CYBERTEC

DATA SCIENCE & POSTGRESQL

Wie schaffe ich 1000 Applikationsuser gleichzeitig?

Laurenz Albe

Laurenz Albe

- PostgreSQL-Contributor seit 2006
- Autor von oracle_fdw u.a. Software im PostgreSQL-Umfeld
- Consultant
- Trainer

DATABASE- SERVICES

DATA Science

- Artificial Intelligence
- Machine Learning
- Deep Learning
- Big Data
- Business Intelligence
- Data Mining
- Etc.

PostgreSQL Services

- 24/7 Support
- High Availability
- Consulting
- Performance Tuning
- Clustering
- Migration
- Etc.

AUSTRIA (HQ)

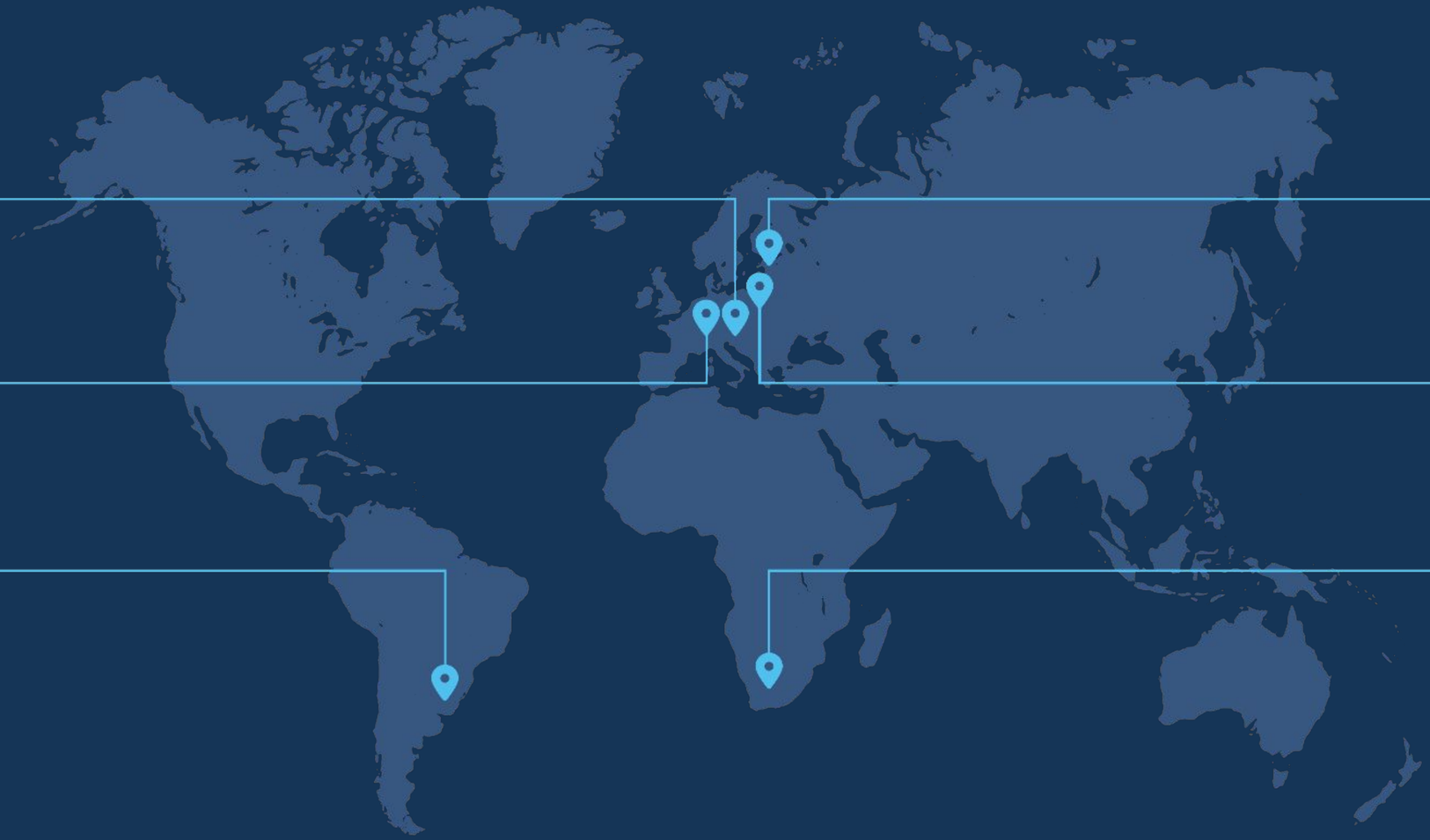
CYBERTEC POSTGRESQL
INTERNATIONAL (HQ)

