Performance-Optimierung

Entwicklung von performanten Perl-Applikationen

Deutscher Perl-Workshop 9.0 / 2007 | http://www.perl-workshop.de/

Alvar C.H. Freude | http://alvar.a-blast.org/ | alvar@a-blast.org

Übersicht

Die nächste halbe Stunde werden wir uns das Folgende anschauen (wenn ich alles in der Zeit schaffe)

- Ein paar grundlegende Tipps und passende Beispiele
- Allgemeines zur Performance-Steigerung mit DBI
- Performance-Tipps für SQL
- Performance-Tipps für den Apache
- Meine Lösung für den c‘t Datenbank-Contest
- Ein paar Tipps für Web-Foren mit performanter Thread-Ansicht

Grundlegendes

* Das wichtigste überhaupt: **Macht Euch Gedanken!**
  Wer immer ein wenig mitdenkt, der schreibt auch performanteren Code
* Varianten Messen, z.B. mit *Benchmark* oder manuell mit *Time::HiRes*
* Profiling mit *Devel::DProf*, *Devel::Profiler*, *Devel::SmallProf*
* Wenig abzuarbeitender Code ist meistens schneller als viel Code

Perl Compilieren

* Ein paar Compiler-Switches beim Bauen von Perl beeinflussen die Performance
  * Generell: C-Compiler-Optimierungen und richtige CPU auswählen. Bei Perl ist der Unterschied zwischen -O2 und -O3 allerdings sehr gering
  * -Dusemymalloc
    Perls eigenes Malloc ist i.d.R. schneller als das System-Malloc, verbraucht aber u.U. (!) mehr RAM.

Performance-Engpässe und Renner in Perl

* Ganz oberflächlich lässt sich sagen:
  * ☺ Hashes sind sehr schnell
  * ☺ String-Verarbeitung und Reguläre Ausdrücke sind, richtig angewendet, sehr schnell
  * ☹ OO ist relativ langsam
  * ☹ Threads sind Mist, *use forks, no threads!*
  * ☹ Schlechter Code ist langsam ... ;-)

Beispiel

```perl
# my $summe = sum($n)
# Summe aller Zahlen von 1 bis $n
sub sum
    {
    my $end = shift;

    my $sum = 0;
    $sum += $_ for (1 .. $end);

    return $sum;
    }
```

Geht das schneller?

Wenn ja: Wieviel schneller?

Das Ende der Fahnenstange ist erst erreicht wenn ...

* Das Ende von Optimierungen ist erreicht, wenn nur noch Daten hin- und hergeschaufelt werden, so schnell wie sie der Speicher hergibt

* Aber: vielleicht lässt sich ja die Datenmenge reduzieren oder das Schaufeln vermeiden? ;-)

* Daher: auch mal nach den Algorithmen schauen!

* Das Beispiel der vorherigen Seite geht für $n=1000$ rund 240 mal schneller!

```perl
#
# Gauss'sche Methode
#

sub sum_nach_gauss
    {
    my $zahl = shift;

    return ($zahl * ($zahl + 1)) / 2;
    }
```

