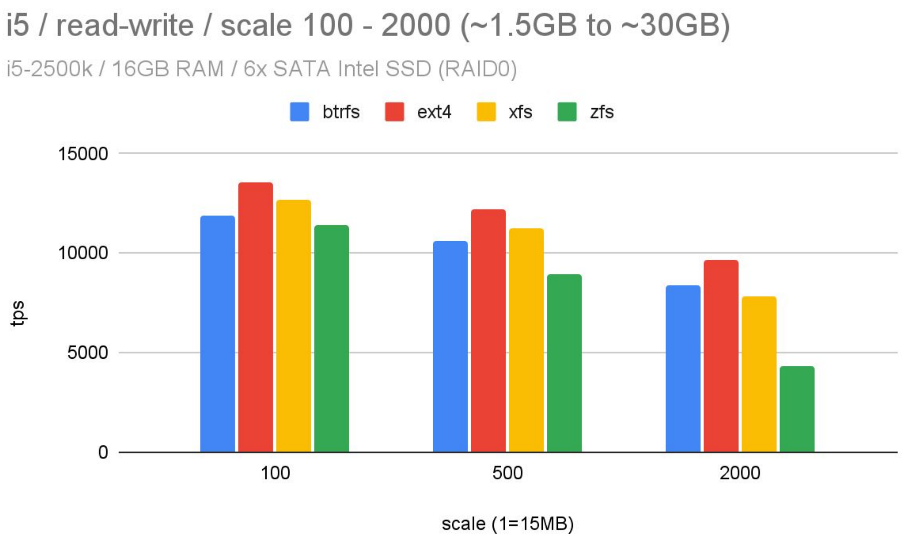
xeon / pgbench init / scale 10000
NVMe SSD
single / btrfs 1291
single / ext4 1366
single / xfs 1348
single / zfs 1715
LVM snapshots / btrfs 3644
LVM snapshots / ext4 5526
LVM snapshots / xfs 6707
native snapshots / btrfs 1269
native snapshots / zfs 1728
0
2000
4000
6000
8000
seconds (lower is better)

OLTP (pgbench, read-only)



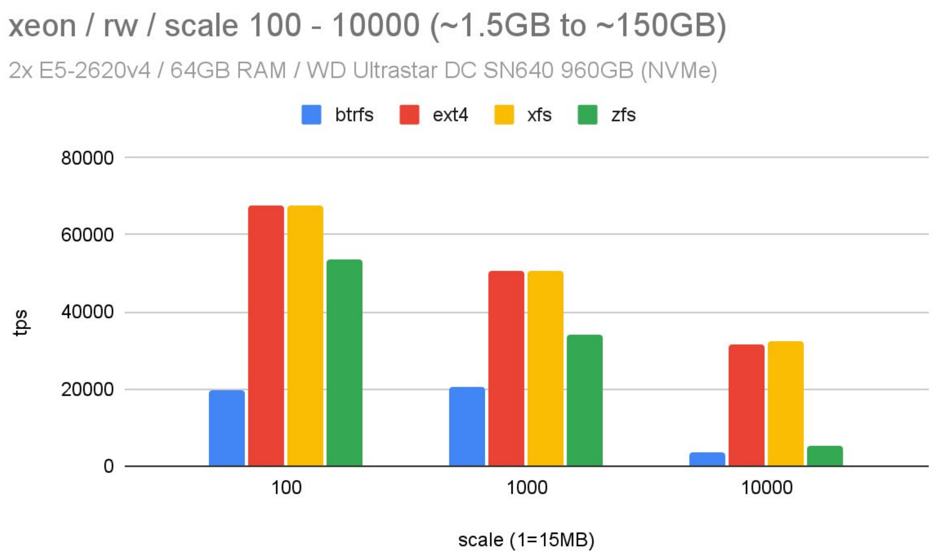


OLTP (pgbench, read-write)



i5 / read-write / scale 100 - 2000 (~1.5GB to ~30GB)

i5-2500k / 16GB RAM / 6x SATA Intel SSD (RAID0)