SWITZERLAND

CYBERTEC POSTGRESQL
SWITZERLAND

URUGUAY

CYBERTEC POSTGRESQL
SOUTH AMERICA



ESTONIA

CYBERTEC POSTGRESQL
NORDIC

POLAND

CYBERTEC POSTGRESQL
POLAND

SOUTH AFRICA

CYBERTEC POSTGRESQL
SOUTH AFRICA

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

DATABASE - PRODUCTS

scalefield

CYPEX

CYBERTEC MIGRATOR

DATA MASKING

CYBERTEC PGEE

POSTGRESQL TDE

KUNDEN BRANCHEN

- ICT
- Universitäten
- Regierungen
- Automotive
- Industrie
- Handel
- Finanzwesen
- uvm.

Worum geht es in diesem Vortrag?

- Verwaltung von Datenbankverbindungen
- Connection Pooling
- Größenbestimmung von Connection Pools
- Fallen, die es zu vermeiden gilt

Die Problemstellung

Anforderungen der Applikation

- wir müssen 1000 Applikationsuser geichzeitig bedienen können
- wir brauchen viele Prozesse/Threads, um die Last abarbeiten zu können
- wir brauchen mehrere Instanzen des Applikationsservers, vielleicht auf mehreren Maschinen
- jeder Thread wird oft auf die Datenbank zugreifen

Der naive Zugang

Der naive Zugang

- jedesmal, wenn die Applikation etwas von der Datenbank braucht, öffnen wir eine Datenbankverbindung und schließen sie nachher wieder
- sehr schlechte Idee, denn das Herstellen einer Datenbankverbindung ist teuer:
  - startet einen Prozess auf dem Datenbankserver
  - lädt Katalogtabellen in einen Cache
  - authentisiert den Benutzer
- wenn die Datenbankanfragen kurz sind, kann mehr als die Hälfte der Ressourcen des Datenbankservers mit Verbindungsaufbau verbraucht werden

Ein Test des naiven Zuganges

pgbench mit persistenten Datenbankverbindungen:

```
$ pgbench -c 5 -T 60 test
tps = 4163.234087
```

pgbench öffnet für jede Anfrage eine neue Verbindung:

```
$ pgbench --connect -c 5 -T 60 test
tps = 446.301527
```

Sogar mit lokalen Verbindungen Leistungsreduktion von fast 90%!

Der „weniger naive” Zugang

Datenbankverbindungen offen halten

- jeder Thread in der Applikation hält seine Verbindungen zur Datenbank offen
- Datenbankverbindungen wiederverwenden statt schließen
- Idee: Verbindungen, die nichts tun, verbrauchen keine Ressourcen
- kann zu tausenden offenen Datenbankverbindungen führen