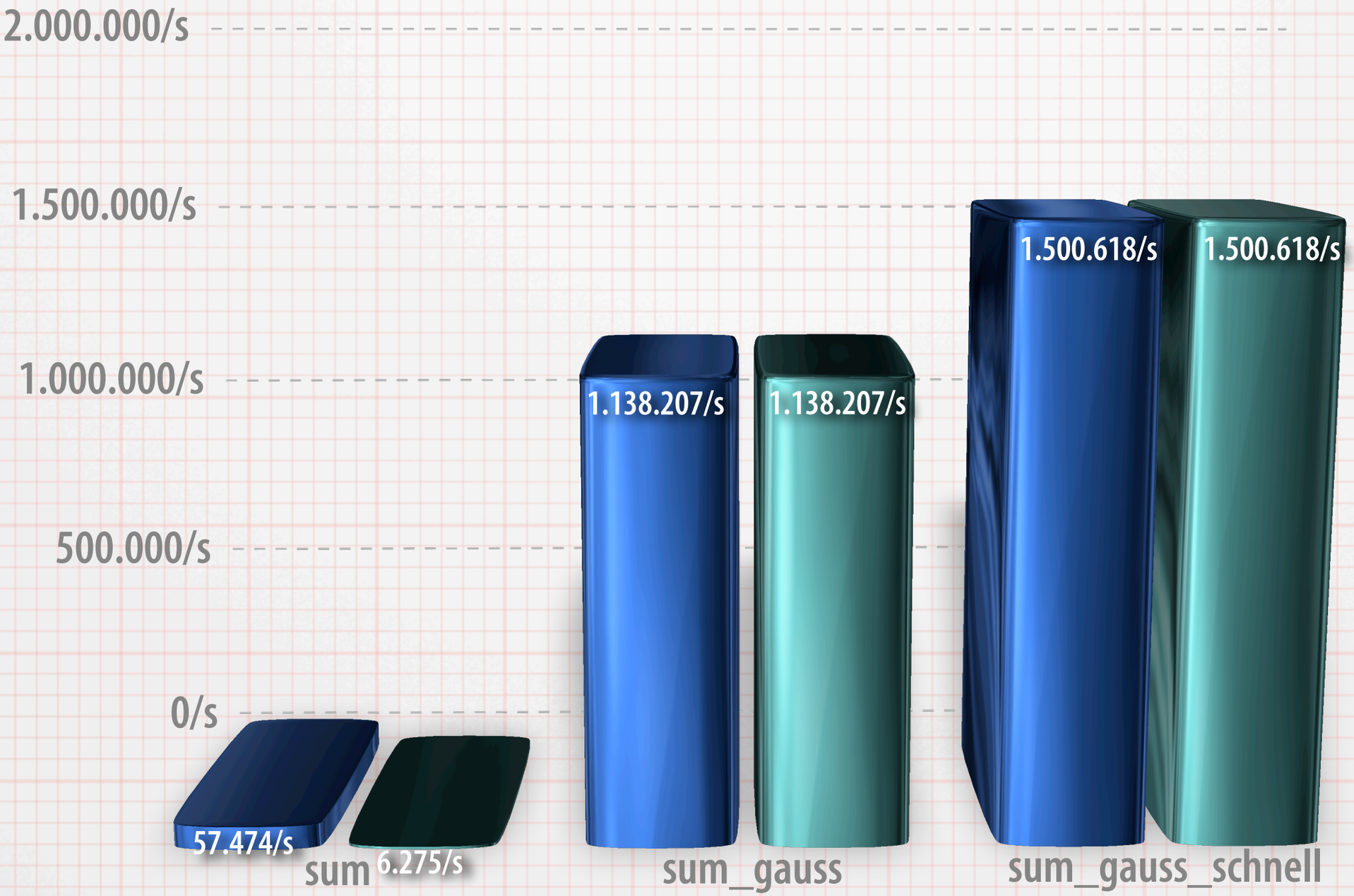
Anderer Algorithmus, ohne Schleife
Aber: gehts noch schneller?

```perl
#
# Gauss'sche Methode
# Und ohne Parameterkopie
# Nicht so schön, aber
# ca. 30% schneller
#

sub sum_nach_gauss_2
   {
($_[0] * ($_[0] + 1)) / 2;
}
```

- Das Kopieren von Variablen kann viel Zeit kosten, jeder Opcode kostet Zeit: Perl ist kein C-Compiler!
- Hier: auf Kosten der Übersichtlichkeit gibt es ca. 30% mehr Performance, das fehlende *return* bringt ca. 4%

Oder in bunt …


2.000.000/s
1.500.000/s
1.000.000/s
500.000/s
0/s
57.474/s
6.275/s
1.138.207/s
1.138.207/s
1.500.618/s
1.500.618/s
sum
sum_gauss
sum_gauss_schnell
1..100
1..1000

Beispiel Reguläre Ausdrücke

```
sub text_nach_html
    {
    my $text = shift;

    $text =~ s{\*(.*?)\*}{<strong>$1</strong>}msxg;
    $text =~ s{(?<=[^<])/(.*?)/}{<em>$1</em>}msxg;
    $text =~ s{"(.*?)"}{&raquo;$1&laquo;}msxg;
    $text =~ s{\n}{<br />}g;

    return $text;
    }
```

4 mal Regex – Idee: diese zusammenfassen?