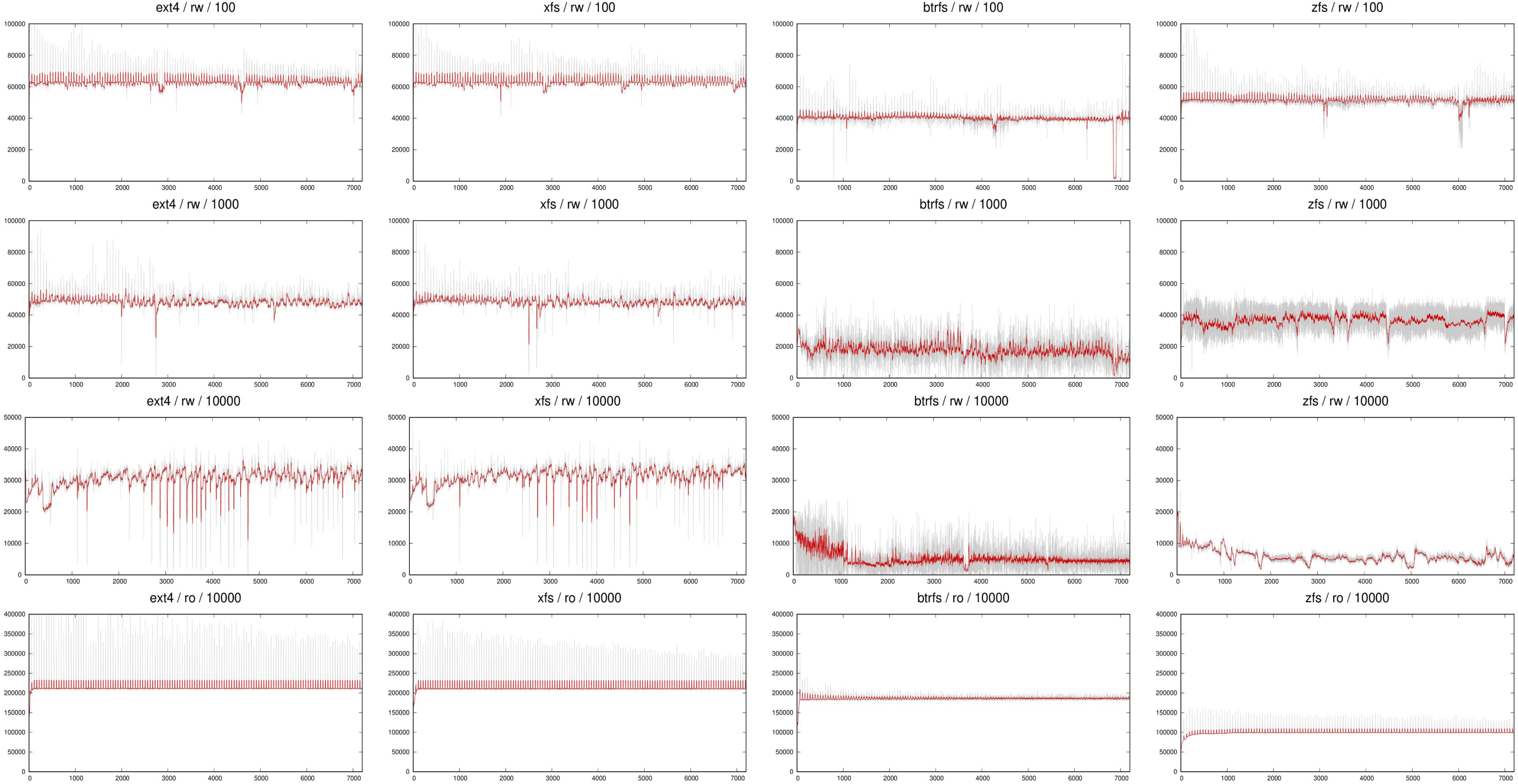


xeon / rw / scale 100 - 10000 (~1.5GB to ~150GB)

2x E5-2620v4 / 64GB RAM / WD Ultrastar DC SN640 960GB (NVMe)

But throughput does not tell the whole story ...

tps (xeon / NVMe)