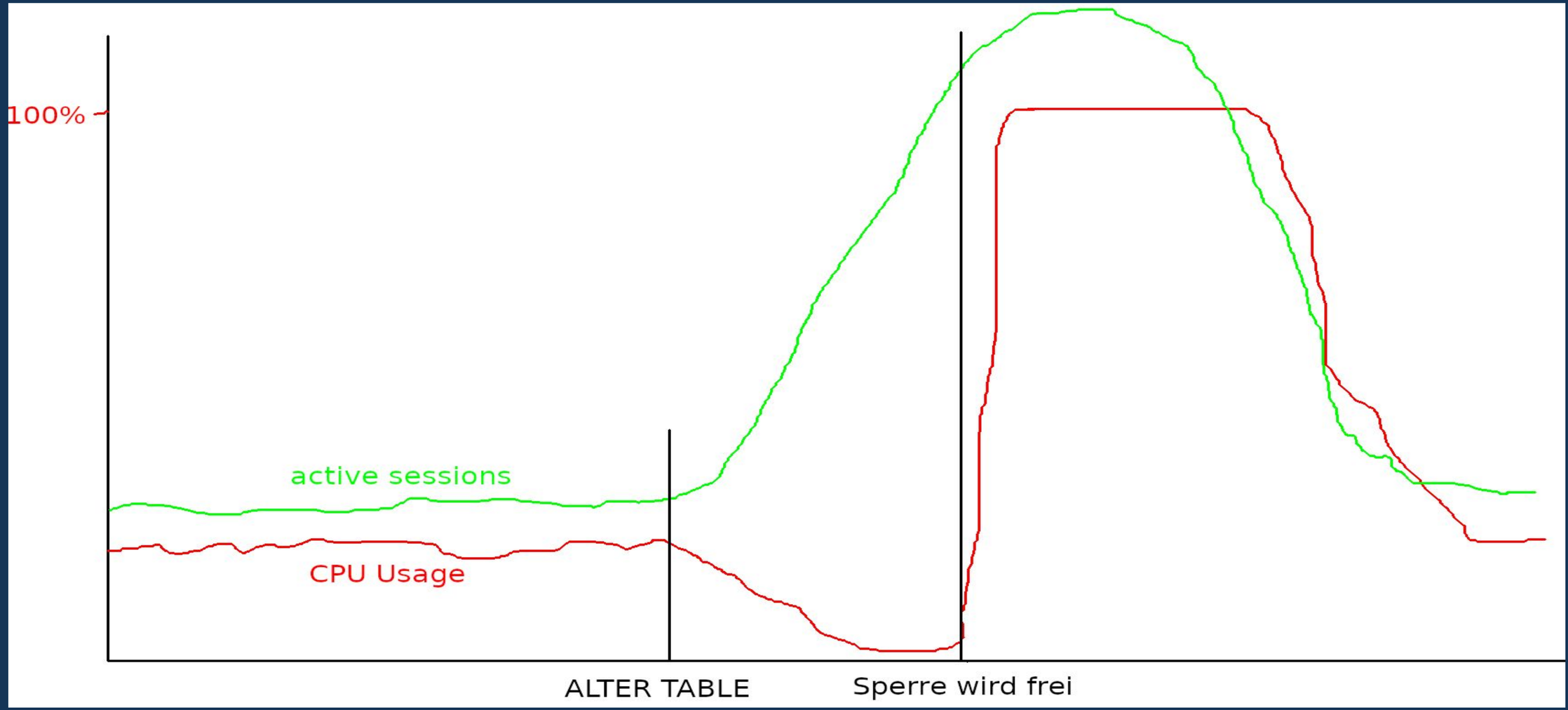
Probleme mit vielen offenen Sessions

- auch Datenbankverbindungen, die nichts tun, beeinträchtigen die Leistung:
  - bremsen den „Snapshot”, den jedes Statement erstellt
  - das wurde in v14 verbessert, ist aber immer noch relevant
- man kann in PostgreSQL die Anzahl der aktiven Datenbankverbindungen nicht einschränken
  - kann zur Überbelastung der Datenbank führen
  - das Risiko wächst mir der Anzahl der Verbindungen
- bei vielen aktiven Verbindungen muss `work_mem` niedrig sein, damit man nicht „Out Of Memory” gehen kann
  - das ist schlecht für die Verarbeitungsgeschwindigkeit

Gründe für Datenbank-Überlastung

- CPU ist am Limit
  - alle SQL-Befehle laufen viel langsamer
- Die Platte ist am Limit
  - alle SQL-Befehle laufen viel langsamer
  - I/O-„wait events” wie `WALSync`, `DataFileRead`, ...
- interne Ressourcenkonflikte in der Datenbank
  - „wait events” für interne „light-weight locks” wie `LockManager`, `BufferContent`, ...
- mit `pg_stat_activity` kann man `wait_events` überwachen