* Hier sieht es so aus, als ob das mehrmalige Durchlaufen des Strings viel Zeit kostet
* Warum also nicht optimieren und (fast) alles in einen Regex?

Ist das schneller?

```
sub text_nach_html_schnell
    {
    my $text = shift;

    $text =~ s{([*/"])(.*?)\1}
        {
        my $symbol = $1;
        my $subtext = text_nach_html_schnell($2);
        if ($symbol eq '*')
            { "<strong>$subtext</strong>" }
        elsif ($symbol eq '/')
            { "<em>$subtext</em>" }
        elsif ($symbol eq '"')
            { "&raquo;$subtext&laquo;" }

        }msxge;

    $text =~ s{\n}{<br />}g;

    return $text;
    }
```

Ist das schneller? (2)

Zeichenklasse

```
sub text_nach_html_schnell
    {
    my $text = shift;

    $text =~ s{([*/"])(.*?)\1}
        {
        my $symbol = $1;
        my $subtext = text_nach_html_schnell($2);
        if ($symbol eq '*')
            { "<strong>$subtext</strong>" }
        elsif ($symbol eq '/')
            { "<em>$subtext</em>" }
        elsif ($symbol eq '"')
            { "&raquo;$subtext&laquo;" }

        }msxge;

    $text =~ s{\n}{<br />}g;

    return $text;
    }
```

Code fürs Ersetzen

Nein, ist nicht schneller!

- Es ist ca. 70% langsamer (je nach Text und -Länge!)
- Zeichenklassen, Rückwärtsreferenzen und Codeausführung machen es langsam
- Zudem Rekursion nötig, um innerhalb eines gefundenen Blockes wieder Textauszeichnungen vorzunehmen
- Also: Testen, und Vorsicht bei Cleverness ...
- Variante mit mehreren Einzel-Regexe schafft bei 1,3 kB Texten ca. 20000 pro Sekunde: 0,05 ms pro Text

Zusammenfassung von Teil 1

Relationen

* Einzelne Befehle bewegen sich im nanosekunden-Bereich
* Selbst Summenberechnung mit 1000er Schleife schafft über 6000 Berechnungen pro Sekunde: 0,16 ms dauert eine – Selbst das ist nicht wirklich zeitkritisch
* Regexe sind oft auch in Rudeln schnell, aber: Testen!
* Daher: bei Optimierung darauf konzentrieren, was wirklich Zeit kostet

Ja, und was kostet denn Zeit?

* Zum Beispiel Datenbank-Zugriffe: viele mehr als 1 ms
* Datenverarbeitung mit Modulen: bei Bedarf einfach mal nachmessen!
* Viele Methodenaufrufe bei OO
* sonstiges IO, eventuell Konfig-Dateien, XML-Bearbeitung
* irgendwelcher unnötiger Mist

