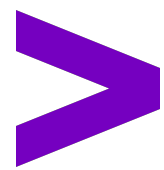
Was hält uns auf?

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

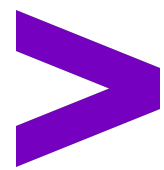
Accenture Technology

2023-06-30



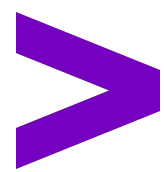
Über Mich

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



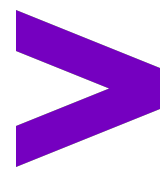
Accenture

- bekannt aus Funk und Fernsehen

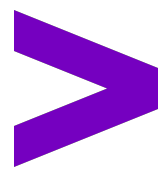


Accenture

- bekannt aus Funk und Fernsehen
- weltweit: 738000 Mitarbeiter, US$ 60 Mrd Umsatz
- Bereiche:
  - Strategy&Consulting
  - Song
  - Technology
  - Operations
  - Industry X

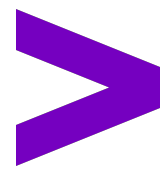


Warten



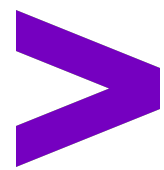
Performance

- Wenige CPU-Zyklen pro Operation
  - Resourcen ausser CPU?
- Interaktion zwischen Prozessen
  - System-Resourcen
  - Daten



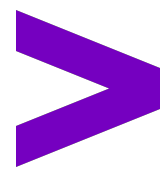
Warten

- User warten auf Interaktion
- Prozesse warten auf Daten
- Applikationen warten auf die Datenbank
- Worauf wartet die Datenbank?



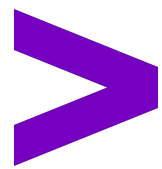
Warten

- Locks
- IO
- CPU
- sonstiges(*)



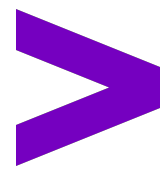
Wait Events

- Wait-Events in `pg_stat_activity`
- Neun Klassen, dutzende Events
  - nicht alle Performance-relevant
- Momentansicht
  - Kurze Events kaum sichtbar
  - Wartedauer?



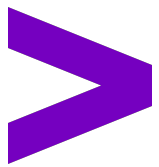
Wait Sampling

- Extension `pg_wait_sampling`
- Sampling: Zyklisches Auslesen der Wait-Events
  - Zyklus-Zeit `pg_wait_sampling.profile_period`
  - Vollständigkeit vs. Overhead
  - Ggf. an Gleichtakt-Verhalten denken



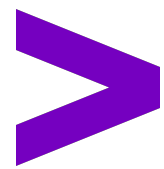
Beispiel: Wait Sampling

pid	event_type	event	queryid	count
4025	Activity	LogicalLauncherMain	0	9403
4019	Activity	CheckpointerMain	0	9403
4042	Client	ClientRead	0	9403
4023	Activity	AutoVacuumMain	0	9403
4022	Activity	WalWriterMain	0	9395
4020	Activity	BgWriterMain	0	7281
10651	Lock	transactionid	-1597659472875653566	3116
10647	Lock	transactionid	4208650773125423351	3113
10648	Lock	transactionid	-1597659472875653566	3100
10645	Lock	transactionid	-1597659472875653566	3093
10653	Lock	transactionid	4208650773125423351	3084
10646	Lock	transactionid	-1597659472875653566	3063
10649	Lock	transactionid	-1597659472875653566	3055
10652	Lock	transactionid	-1597659472875653566	3052
10650	Lock	transactionid	4208650773125423351	3051
10651	Lock	transactionid	4208650773125423351	3019



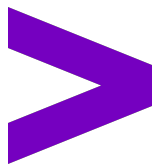
Wait Sampling

- Je mehr Zeit ein Prozess in einem Zustand verbringt, desto häufiger wird er dort gezählt
- seltene, kurze Events können durch das Raster fallen
- Inaktive Prozesse dienen zur Normierung
- Prozess-Laufzeit beachten
- Aggregieren oder PID unterdrücken: Sicht auf Gesamtsystem



