PostgreSQL - Troubleshooting

Bernd Helmle

24. August 2008

Agenda

1. Installation
2. Administration
3. Entwicklung
4. Backup

Installation (1)

- Installationspfade, Tablespaces und Mountpoints bestimmen
  - RAID-Level und LVM
  - Separates WAL-Laufwerk?

```
# initdb -X /mnt/wal
```

  - `temp_tablespaces`
  - Tablespace für Indexe
- Zu erwartende Größe von Tabellen und Indexe

```
tuple header: 24 Bytes
item pointer: 4  Bytes
data size   : 8  Bytes (2 x Integer)
----------------------
              34 Bytes
```

Installation (2)

- Welche Locales, welches Encoding?

```
# initdb -E UTF-8 --locale=de_DE.UTF-8
```

- Collation bestimmt Indexsortierung

```
#= INSERT INTO foo VALUES('ä'), ('Ü'), ('Ä'), ('Ö'), ('ö'), ('ü');

#= SELECT * FROM foo ORDER BY value DESC;
 value
-------
 ü
 ö
 ä
 Ü
 Ö
 Ä
(6 rows)
```

```
#= SELECT * FROM foo ORDER BY value DESC;
 value
-------
 Ü
 ü
 Ö
 ö
 Ä
 ä
(6 rows)
```

VACUUM (1)

Tabellen (Heap) fragmentieren

- Erstellen geeigneter VACUUM-Policy, Anwendungsabhängig
- Manuelles VACUUM, Autovacuum, Cost-based VACUUM
- pg_autovacuum für spezifische Tabellen
- Optimizerstatistiken müssen aktuell sein

VACUUM und ANALYZE unbedingt notwendig!

VACUUM (2)

Manuelles VACUUM

```
#= VACUUM ANALYZE VERBOSE;
...
INFO:  vacuuming "public.foo"
INFO:  "foo": found 0 removable, 6 nonremovable row versions in 1 pages
DETAIL:  0 dead row versions cannot be removed yet.
There were 0 unused item pointers.
1 pages contain useful free space.
0 pages are entirely empty.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO:  vacuuming "pg_toast.pg_toast_16385"
INFO:  index "pg_toast_16385_index" now contains 0 row versions in 1 pages
DETAIL:  0 index row versions were removed.
0 index pages have been deleted, 0 are currently reusable.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO:  "pg_toast_16385": found 0 removable, 0 nonremovable row versions in 0 pages
DETAIL:  0 dead row versions cannot be removed yet.
There were 0 unused item pointers.
0 pages contain useful free space.
0 pages are entirely empty.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO:  analyzing "public.foo"
INFO:  "foo": scanned 1 of 1 pages, containing 6 live rows and 0 dead rows; 6 rows in sample, 6 estimated
...
INFO:  free space map contains 55 pages in 61 relations
DETAIL:  A total of 976 page slots are in use (including overhead).
976 page slots are required to track all free space.
Current limits are:  204800 page slots, 1000 relations, using 1305 kB.
VACUUM
Time: 16,793 ms
```

VACUUM (3)

Autovacuum automatisiert diese Wartungsaufgabe

- Standardmässig aktiviert

```
autovacuum = on
track_counts = on
```

- Ab Version 8.3 mehrere “Worker” möglich

```
autovacuum_max_workers = 3
autovacuum_naptime = 60s
```

- Grenzwerte beachten:

```
vacuum threshold  = autovacuum_vacuum_threshold
                  + autovacuum_vacuum_scale_factor * number of tuples
analyze threshold = autovacuum_analyze_threshold
                  + autovacuum_analyze_scale_factor * number of tuples
```

- Überwachungsmöglichkeiten in 8.3 erheblich verbessert

VACUUM (4)

Cost-based VACUUM hilft, I/O-Ressourcen einzusparen

```
vacuum_cost_delay = 0 # disabled
vacuum_cost_limit = 200 # 200ms Delay
```

- Verlängerte VACUUM-Laufzeiten
- `vacuum_cost_limit` schwierig zu bestimmen
- `vacuum_cost_delay` sollte mit Bedacht gewählt werden

VACUUM (5)

- Spezielle Konfigurationen für spezifische Tabellen möglich

```
#= \d pg_autovacuum
   Table "pg_catalog.pg_autovacuum"
      Column      |  Type   | Modifiers
------------------+---------+-----------
 vacrelid         | oid     | not null
 enabled          | boolean | not null
 vac_base_thresh  | integer | not null
 vac_scale_factor | real    | not null
 anl_base_thresh  | integer | not null
 anl_scale_factor | real    | not null
 vac_cost_delay   | integer | not null
 vac_cost_limit   | integer | not null
```

- -1 setzt den Systemdefault
- **Hinweis für Backups:** pg_dump berücksichtigt diese Einstellungen nicht!

Free Space Map

Die Free Space Map speichert fragmentierten Bereich einer Tabelle oder Index.