ext4 / rw / 100

xfs / rw / 100

btrfs / rw / 100

zfs / rw / 100

ext4 / rw / 1000

xfs / rw / 1000

btrfs / rw / 1000

zfs / rw / 1000

ext4 / rw / 10000

xfs / rw / 10000

btrfs / rw / 10000

zfs / rw / 10000

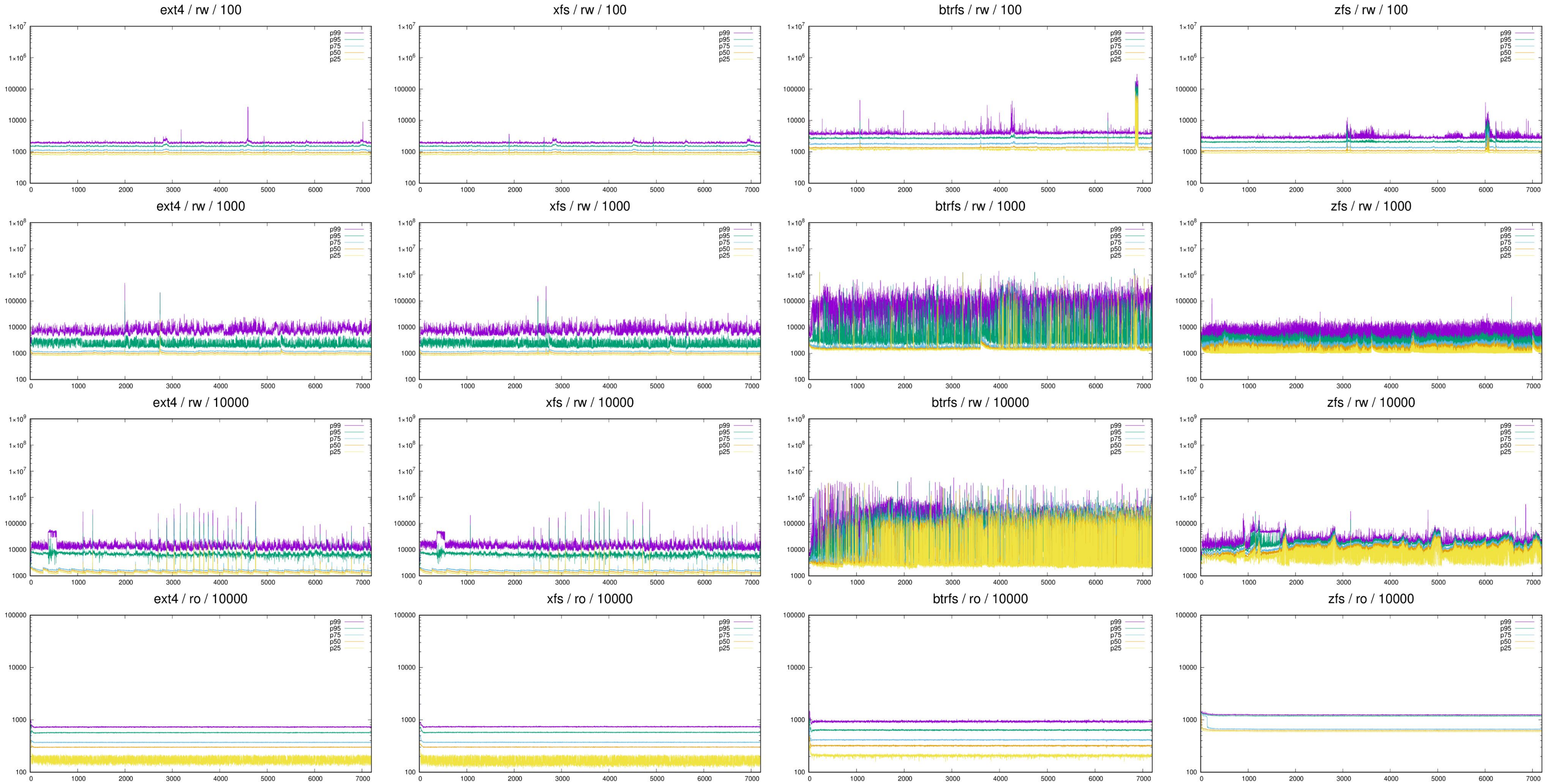
ext4 / ro / 10000

xfs / ro / 10000

btrfs / ro / 10000

zfs / ro / 10000

latencies (xeon / NVMe)