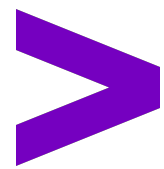
Beispiel: Wait Sampling

pid	event_type	event	queryid	count
4025	Activity	LogicalLauncherMain	0	9403
4019	Activity	CheckpointerMain	0	9403
4042	Client	ClientRead	0	9403
4023	Activity	AutoVacuumMain	0	9403
4022	Activity	WalWriterMain	0	9395
4020	Activity	BgWriterMain	0	7281
10651	Lock	transactionid	-1597659472875653566	3116
10647	Lock	transactionid	4208650773125423351	3113
10648	Lock	transactionid	-1597659472875653566	3100
10645	Lock	transactionid	-1597659472875653566	3093
10653	Lock	transactionid	4208650773125423351	3084
10646	Lock	transactionid	-1597659472875653566	3063
10649	Lock	transactionid	-1597659472875653566	3055
10652	Lock	transactionid	-1597659472875653566	3052
10650	Lock	transactionid	4208650773125423351	3051
10651	Lock	transactionid	4208650773125423351	3019



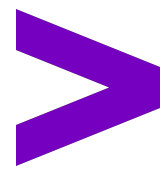
Beispiel: Wait Sampling

- Hier: 30% der Zeit im Lock-Wait
- Erinnerung: *transactionid*-Locks sind Row-Locks.
  - pgbench Scalefactor 1, 10 Clients
  - mit Scalefactor 100: Lock-Wait-Time <3%



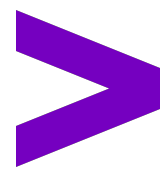
Exkurs: Locks

- Locks werden bis zum Ende der Transaktion gehalten
- `BEGIN`, `UPDATE`, Wochenende!
- Applikation hält Transaktion lange offen
  - the bad: viel Application-Time während DB-Transaktion
  - the ugly: Application-Interlocking mit Row Locks
- langlaufende Transaktionen (DDL!)
  - `CREATE INDEX CONCURRENTLY`



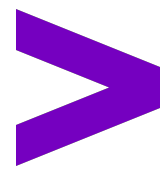
Monitoring Lock Waits

- `log_lock_waits` – sehr lange Wartezeit
- In `pg_locks` nach `granted = 'f'` schauen
- Für non-Fast-Path-Locks: Lockmanager wird gelockt
- Extension `pg_rowlocks` hilft bei Identifikation von Rowlocks



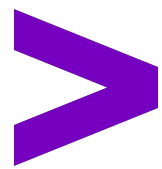
Umgang mit Lock Waits

- `idle_in_transaction_session_timeout` – für Extremfälle
  - Schneller: Prozesse terminieren, wenn möglich
- Opportunistisches Locking statt `SELECT FOR UPDATE`
- Application-Dev hilft, Contention zu finden und zu beheben
  - Geschicktes Schemadesign hilft
- Maintenance incl. DDL sorgfältig planen
  - Impact Assessment
  - Downtime einplanen



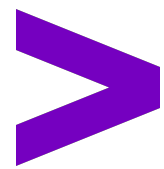
Replikation

- Synchrone Replikation wartet auf Bestätigung vom Standby
- Overhead schätzen: Netz-Roundtrip und Commit auf Standby
- Overhead messen: `pgbench`
  - Transaktionsrate und Clients vorgeben, Latenz beobachten
  - Vergleichen mit Standalone/Asynchrone Replikation
- Vorsicht wenn Ausfall
  - Auch bei Auto-Failover muss der Ausfall erstmal erkannt werden



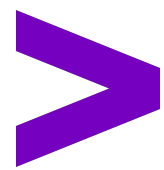
CPU

- Alle Cores beschäftigt
- Context Switching ist teuer: mehr Zeit im Kernel
  - `top`, `htop`: Pro Prozess und pro CPU
  - `mpstat`: Pro CPU
- Anzeige pro Intervall: Zykluszeit kurz wählen
- Zu langes Intervall verliert Informationen
- Load kann irreführend sein!



