Die PostgreSQL Performance Schnelldiagnose

Hans-Jürgen Schönig

www.postgresql-support.de

Performance Probleme diagnostizieren

Unser Ziel

- Finden der häufigsten Bottlenecks
- Lösen der wichtigsten Probleme
- Viele Probleme können mit wenigen Handgriffen gelöst werden.
- Diese Anleitung ist in keinster Weise vollständig!

Langsame Abfragen

pg_stat_statements: Queries tracken

- pg_stat_statements hilft, langsame Abfragen zu finden.
- Eines der wichtigsten Module überhaupt
- Sollte in jeder Database aktiviert sein

pg_stat_statements aktivieren

- postgresql.conf editieren:

“‘bash shared_preload_libraries = ‘pg_stat_statements’ -
PostgreSQL muss neu gestartet werden

- Die Extension installieren:

```
CREATE EXTENSION pg_stat_statements;
```

Wertvolle Information (1)


```
test=# \d pg_stat_statements
          View "public.pg_stat_statements"
       Column        |       Type        | Modifiers
---------------------+-------------------+-----------
...
 query               | text              |
 calls               | bigint            |
 total_time          | double precision  |
 min_time            | double precision  |
 max_time            | double precision  |
 mean_time           | double precision  |
 stddev_time         | double precision  |
```

Ausführungszeiten

- total_time sagt uns, wieviel Zeit eine Art von Query in Summe benötigt hat.
- Viele kurze Abfragen machen oft viel mehr Aufwand als wenige große Queries.
- Die Standardabweichung (stddev_time) sagt, wie gleichmäßig die Database antwortet.
- Eine hohe Standardabweichung kann viele Ursachen haben:
  - Ungleiche Verteilung der Daten
  - Probleme im Zusammenhang mit Locking

Wertvolle Information (2)


```
rows                | bigint |
shared_blks_hit     | bigint |
shared_blks_read    | bigint |
shared_blks_dirtied | bigint |
shared_blks_written | bigint |
local_blks_hit      | bigint |
local_blks_read     | bigint |
local_blks_dirtied  | bigint |
local_blks_written  | bigint |
```

Speichermanagement

- **blks*hit und **blks*read können zur Berechnung der Cache Hit Rate verwendet werden.
- Interessante Fragen:
  - Ist die Anzahl der Blöcke pro Query sinnvoll?
  - Ist die Cache Hit Rate brauchbar?
  - Werden viele lokale Buffer verwendet?
  - Gibt eine Query unnatürlich viele Zeilen zurück?

Wertvolle Information (3)


```
temp_blks_read       | bigint           |
temp_blks_written    | bigint           |
blk_read_time        | double precision |
blk_write_time       | double precision |
```

- Hohe temp_* Werte können auf falsche work_mem Settings oder fehlende Indices hinweisen

I/O Timing

- blk_*_time ist in den Default-Settings deaktiviert.
- track_io_timing in postgresql.conf kann eingeschaltet werden.
- Das Messen der Zeit kann etwas Overhead erzeugen.
- Um diesen Overhead zu bestimmen, gibt es ein Tool names pg_test_timing.

pg_test_timing

- Ergebnisse zwischen 19 und 1400 ns

```
iMac:~ hs$ pg_test_timing
Testing timing overhead for 3 seconds.
Per loop time including overhead: 39.18 nsec
Histogram of timing durations:
< usec   % of total      count
     1     96.14291   73610005
     2      3.85010    2947759
     4      0.00032        248
     8      0.00012         92
    16      0.00636       4866
    32      0.00016        123
```

Relevanz erzeugen

- pg_stat_statements sollte sortiert abgefragt werden.
- Idealerweise immer im Context:

```
SELECT  query, total_time, calls, sum(total_time) OVER ()
FROM    pg_stat_statements
ORDER BY total_time DESC
LIMIT 20;
```

- Ein Beispiel: http://www.cybertec.at/2015/10/pg_stat_statements-the-way-i-like-it/

Indexing . . .

