

PostgreSQL und memcached

Building a generic Query Cache

Björn Häuser

imos GmbH

07.12.2010 / PGDays Stuttgart 2010

Outline

- 1 Einführung
- 2 Cache-Hits
- 3 Cache-Maintenance

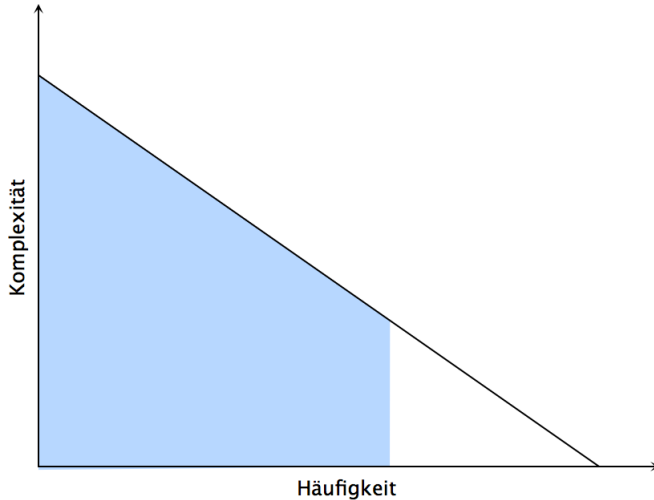
Einteilung von Caches

- Tradeoff zwischen
 - Maintenance
 - Komplexität der Anfragen
 - Dirty-Cache-Hit / Cache-Miss
- Transparenz
 - Beeinflussung des Programmieraufwandes

Zielsetzung

- Häufig auftretende Queries speichern
- Selbstständige Maintenance
- Cache-Miss
- Minimaler Einfluss auf Applikation
- Jederzeit zu- und abschaltbar

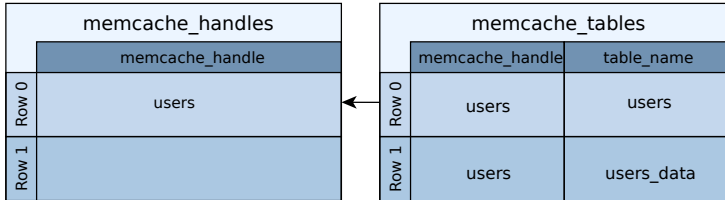
Komplexität vs. Häufigkeit



Grundlegende Arbeitsweise

- Tabellen werden nach Zugehörigkeit gruppiert
- Jede Gruppe bekommt einen eindeutigen Bezeichner: *Handle*
 - Beziehung zwischen Tabellen und *Handle* wird gespeichert
- Query wird mindestens einem *Handle* zugeordnet

Schema



Aufbau der Schlüssel

- Ein Handle speichert die aktuelle Revision
 - `$rev = "{$handle}_rev"`
- Cache-Key bestehend aus:
 - Query
 - Parameter
 - Revision des Handles

Beispiel

Query:

```
SELECT user_id, login, firstname, lastname  
FROM users  
WHERE user_id = ?
```

Handle:

```
users
```

Abfragen der Revision des Handles:

```
$rev = memcache_get ("{$handle}_rev");
```

Zusammenbauen des Keys:

```
$key = $handle . md5($query, implode(",", $params))  
      "_rev_" . $rev;
```

Cache-Maintenance

- Bei Änderungen an Tabellen müssen die zugeordneten *Handles* inkrementiert werden
- Folgen:
 - Alle zwischengespeicherten Queries werden automatisch ungültig
- On-Commit-Event-Problematik

On-Commit-Problematik

- Cache nur invalidieren wenn Transaktion *COMMITTED* wurde
- Cache und Datenbank synchronisieren
 - Race-Conditions