Beispiel für Datenbanküberlastung



(jede Ähnlichkeit mit bestehenden Kundensystemen ist zufällig)

Das Beispiel erklärt

- Verbindungen bleiben hinter einer `ACCESS EXCLUSIVE`-Sperre stecken
- „aktive” Verbindungen stauen sich
- irgendwann wird die Speere aufgehoben
- dann bricht die Lawine los ⇒ CPU-Spitze
- die Intuition sagt:
  viele aktive Verbindungen, also müssen wir mehr Verbindungen zulassen
- aber mehr Verbindungen verschlimmern das Problem weiter

Datenbanküberlastung mit Connection Pool bekämpfen

Verbindungszahl beschränken

- `max_connections` ist das Cluster-weite Limit
- `ALTER DATABASE|ROLE ... CONNECTION LIMIT n`
- Fehlermeldung, wenn die Grenze überschritten wird
  ⇒ keine benutzerfreundliche Lösung
- wir würden gerne die Anzahl der **aktiven** Verbindungen beschränken
  - das ist in PostgreSQL nicht möglich

Was ist ein Connection Pool?

- ein Stück Software, das eine bestimmte Anzahl an Datenbankverbindungen dauerhaft offen hält
- Client-Prozesse holen sich eine Verbindung aus dem Pool und geben sie nach Gebrauch zurück
- wenn alle Verbindungen vergeben sind, hängt der Client-Prozess
- typischerweise in den Applikationsserver eingebaut, kann aber auch ein eigenständiges Program sein (pgBouncer)

Vorteile eines Connection Pools

- Verbindungen werden offengehalten
  ⇒ keine Verschwendung durch Öffnen vieler Verbindungen
- ideale Einstellung Minimalgrösse = Maximalgrösse
- die Anzahl der aktiven Datenbankverbindungen ist beschränkt
  ⇒ verhindert Überlastung der Datenbank
- weniger Datenbankprozesse
  ⇒ mehr Ressourcen für jeden
  ⇒ weniger „Context Switches” in der CPU

Pool-Strategie „Session Pooling”

- Applikations-Thread hält die Verbindung, solange er existiert
- nützt nur dann, wenn Applikations-Threads kurzlebig sind
- keine Einschränkungen beim verwendeten SQL (der Status der Verbindung kann zurückgesetzt werden, wenn die Verbindung wieder in den Pool kommt)

Pool-Strategie „Statement Pooling”

- Verbindung kommt nach jedem Statement zurück in den Pool
- effektivste Methode, braucht wenige Verbindungen (verwendete Verbindungen sind nie „idle”)
- keine Transaktionen über mehrere Statements (außer in Datenbankfunktionen)
  ⇒ nur eingeschränkt brauchbar

Pool-Strategie „Transaction Pooling”

- Verbindung kommt nach jeder Transaktion zurück in den Pool
- der beste Kompromiss zwischen Brauchbarkeit und Effektivität
- kann nicht mit SQL-Konstrukten verwendet werden, die eine Transaktion überdauern

Connection Pooling: Einschränkungen

- Datenbankverbindungen kann nur für die selbe Datenbank und den selben Benutzer wiederverwendet werden
  ⇒ nur einen Datenbankbenutzer verwenden
- außer bei Session Pooling können Konstrukte, die länger als eine Transaktion leben, nicht verwendet werden:
  - temporäre Tabellen (Ersatz: `UNLOGGED` Tabellen)
  - `WITH HOLD` Cursors (Ersatz: `UNLOGGED` Tabellen)
  - Prepared Statements
- ein Ersatz für Prepared Statements könnnen PL/pgSQL-Funktionen sein, weil auch sie die Ausführungspläne von Statements aufbewahren