ext4 / rw / 100

xfs / rw / 100

btrfs / rw / 100

zfs / rw / 100

ext4 / rw / 1000

xfs / rw / 1000

btrfs / rw / 1000

zfs / rw / 1000

ext4 / rw / 10000

xfs / rw / 10000

btrfs / rw / 10000

zfs / rw / 10000

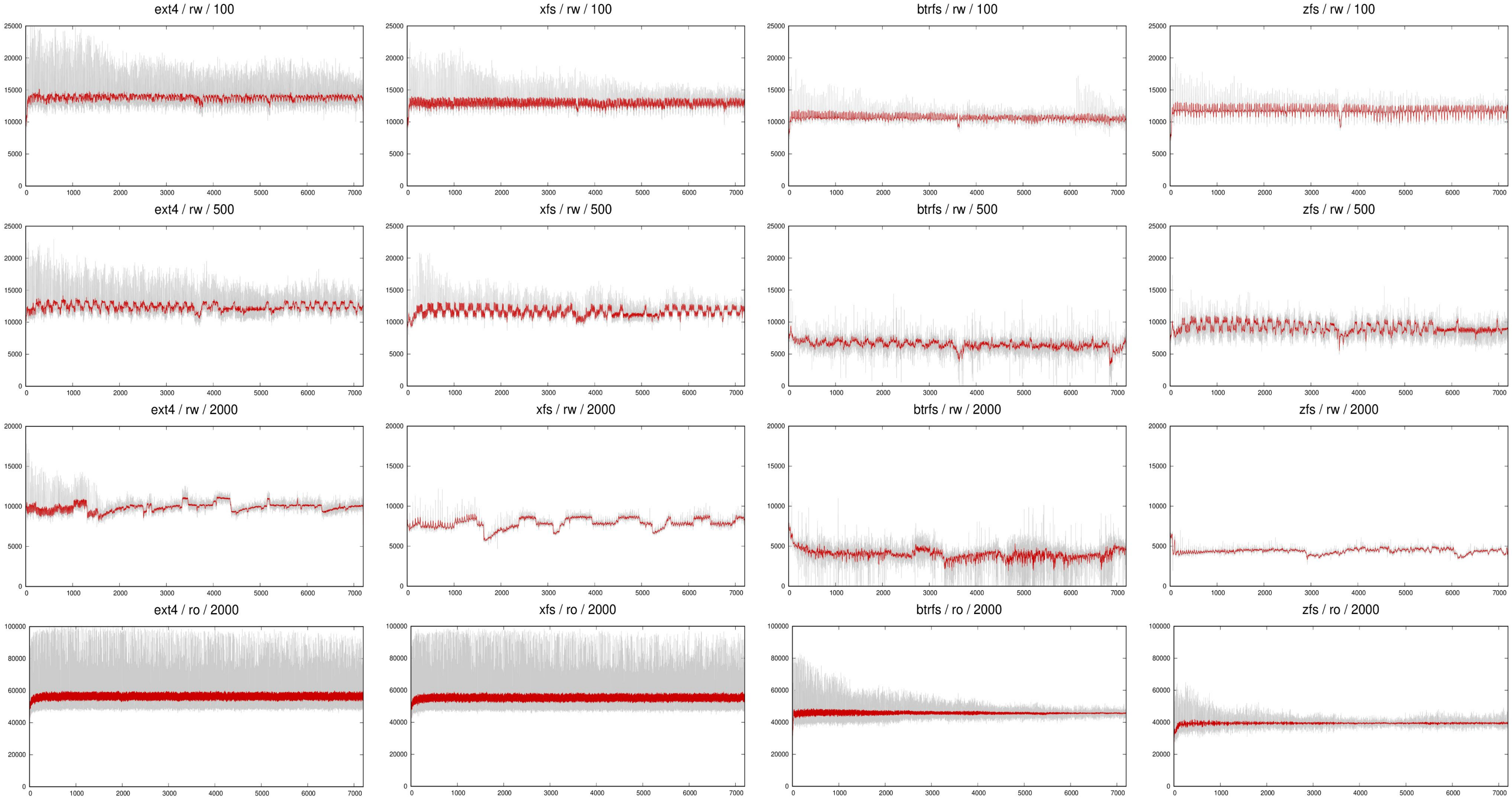
ext4 / ro / 10000

xfs / ro / 10000

btrfs / ro / 10000

zfs / ro / 10000

tps (i5 / SATA SSD)