Maintenance per Trigger

- Reagiert auf Veränderungen der Tabelle (INSERT, UPDATE und DELETE)
- Zugeordnete Handles auslesen und inkrementieren
- Vorteile:
 - Einfach umzusetzen
 - Nur Abhängigkeit auf `pg_memcache`
- Nachteile:
 - Reset auch bei *ROLLBACK*
 - Race-Conditions

Maintenance per Trigger

```
FOR p_memcached_handle IN
  SELECT mt.memcached_handle
  FROM memcached_tables mt
  WHERE mot.table_name = TG_RELNAME
LOOP
  p_rev_name = p_memcached_handle .
              memcached_handle || '_rev';
  p_rev = memcache_get(p_rev_name);
  IF p_rev IS NOT NULL THEN
    PERFORM memcache_incr(p_rev_name);
  END IF;
END LOOP;
```

Maintenance per Trigger

- Race-Condition 1:
 - Transaktion **A** resettet *Handles*
 - Transaktion **B** hinterlegt alte Daten unter neuer Revision
 - Transaktion **A** *COMMITTED* seine Daten
 - Folge: Alte Daten liegen im Cache
- Race-Condition 2:
 - Transaktion **A** resettet *Handles*
 - Transaktion **B** hinterlegt alte Daten unter neuer Revision
 - Transaktion **A** zerstört die Transaktion und *ROLLBACKED*
 - Folge: Richtige Daten im Cache, aber unnötiger Cache-Miss

Maintenance per LISTEN / NOTIFY

- NOTIFY
 - Seit PostgreSQL 9.0 mit Payload
 - Wird erst beim *COMMIT* zugestellt
- Handler übernimmt das Inkrementieren der *Handles*
- Nachricht vs. Event transportieren
- Vorteile:
 - Keine Abhängigkeit in der Datenbank
 - Eine Nachricht kann mehreren Destinationen zugestellt werden
 - Gleiche Nachrichten werden zusammengefasst
- Nachteile:
 - Client muss entweder *POLL*en oder *SELECT*en
 - Kein Two-Phase-Commit möglich
- Poor-Mans-Queue?

Maintenance per LISTEN / NOTIFY

NOTIFY:

```
EXECUTE pg_notify(  
    TG_TABLE_SCHEMA,  
    (SELECT string_agg(memcached_handle, ','))  
    FROM memcached_tables  
    WHERE table_name = TG_TABLE_NAME)  
);
```

LISTEN:

http:

[//rhodiumtoad.org.uk/junk/listen-min.pl.txt](http://rhodiumtoad.org.uk/junk/listen-min.pl.txt)

Maintenance per LISTEN / NOTIFY

- Wartezeiten bis *NOTIFY* die Destination erreicht
- Wenig Kontrolle über die Nachricht

Maintenance per Queue

- Eleganteste Lösung
 - *PgQ*
- Nachricht vs. Event transportieren
- Consumer übernimmt das Inkrementieren der *Handles*

Maintenance per Queue

- Vorteile:
 - Volle Kontrolle über Nachrichten
 - Eine Nachricht kann mehreren Destinationen zugestellt werden
 - *PgQ* übernimmt die Verteilung
- Nachteile:
 - Externer Dienst (Ticker)
- Links:
 - <http://www.pgcon.org/2008/schedule/events/79.en.html>
 - http://wiki.postgresql.org/wiki/PGQ_Tutorial

Allgemein

- Probleme
 - Verdecken von langsamen Queries
- Memcached
 - Monitoring
 - <http://code.google.com/p/memcached/wiki/NewProgrammingFAQ>
 - Tuning
 - <http://code.google.com/p/memcached/wiki/NewUserInternals>
 - *Expire* setzen

Summary

- Leicht **integrierbare Lösung** um Antwortzeiten zu erhöhen
- **Wenig Einfluss** auf Programmierfluss
- Race-Conditions

- Zukunft
 - Race-Conditions minimieren
 - Umstellung auf Queueing-Lösung

- **Leave Feedback!**
 - `http://2010.pgday.eu/feedback`