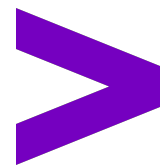
PostgreSQL AIO in der Praxis

Christoph Mönch-Tegeder
c.moench-tegeder@accenture.com

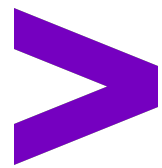
Accenture Technology

2026-04-21



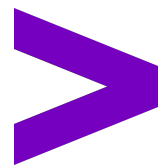
Über Mich

- PostgreSQL seit Version 7
- Berater, Trainer, ...
- Digital Core – Data Engineering bei Accenture
  - Datenbanken, Architektur, Konzepte, ...



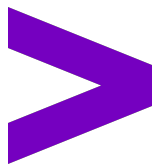
Accenture

- bekannt aus Funk und Fernsehen



Accenture

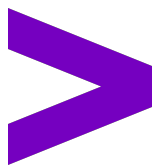
- bekannt aus Funk und Fernsehen
- weltweit: 786000 Mitarbeiter, US$70 Mrd Umsatz (FY2025)
- Bereiche:
  - Strategy&Consulting
  - Technology
  - Operations
  - Industry X
  - Song



Wir haben Elefanten

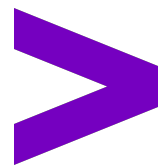
„Accenture Surfing Elephant“

PostgreSQL und IO



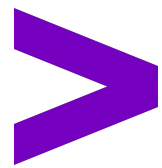
IO für (relationale) Datenbanken: Heap und Index

- Single Block Random Reads
- Index- und Heap-Blocks – OLTP
- Optimal: Selten – (fast) alle Blöcke im Shared Buffer
- „Nicht genug RAM für Working Set“
- `pg_prewarm` lädt Buffer beim Start



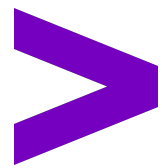
IO für (relationale) Datenbanken: WAL

- Sequential Synchronous Writes
- Transaktionslog: erlaubt Crash-Recovery, Replikation
- Muss bei Commit sicher auf Storage geschrieben sein
- Low Latency IOPS



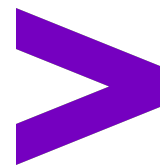
IO für (relationale) Datenbanken: WAL

- Sequential Synchronous Writes
- Transaktionslog: erlaubt Crash-Recovery, Replikation
- Muss bei Commit sicher auf Storage geschrieben sein
- Low Latency IOPS
- Seit PostgreSQL 7.1 – Release 2001-04-13



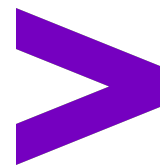
IO für (relationale) Datenbanken: Writes

- Random Block Asynchronous Writes
- Checkpointer, Background Writer, Backends
- OS-Buffered, periodische Syncs
- Writes durch Backend vermeiden



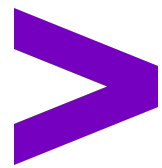
IO für (relationale) Datenbanken: Writes

- Random Block Asynchronous Writes
- Checkpointer, Background Writer, Backends
- OS-Buffered, periodische Syncs
- Writes durch Backend vermeiden
- Moderne Optimierungen finden Filesystem-Bugs



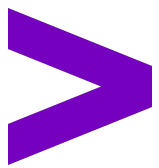
IO für (relationale) Datenbanken: Writes

- Commit Log
- Notify Queue
- Temp Files

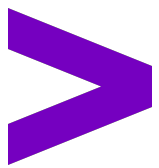


IO für (relationale) Datenbanken: Reads

- (Quasi-) Sequentielle Block Reads
  - Sequential Reads, Bitmap Index/Heap Scans
  - Besonders bei Data Warehouses und Analytics (Aggregate)
  - Ursprünglich gleicher Pfad wie Single-Block-Reads
  - OS-Level Read-Ahead hilft
- `effective_io_concurrency` – „Voranmeldung“
  - Seit PostgreSQL 8.4
  - Bis PostgreSQL 17 nur für Bitmap Heap Scans
  - nutzt `posix_fadvise()` – „Wunschmaschine“