ext4 / rw / 100

xfs / rw / 100

btrfs / rw / 100

zfs / rw / 100

ext4 / rw / 500

xfs / rw / 500

btrfs / rw / 500

zfs / rw / 500

ext4 / rw / 2000

xfs / rw / 2000

btrfs / rw / 2000

zfs / rw / 2000

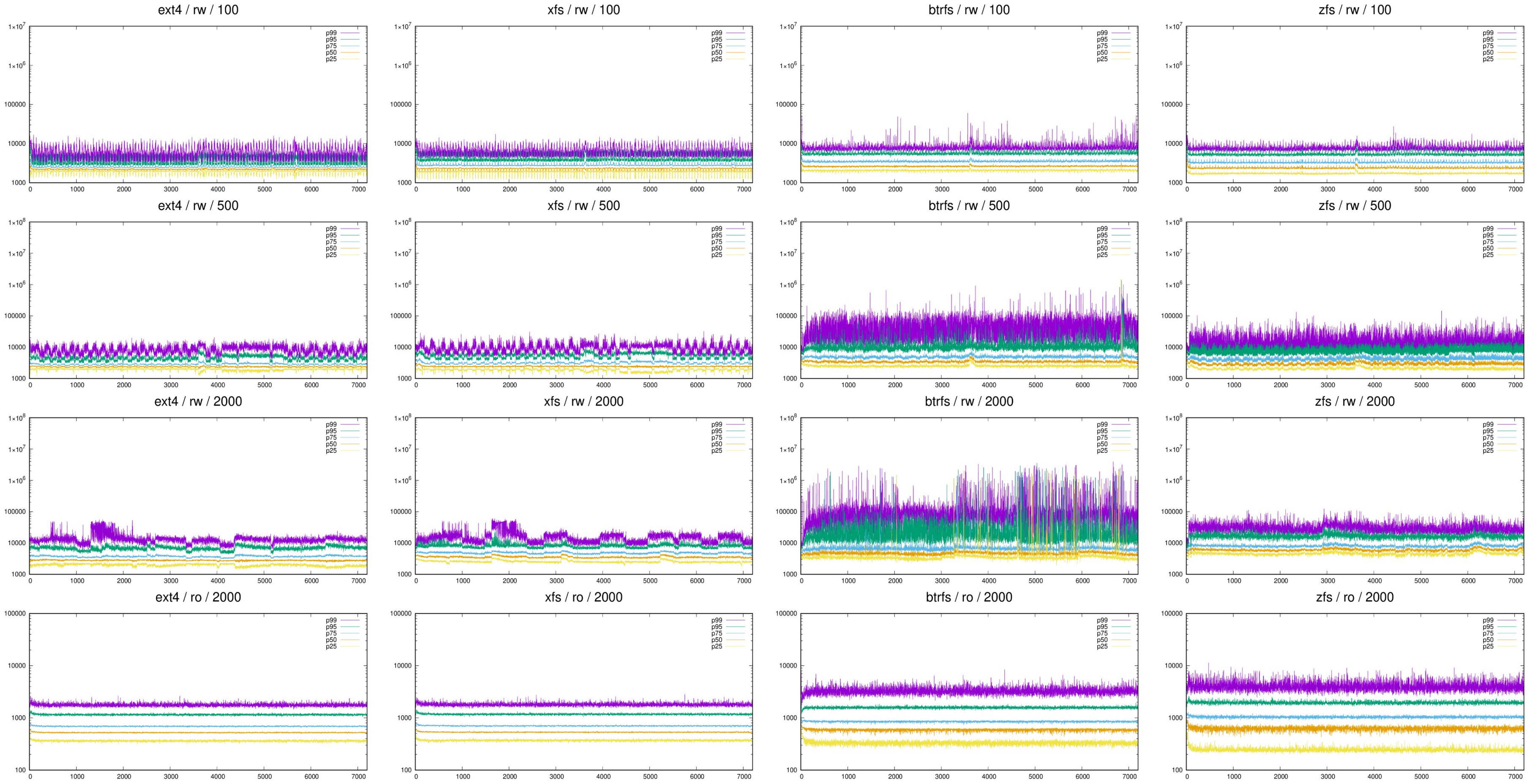
ext4 / ro / 2000

xfs / ro / 2000

btrfs / ro / 2000

zfs / ro / 2000

latencies (i5 / SATA SSD)