Pooling im Applikationsserver

- am effizientesten
- leicht zu verwenden (die meisten Applikationsserver und ORMs haben Support eingebaut)
- ist effektiv nur dann, wenn es einen einzigen Applikationsserver gibt
- bei mehreren Applikationsservern hat man mehrere Pools
  ⇒ Anzahl der aktiven Verbindungen nicht effektiv begrenzt

Connection Pooling mit pgBouncer

- https://www.pgbouncer.org/
- einfache Software, Proxy zwischen Client und Datenbank
- üblicherweise auf der Datenbankmaschine installiert (lokale Verbindungen)
- weniger Einstellungen als pgPool, aber einfacher und robuster
- verwenden, wenn es keinen eingebauten Connection Pool gibt oder dieser nicht effektiv ist (z.B. wenn es mehrere Applikationsserver gibt)

Größe des Connection Pools bestimmen

Das Problem der Größenbestimmung

- wenn der Pool zu klein ist, ist die Leistung schlecht
- wenn der Pool zu groß ist, ist die Datenbank überlastet und die Leistung leidet auch
- wird üblicherweise mit Versuch und Irrtum ermittelt
  (aber wir wollen es besser machen!)
- gute Lasttests helfen hier sehr

Grenzwerte für aktive Verbindungen

- die Grenzwerte werden normalerweise durch die Kapazität des Sekundärspeichers und der CPU gegeben
  - nicht mehr Verbindungen als CPU-Cores
  - nicht mehr als die Anzahl an parallelen Speicheroperationen, die der Sekundärspeicher verkraftet
- datenbankinterne Ressourcenkonflikte sind schwer abzuschätzen und werden hier ignoriert

Berücksichtigen von „idle in transaction”

- aus dem Pool vergebene Verbindungen sind nicht immer „active”
- vergebene Verbindungen, die nichts tun, verbrauchen keine Ressourcen
- wir müssen das folgende Verhältnis bestimmen:

$$\texttt{aktiv\_anteil} = \frac{\text{Verbindung ist aktiv}}{\text{Verbindung ist nicht im Pool}}$$

`aktiv_anteil` ermitteln

- mit den neuen Datenbankstatistiken in PostgreSQL v14 kann man den Wert leicht ermitteln:

```
SELECT active_time / (active_time + idle_in_transaction_time)
FROM pg_stat_database
WHERE datname = 'mydb'
AND active_time > 0;
```

- für eine grobe Abschätzung kann man mehrmals `pg_stat_activity` abfragen und das Verhältnis berechnen:

```
SELECT (count(*) FILTER (WHERE state = 'active'))::float8
       / count(*) FILTER (WHERE state in
                                ('active', 'idle in transaction'))
FROM pg_stat_activity;
```

Maximale Pool-Größe berechnen

$$\text{connections} < \frac{\min(\text{anzahl\_cores}, \text{max\_parallele\_ios})}{\text{aktiv\_anteil} \times \text{parallelismus}}$$

- „anzahl_cores” ist die Anzahl an CPU-Cores
- „max_parallele_ios” ist die Anzahl an gleichzeitigen Anforderungen, die der Sekundärspeicher verarbeiten kann
- „parallelismus” ist die durchschnittliche Anzahl an Serverprozessen, die eine einzelne Anfrage bearbeiten

Einfluß Abfrage-/Transaktionsdauer

- je kürzer die Statements, desto mehr kann man ausführen
- je mehr Statements man ausführen kann, desto mehr gleichzeitige Applikationsbenutzer kann das System verkraften
  ⇒ **Abfragen möglichst beschleunigen**
- „idle in transaction” Zeit möglichst kurz halten
  (sonst wird der Connection Pool nicht effektiv genutzt)
- lange Transaktionen möglichst vermeiden (das gilt immer)

Das Wichtigste in Kürze

Zusammenfassung

- man braucht einen einzigen Connection Pool
- wenn man das nicht von der Applikation bekommt ⇒ pgBouncer
- Connection Pool nicht zu groß einstellen
- die Maximalgröße des Connection Pools kann man berechnen
- kurze Statements/Transaktionen ⇒ viele gleichzeitige Benutzer

Fragen