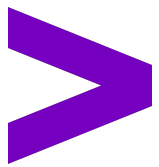


IO – Das Problem



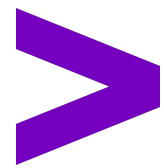
Moderne Systeme können mehr

- Storage bearbeitet Requests parallel
- Multi-Queue-IO-Scheduler (`mq_deadline` et al)
- Ping-Pong zwischen User- und Kernelspace nicht optimal
- Auch bei Readahead muss Buffer noch „gelesen“ werden
- Zeit, die das Backend in Read-Requests steht, könnte besser genutzt werden



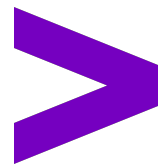
Streaming IO

- Seit PostgreSQL 17: Streaming IO
- Zugriffe auf Blöcke in einer Operation zusammenfassen
- Konfiguration: `io_combine_limit`
- Benutzt weiterhin `posix_fadvise()`
- Abstrahiert IO-Funktionalität



Asynchronous IO

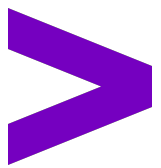
- Seit PostgreSQL 18: Asynchronous IO
- Varianten:
  - `sync` Kein AIO
  - `worker` (default) AIO wird auf Worker-Prozesse verteilt
  - `io_uring` Nutzt Linux `io_uring` Interface



Was ist mit POSIX AIO?

- Standardisiert - seit POSIX.1-2001
- In glibc in Userspace implementiert
- Nicht wirklich beliebt

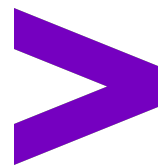
  AIO was always really really ugly[1]



[1] Linus Torvalds, 2016

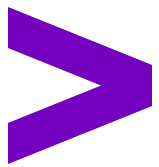
Wo?

- Sequential Scans
- Bitmap Heap Scans
- VACUUM/ANALYZE
- `pg_prewarm`
- *Nur Reads, keine Writes*



Konfiguration

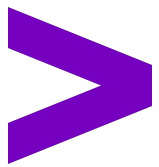
- `io_method`
  - Nur Linux: `io_uring`
  - Überall: `worker` – `io_workers` beachten
- `io_combine_limit` – User-Context Setting
- `io_max_combine_limit` – Serverseitiges Limit für `io_combine_limit`
- Defaults 128kB – höhere Werte können Latenz treiben
- Nicht jede Workload nutzt hohe IO-Size aus



Worker Monitoring – OS

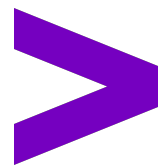
```
postgres  2988  2987  4 13:30 ?    00:00:18 postgres: 18/main: io worker 0
postgres  2989  2987  4 13:30 ?    00:00:18 postgres: 18/main: io worker 1
postgres  2990  2987  4 13:30 ?    00:00:18 postgres: 18/main: io worker 2
postgres  2991  2987  4 13:30 ?    00:00:18 postgres: 18/main: io worker 3
```

- IO-Worker sind markiert und numeriert
- Vom Postmaster beim Startup gestartet



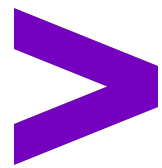
Monitoring – Database

- Neuer System-View `pg_aios`
- Eine Zeile pro IO-Handle
  - IO-Handle: Referenz auf IO-Operation
- Sehr schnellebig – ggf. Sampling
- „This view is mainly useful for developers of PostgreSQL“