Der übliche Bottleneck

- Indices haben bei der Optimierung das größte Potential
- Indices sind oft der einfachste Weg, die Performance zu verbessern
- Nichts wird so oft vergessen wie ein Index
- Nicht ist so “uncool” wie ein Index
  - Leute optimieren lieber RAID Level, Filesystem Settings, Kernel Parameter, Speicher, etc.

Was passieren kann

- Man stelle sich vor:
  - Wir haben eine Tabelle mit 60.000 Einträgen
  - Es fehlt ein einziger Index
  - Wir haben 1.000 Requests / Sekunde
- Das System liest 60.000.000 Zeilen vollkommen umsonst.
- Liest Du gerne das ganze Telefonbuch, um eine einzige Nummer zu finden?

Wie findet man das?

- Fehlenden Indices kann auf die Spur gekommen werden:
  - Durch das Auffinden langsamer Statements in pg_stat_statements
  - Durch geschicktes Auswerten von pg_stat_user_tables
- Oft sind die problematischen Spalten offensichtlich

Die wichtigste Query

- Die wichtigste Query eures Lebens:

```
SELECT  schemaname, relname, seq_scan, seq_tup_read,
    idx_scan, seq_tup_read / seq_scan AS avg
FROM    pg_stat_user_tables
WHERE   seq_scan > 0
ORDER BY seq_tup_read DESC
LIMIT 25;
```

Interpretation der Daten

- Wieso liest jemand 5 Millionen Zeilen 5 Millionen mal?
- Im "Problemfall" sieht man in seq_tup_read so etwas wie einen "Hockeystick"
- Es wird immer Sequential Scans geben
- Viele teure Scans sind das Problem
- Die Daten sind alle da
- Oft werden die Daten ignoriert oder falsch interpretiert

Zu viele Indexes (1)

- Auch zu viele Indexes sind ein Problem
- pg_stat_user_indexes hilft bei der Diagnose:

```
test=# \d pg_stat_user_indexes
View "pg_catalog.pg_stat_user_indexes"
    Column     |  Type   | Modifiers
---------------+---------+-----------
 schemaname    | name    |
 relname       | name    |
 indexrelname  | name    |
 idx_scan      | bigint  |
```

Zu viele Indexes (2)

- Zu viele Indexes sind viel “subtiler” als fehlende Indexes
- Bedenke: Schreibzugriffe müssen auch die Indexes updaten
- Indexes führen sehr oft zu Random I/O
- Random I/O ist teuer

Typische Probleme mit Abfragen

LIKE: Der klassische Killer

- LIKE Abfragen führen in vielen Anwendungen zu Sequential Scans
- Abfragen können sehr leicht beschleunigt werden:

```
CREATE EXTENSION pg_trgm;
CREATE INDEX idx ON tab USING gist (spalte gist_trgm_ops);
```

UNION vs. UNION ALL


```
SELECT 1 AS n UNION ALL SELECT 1;
 n
---
 1
 1
(2 rows)

test=# SELECT 1 AS n UNION SELECT 1;
 n
---
 1
(1 row)
```

Semantische Fehler

- Meistens ist es ein semantisches Problem, das als Performance Problem daher kommt.
- Bedenke: UNION filtert doppelte Einträge.
- Beachte: Kann es überhaupt doppelte Einträge geben?

I/O Performance

Schreibperformance

- Wer kennt diese Meldung?

```
checkpoints are occurring too frequently (%d second apart)
```

Checkpoints

- Checkpoints sind teuer
- Die Default Distanz zwischen zwei Checkpoints ist sehr sehr niedrig
- Höhere Checkpoint Distanzen beschleunigen Schreibzugriffe

Finally . . .

Contact us ...

Cybertec Schönig & Schönig GmbH
Gröhrmühlgasse 26
A-2700 Wiener Neustadt Austria

- More than 15 years of PostgreSQL experience:
  - Training
  - Consulting
  - 24x7 support