- Re-Konfigurierbar nur bei Serverneustart, Shared Memory

```
max_fsm_pages = 204800
max_fsm_relations = 1000
```

- Zu kleine Maps verhindern die Wiederverwendung von Speicherplatz innerhalb von Tabellen oder Indexe
- Monitoring durch **VACUUM VERBOSE** oder **pg_freespacemap** (contrib-Modul)
- Vorraussichtlich in 8.4 flexiblere Infrastruktur (Relation Forks)

REINDEX

```
#= REINDEX DATABASE db;
```

- Indexe fragmentieren ebenfalls
- Regelmässiges REINDEX garantiert schlanke Indexe
- Wartungsfenster erforderlich

Überwachung (1)

- Überwachung von Statistiken über System-Views

```
   Schema   |           Name            |  Type   |  Owner
------------+---------------------------+---------+----------
 pg_catalog | pg_locks                  | Sicht   | postgres
 pg_catalog | pg_stat_activity          | Sicht   | postgres
 pg_catalog | pg_stat_user_indexes      | Sicht   | postgres
 pg_catalog | pg_stat_user_tables       | Sicht   | postgres
 pg_catalog | pg_statio_user_indexes    | Sicht   | postgres
 pg_catalog | pg_statio_user_sequences | Sicht   | postgres
 pg_catalog | pg_statio_user_tables     | Sicht   | postgres
```

- Statistische Daten wie Anzahl INSERTs oder I/O-Auslastung von Tabellen und Indexe in `pg_stat*`- und `pg_statio*`-Views
- `pg_stat_activity` und `pg_locks` protokollieren aktuelle Tasks auf der Datenbank und assoziierte Sperren

Überwachung (2)

Aktuelle Abfragen einer Datenbank

```
#= \x
Expanded display is on.
#= SELECT * FROM pg_stat_activity ;
-[ RECORD 1 ]-+---------------------------------
datid         | 16384
datname       | bernd
procpid       | 16351
usesysid      | 10
usename       | bernd
current_query | SELECT * FROM pg_stat_activity ;
waiting       | f
xact_start    | 2008-08-20 00:23:26.905504+02
query_start   | 2008-08-20 00:23:26.905504+02
backend_start | 2008-08-19 22:49:02.721159+02
client_addr   | 127.0.0.1
client_port   | 54952
```

Überwachung (3)

Sind Abfragen als **waiting=t** markiert, sollte eine Analyse über pg_locks erfolgen

```
#= SELECT * FROM pg_locks WHERE granted = 'f';
-[ RECORD 1 ]------+--------------
locktype           | transactionid
database           |
relation           |
page               |
tuple              |
virtualxid         |
transactionid      | 691
classid            |
objid              |
objsubid           |
virtualtransaction | 1/926
pid                | 20908
mode               | ShareLock
granted            | f
```

Überwachung (4)

Protokollieren problematischer Queries

```
log_min_duration_statement = 2min
log_min_error_statement = ERROR
```

Eckdaten der Datenbank erfassen

```
log_checkpoints = true
log_lock_waits = true
log_temp_files = 32768
log_autovacuum_min_duration = 10min
```

Überwachung (5)

```
EXPLAIN ANALYZE SELECT * FROM tenk1 t1, tenk2 t2
    WHERE t1.unique1 < 50 AND t1.unique2 = t2.unique2;

                                  QUERY PLAN
-------------------------------------------------------------------------------
 Nested Loop  (cost=0.00..327.02 rows=49 width=296)
              (actual time=1.181..29.822 rows=50 loops=1)
   ->  Index Scan using tenk1_unique1 on tenk1 t1
                    (cost=0.00..179.33 rows=49 width=148)
                    (actual time=0.630..8.917 rows=50 loops=1)
         Index Cond: (unique1 < 50)
   ->  Index Scan using tenk2_unique2 on tenk2 t2
                    (cost=0.00..3.01 rows=1 width=148)
                    (actual time=0.295..0.324 rows=1 loops=50)
         Index Cond: ("outer".unique2 = t2.unique2)
 Total runtime: 31.604 ms
```

Hausaufgaben

- Identifizieren von problematischen Queries (EXPLAIN, Sperren)
- Index-Nutzung
- Anpassen der VACUUM-Policy
- **VACUUM FULL** vermeiden
- **CLUSTER** und **REINDEX** in separate Wartungsfenster
- Größenüberwachung von Tabellen/Indexe, Tablespaces und Festplatten
- Logfile-Auswertungen ggf. notwendig

Rules vs. Trigger

- Trigger oftmals besser als Rules
- Rules sind “Makros”, ursprüngliche Query wird “umgeschrieben”
- Vorsicht bei Multi-Statements-Rules
- Volatile Funktionen werden u.U doppelt evaluiert

Server Side Prepared Statements

- Privat für jede Session
- Kann zu suboptimalen Plänen führen
- Sessioneigene Prepared Statements können über `pg_prepared_statements`

  angezeigt werden

pl/pgsql

- Leistungsfähige prozedurale Sprache für Stored Procedures
- Parser verbesserungswürdig
- Pläne für Abfragen werden innerhalb pl/pgsql gecached
- Vor 8.3 keine Plan-Invalidierung:

  relation with OID 1013156 does not exist

pg_dump

- Drei Formate verfügbar: plain (SQL), tar, custom
- pg_dump für Dumps spezifischer Datenbanken

```
pg_dump -Fc -E LATIN9 -U postgres yourdb
pg_dump -Fp -E LATIN9 -U postgres yourdb
```

- pg_dumpall für komplette Dumps (Sicherung kompletter Instanzen, Migration)

```
# globale Objekte
pg_dumpall -g -E LATIN9 -U postgres postgres
# alles
pg_dumpall -E LATIN9 -U postgres
```

- Problematisch bei sehr großen Datenbanken

WAL-based Backup

- Backups quasi “nebenher”
- Ideal für große Datenbanken mit mehreren hundert GigaByte oder mehr
- Base Backups über Snapshots (SAN, LVM)
- Wiederherstellung jedoch nur über Downtime
- Speicherverbrauch muß streng kontrolliert werden
- Einsatz von Hash Indexe erfordert Spezialbehandlung
- Tablespaces beachten!

Absicherung über Replikation

- DRBD + Heartbeat als Hochverfügbarkeitscluster
- Slony-I
- WAL-based StandBy
- "Scale-Out" erfordert meist umfangreiche Planung

"Replikation ist kein Backup"

PQfinish(lecture);