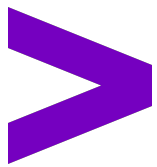
Worker Tuning

- Parameter `io_workers` (Default 3, Max 32)
- IO-Requests aller Backends werden via Workqueue über Worker verteilt
- Mehr Worker – mehr Durchsatz? mehr Overhead?
- `io_workers = 4`: ca. 1,5GB/s
  `io_workers = 8`: ca. 3,5GB/s
  spezifisches System, spezifischer Benchmark
- Automatisches, dynamisches Pool-Sizing in PostgreSQL 19



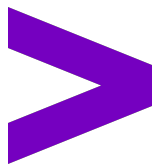
io_uring – Nie gehört

- Linux Only - seit Linux 5.1
- API für asynchrone IO-Operationen
- Ring-Buffer zwischen Kernel und Userspace
  - Ein Ring für Requests, ein Ring für Completion
  - *Input/Output User Ring*, get it?
- Effizienter als Daten kopieren
- Kernel bearbeitet Requests in beliebiger Reihenfolge



PostgreSQL und io_uring

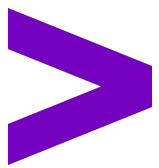
- Jedes Backend hat eigene `io_uring` Instanz
- Kein Tuning durch DBA nötig/möglich
- `io_uring` Instanzen werden von Postmaster angelegt
- Completions können von jedem Backend verarbeitet werden
- Hilfreich, wenn mehrere Sessions aktiv



Mixed Load und AIO

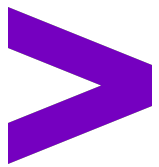
```
progress: 25.0 s, 992.4 tps, lat 5.508 ms stddev 2.434, 0 failed, lag 2.101 ms
progress: 30.0 s, 998.2 tps, lat 5.896 ms stddev 2.727, 0 failed, lag 2.420 ms
progress: 35.0 s, 855.6 tps, lat 70.723 ms stddev 166.874, 0 failed, lag 66.533 ms
progress: 40.0 s, 580.0 tps, lat 1929.590 ms stddev 618.390, 0 failed, lag 1922.699 ms
progress: 45.0 s, 469.6 tps, lat 3978.932 ms stddev 703.456, 0 failed, lag 3970.443 ms
progress: 50.0 s, 827.7 tps, lat 6491.492 ms stddev 247.106, 0 failed, lag 6486.655 ms
progress: 55.0 s, 1240.6 tps, lat 5720.047 ms stddev 354.645, 0 failed, lag 5716.834 ms
```

- Quiz: Wann ist AIO gestartet?



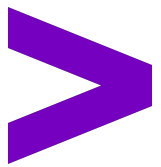
Mixed Load – OLTP vs OLAP

- AIO will IO möglichst auslasten – maximale Bandbreite
- Erhöht Latenz für parallel laufendes OLTP
- Konfiguration nur bei `io_mode = worker`
- Vorsicht bei Reports aus operativer Datenbank
- „In jeder großen Datenbank steckt ein kleines DWH – und will raus“



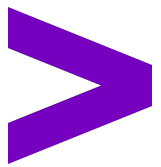
Vergleich Worker vs io_uring

- Worker
  - Portabel – überall verfügbar
  - Tuning benötigt (noch) für Durchsatz vs. Overhead
  - Höherer Overhead durch IPC
- `io_uring`
  - Linux Only – benötigt `liburing`
  - Kein Tuning nötig
  - Geringerer Overhead durch direkte Userland/Kernel Kommunikation

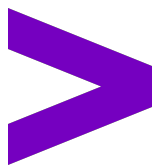


What's next?

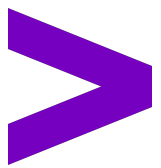
- IO-Scheduling mit `cgroup-v2` managen
- Scheduling-Controller für IOPS, Bandwidth, *Latency*
- Latency: „This is a cgroup v2 controller for IO workload protection.“
- Wie auf unterschiedliche Workloads filtern?
- Kompatibilität zwischen IO-Queue und Scheduling Controller
- Ein Thema für einen weiteren Vortrag



Fragen?



We Are Hiring!



Danke!