Postgres vs. filesystems

Tomas Vondra
FOSDEM February 3-4, 2024
https://postgres.land/talks/
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)

https://postgres.land/talks/
Test cases

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

https://postgres.land/talks/
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

https://postgres.land/talks/
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
Computers are like onions. Everything is layers built on layers, and every layer makes you cry.

#sysadmin
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 developer capacity, outside of 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, ...)

https://postgres.land/talks/
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)

https://postgres.land/talks/
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
rule #1 - 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)
- OK, regressions exist too ...

https://postgres.land/talks/
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.

;-)
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 (COPY into table)

i5 / pgbench init / scale 2000
6x SATA SSD (RAID)

- LVM / btrfs: 270 seconds
- LVM / ext4: 288 seconds
- LVM / xfs: 279 seconds
- LVM / zfs: 418 seconds
- btrfs / native: 262 seconds
- native / zfs: 412 seconds

no snapshots

xen / pgbench init / scale 10000
NVMe SSD

- LVM snapshots / btrfs: 396 seconds
- LVM snapshots / ext4: 502 seconds
- LVM snapshots / xfs: 529 seconds
- LVM snapshots / zfs: 425 seconds
- native snapshots / btrfs: 268 seconds
- native snapshots / zfs: 411 seconds

no snapshots

https://postgres.land/talks/
Bulk load (COPY into table)

i5 / pgbench init / scale 2000
6x SATA SSD (RAID0)

- LVM / btrfs: 270 seconds
- LVM / ext4: 288 seconds
- LVM / xfs: 279 seconds
- LVM / zfs: 418 seconds
- btrfs / native: 262 seconds
- native / zfs: 412 seconds

LVM snapshots
- LVM snapshots / btrfs: 396 seconds
- LVM snapshots / ext4: 502 seconds
- LVM snapshots / xfs: 529 seconds
- LVM snapshots / zfs: 422 seconds

native snapshots / btrfs: 266 seconds
native snapshots / zfs: 411 seconds

xeon / pgbench init / scale 10000
NVMe SSD

- single / btrfs: 1291 seconds
- single / ext4: 1366 seconds
- single / xfs: 1348 seconds
- single / zfs: 1715 seconds

LVM snapshots
- LVM snapshots / btrfs: 3644 seconds
- LVM snapshots / ext4: 5520 seconds
- LVM snapshots / xfs: 6767 seconds

native snapshots / btrfs: 1269 seconds
native snapshots / zfs: 1728 seconds

https://postgres.land/talks/
Bulk load (COPY into table)

i5 / pgbench init / scale 2000
6x SATA SSD (RAID0)

- LVM / btrfs: 270 seconds
- LVM / ext4: 289 seconds
- LVM / xfs: 279 seconds
- LVM / zfs: 418 seconds
- btrfs / native: 262 seconds
- native / xfs: 412 seconds
- native / zfs: 412 seconds
- LVM snapshots / btrfs: 396 seconds
- LVM snapshots / ext4: 502 seconds
- LVM snapshots / xfs: 529 seconds
- LVM snapshots / zfs: 425 seconds
- native snapshots / btrfs: 266 seconds
- native snapshots / zfs: 411 seconds

xeon / pgbench init / scale 10000
NVMe SSD

- single / btrfs: 1291 seconds
- single / ext4: 1366 seconds
- single / xfs: 1346 seconds
- single / zfs: 1715 seconds
- LVM snapshots / btrfs: 3644 seconds
- LVM snapshots / ext4: 5520 seconds
- LVM snapshots / xfs: 6767 seconds
- LVM snapshots / zfs: 1269 seconds
- native snapshots / btrfs: 1728 seconds
- native snapshots / zfs: 411 seconds

built-in snapshots
OLTP (pgbench, read-only)

i5 / read-only / scale 2000 (~30GB)
i5-2500k / 16GB RAM / 6x SATA Intel SSD (RAID0)

- btrfs: 50626
- ext4: 53010
- xfs: 55714
- zfs: 38857

xeon / read-only / scale 10000 (~150GB)
2x E5-2620v4 / 64GB RAM / WD Ultrastar DC SN640 960GB (NVMe)

- btrfs: 137424
- ext4: 215207
- xfs: 214640
- zfs: 127407

https://postgres.land/talks/
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)

https://postgres.land/talks/
But throughput does not tell the whole story ...
Latencies (xeon / NVMe)

- ext4
- xfs
- btrfs
- zfs

Small

Medium

Large

Large / rw
latencies (i5 / SATA SSD)
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

https://postgres.land/talks/
vm.dirty_background_bytes = 32MB vs. 1GB
what about snapshots?
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 / configuration
  ○ different behaviors on old vs. new hardware
  ○ LVM vs. mdraid + LVM

• what about many files?
  ○ large relations: 1TB relation is ~1000 files, partitioning
  ○ caching, but max_files_per_backend = 1000 (=> syscalls)

• different workloads
  ○ OLTP is heavy on random I/O, but fairly simple
  ○ OLAP or mixed (OLTP + OLAP) workload

https://postgres.land/talks/
Q & A