Datenbanken, DBI, SQL

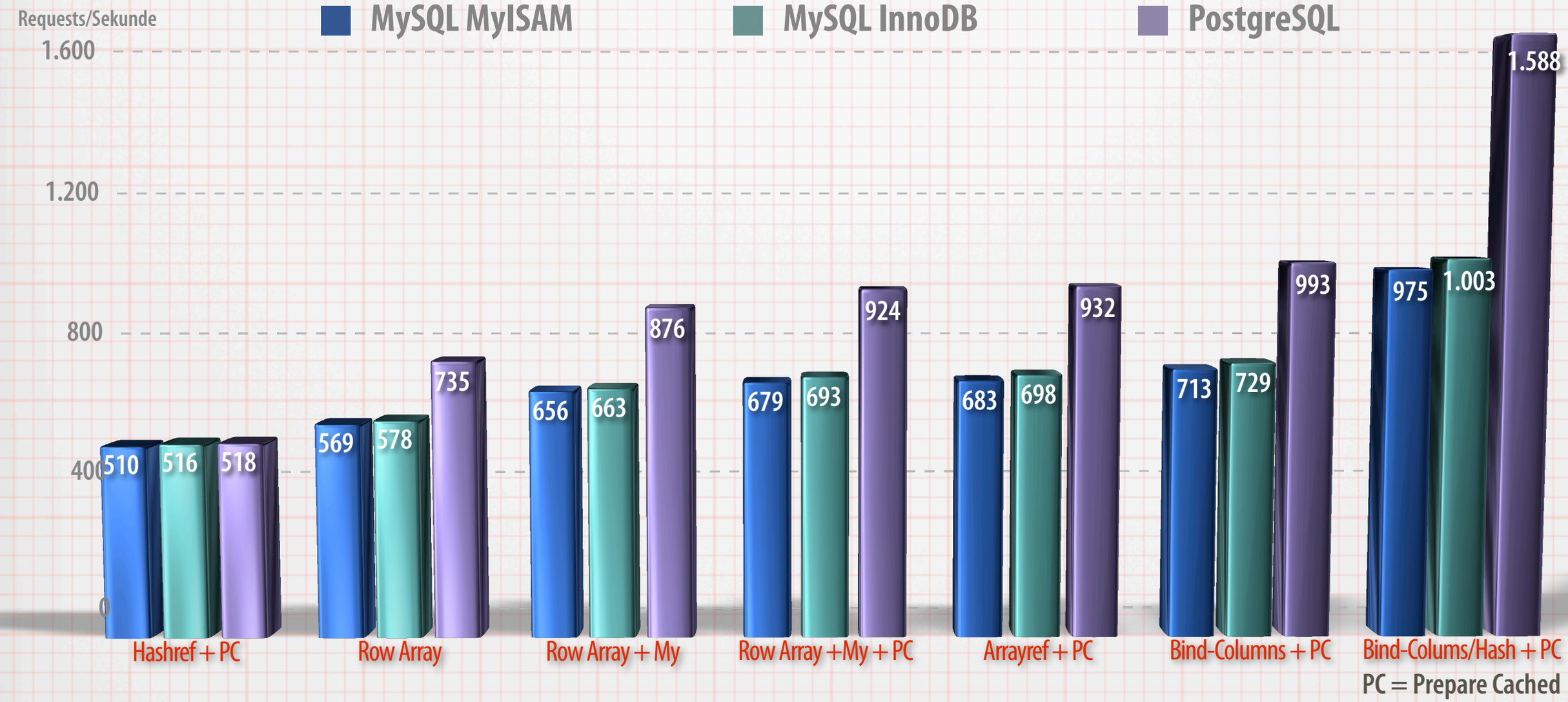
Ein kleiner Crash-Kurs für Performance-Optimierung mit DBI und SQL

Holen von Daten aus Datenbank

Was ist am schnellsten, um Daten zeilenweise in einen Hash einzulesen?

- ->fetchrow_array mit lokal definierten Variablen?
- ->fetchrow_array mit vor der Schleife definierten Variablen?
- ->fetchrow_arrayref? (->fetch), ->fetchrow_hashref?
- ->fetch mit bind_columns?
- Und was ist mit prepared_cached?
- Und was ist hierbei schneller: PostgreSQL oder MySQL?