ext4 / rw / 100

xfs / rw / 100

btrfs / rw / 100

zfs / rw / 100

ext4 / rw / 500

xfs / rw / 500

btrfs / rw / 500

zfs / rw / 500

ext4 / rw / 2000

xfs / rw / 2000

btrfs / rw / 2000

zfs / rw / 2000

ext4 / ro / 2000

xfs / ro / 2000

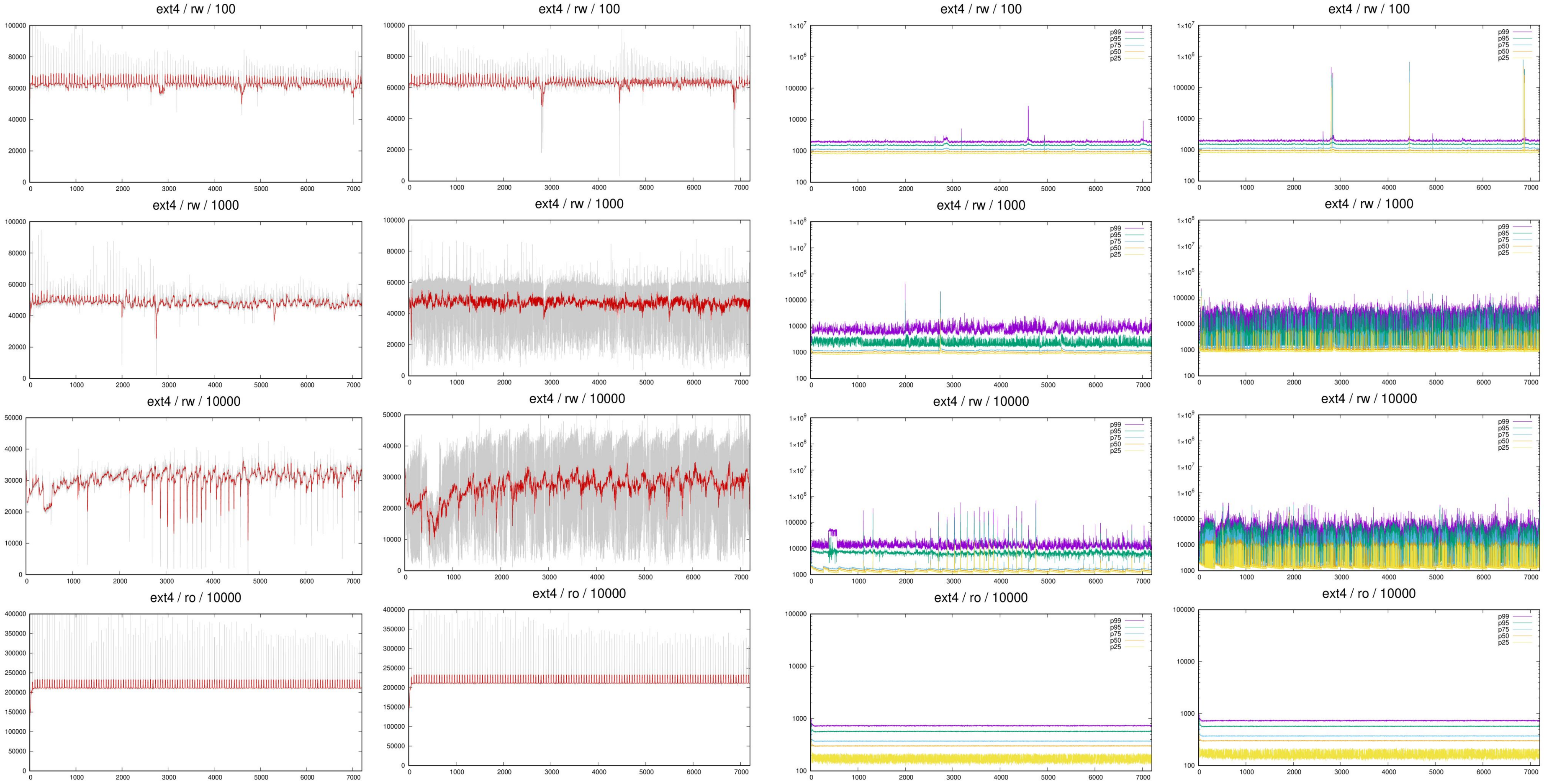
btrfs / ro / 2000

zfs / ro / 2000

More important ...