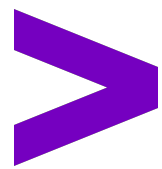
CPU beheben

- Code in der Datenbank?
- Komplexe Queries?
  - Auch für DBAs gilt: „Code ist nicht Dein Freund“
- Langsame Queries finden und optimieren
  - Extension `pg_stat_statements`
- Read-Offloading
- Analytics Offloading: Data Warehousing
  - Auf eigenen Systemen
  - Mit geeignetem Schema



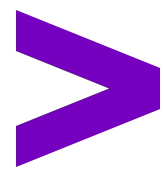
CPU

- Datenbank-CPUs skalieren schlechter als Appserver-CPUs
- OLTP-Systeme meist nicht CPU-begrenzt
- wenn doch:
  - Queries optimieren
  - Luxusproblem!



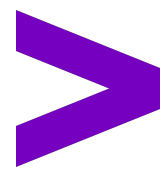
IO-Operationen

- Unterschiedliche Arten von I (lesen) und O (schreiben)
  - Manches IO stört mehr als anderes
- Durch verschiedene Subsysteme ausgelöst



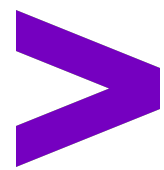
IO: Write Ahead Log

- Meist durch schreibende Transaktion (DML)
- Synchrone Writes, meist klein, Commit wartet
  - Latenz Matters
  - kann mit Hardware gelöst werden
- Benchmark: `fio`, Engine `sync`, Option `fdatasync`
- Bulk-Loads meist nicht WAL/Sync-limitiert



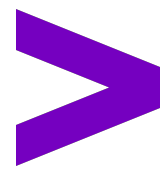
IO beobachten

- `iostat`: Sicht auf Devices (`w/s`, `await`)
- `sar`: Historische Daten, mäßige Auflösung
- `atop`: wie `top`, mit historischen Daten
  - Vorteil `atop`: Liest IO-Accounting (per-Process-Statistics)
- `blktrace`, `iowatcher` für das Blocklayer



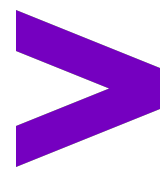
IO: Checkpoint

- Ausgelöst durch
  - Zeit (`checkpoint_timeout`, `starting: time`)
  - WAL-Menge (`max_wal_size`, `starting: xlog`)
  - Befehl/Backup (`starting: [immediate] force wait`)
- IO größtenteils sequentiell
- Gleichmäßiges Schreiben (`checkpoint_completion_target`)
- Hardware mit guten Durchsatz einsetzen



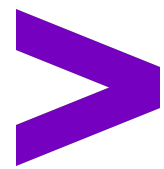
IO: Reads

- Während der Query: direkt spürbar
- Ziel: wenige Reads
  - Massenhafte parallele Reads besonders schmerzhaft
- Monitoring: `pg_statio_all_tables`
- Betriebssytem unterstützt Shared Buffer
- Abhilfe: mehr RAM und Buffer, Extension `pg_prewarm`



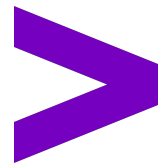
IO: Temp Files

- Spillover wenn `work_mem` nicht ausreichend
- Reads und Writes während der Query
- Indikator für große Datenmenge
- Unkoordinierte Zugriffe mehrer Backends
- Schwer vorhersagbare Laufzeiten



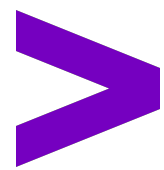
IO: Monitoring Temp Files

- Extension `pg_stat_statements`
- View `pg_stat_database`
- Logging: `log_temp_files`
- OS-Seitig IO beobachten:
  - `iostat`, `atop`, `sar`

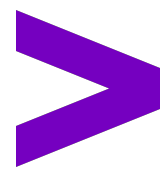


Internals & Low Level

- The Unknown Unknowns
- Naive Graphen helfen hier nicht
- Wo verbringen die Prozesse Zeit?
  - Beobachten: `perf`
  - Häufigkeit und Dauer einzelner Funktionen etc.: `bpftrace`
  - Syscalls auch mit `strace` (Overhead!)
- Benötigt Erfahrung und (oft) Source Code
- „Schwierige“ und „gute“ Systeme vergleichen
  - Rechtzeitig im Betrieb üben und dokumentieren

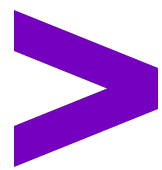


Fragen?



We Are Hiring!

- DBAs
- Consultants
- Software Developer
- Cloud Specialists
- Project Manager
- . . . and many more
- `https://www.accenture.com/de-de/careers`



Danke!