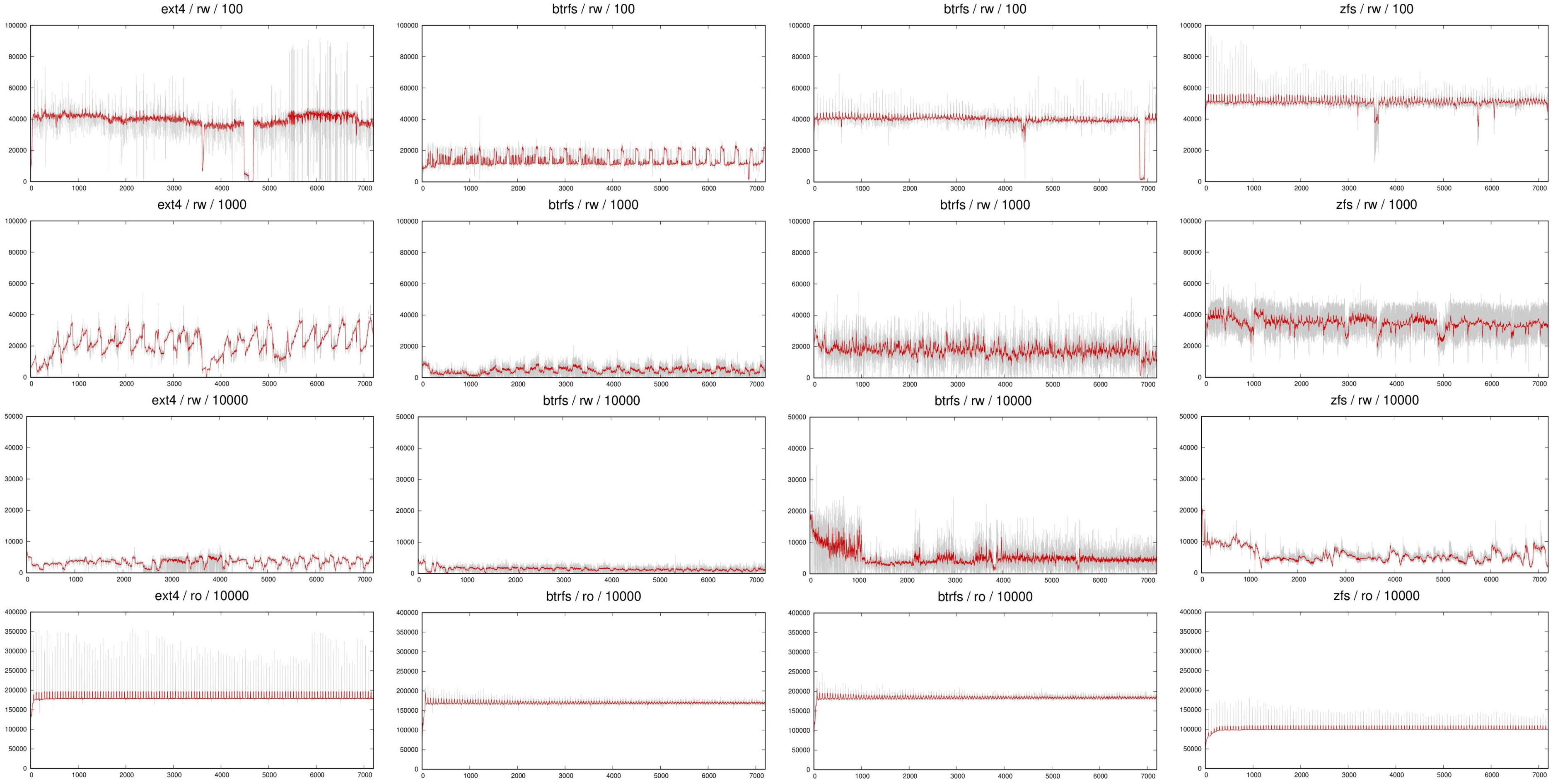
- dirty page cache (kernel)
  - evicted by OS, can cause spikes in latency
  - reduce vm.dirty_background_bytes / vm.dirty_expire_centisecs
  - and/or set backend_flush_after (disabled by default)
- full_page_writes (PG)
  - necessary on most file systems (zfs exception)
  - possible source of massive write amplification
  - maybe increase max_wal_size (but has drawbacks too)
- zfs prefetch (read-ahead)?
  - pg_dump durations ~2x higher than other filesystems

vm.dirty_background_bytes = 32MB vs. 1GB

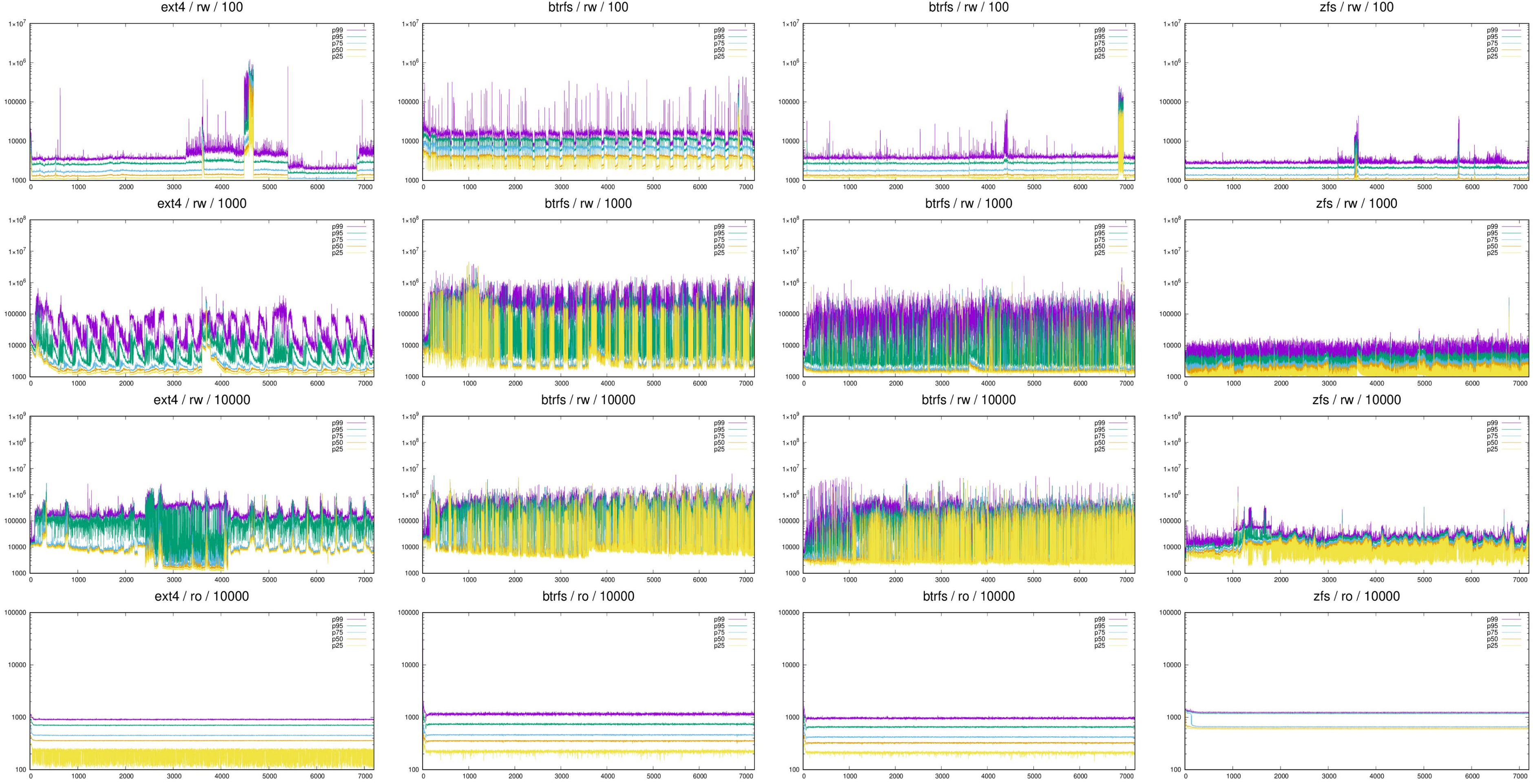


what about snapshots?

LVM LVM native native



LVM LVM native native



ext4 / rw / 100

btrfs / rw / 100

btrfs / rw / 100

zfs / rw / 100

ext4 / rw / 1000

btrfs / rw / 1000

btrfs / rw / 1000

zfs / rw / 1000

ext4 / rw / 10000

btrfs / rw / 10000

btrfs / rw / 10000

zfs / rw / 10000

ext4 / ro / 10000

btrfs / ro / 10000

btrfs / ro / 10000

zfs / ro / 10000

Questions

- how much more we could get from NVMe?
  - can we saturate NVMe for reads/writes?
  - not really, we're quite CPU heavy (cycles per I/O request)

- What Modern NVMe Storage Can Do, And How To Exploit It:
  High-Performance I/O for High-Performance Storage Engines
  Gabriel Haas, Viktor Leis, Technische Universität München
  https://www.vldb.org/pvldb/vol16/p2090-haas.pdf

Future tests

- different hardware
  - somewhat different patterns on old vs. new hardware
- what about many files?
  - large relations: 1TB relation is ~1000 files
  - 1 table -> multiple files (forks: data, vm, fsm), so many relations ...
  - there's caching, but ultimately it's up to the filesystem
- different workloads
  - OLTP is heavy on random I/O, but fairly simple
  - OLAP or mixed (OLTP + OLAP) workload

Q & A