Arrayrefs und Bind-Colums sind am schnellsten



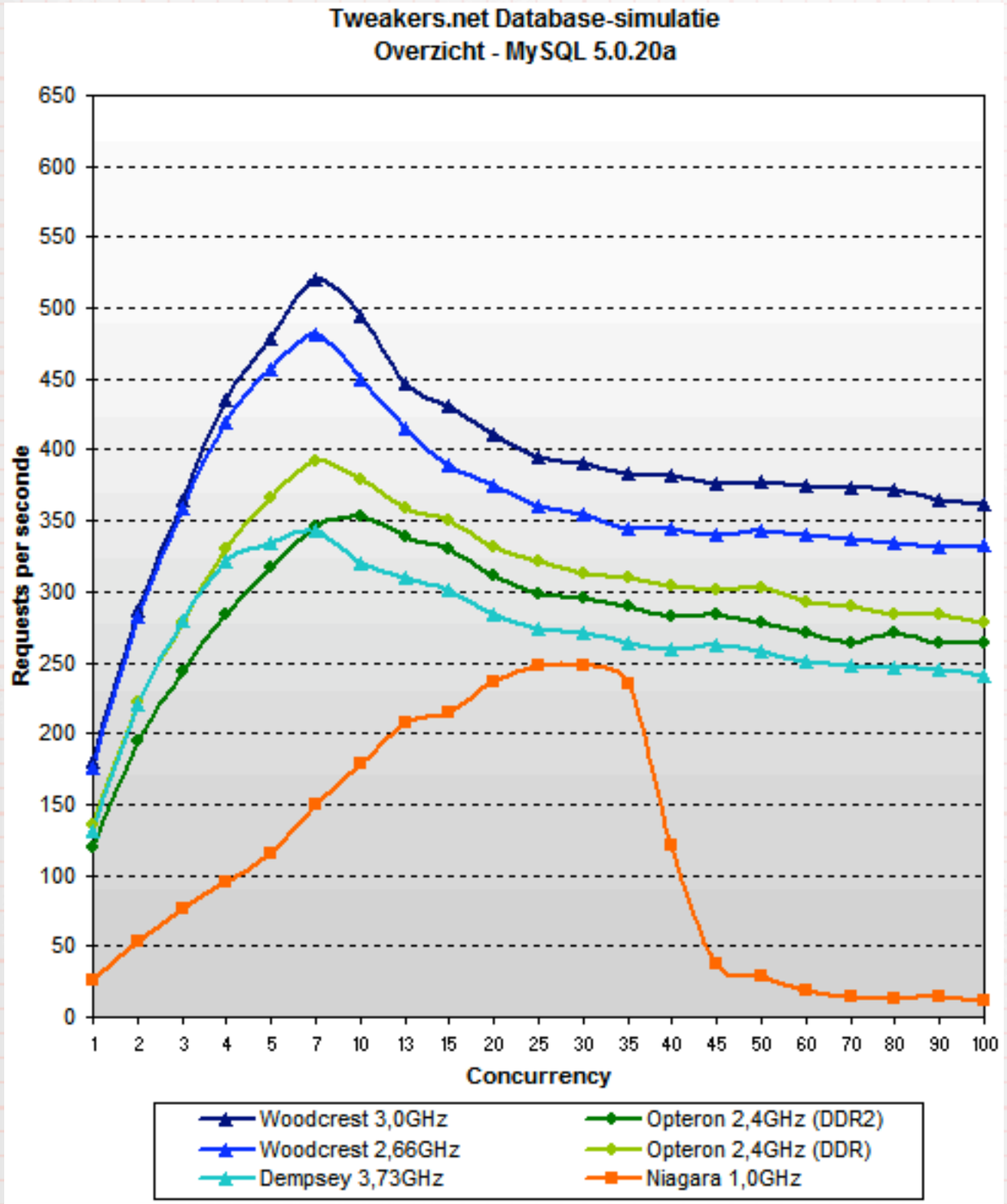
Auslesen von 100 Zeilen aus der DB, mehrere Text- und Integer-Spalten; alles aus Cache (immer die gleichen Zeilen), ohne Index-Nutzung. Pro Zeile in einen Hash eingelesen. Das Kopieren in den Hash kostet relativ viel Zeit, ohne dies ist Arrayref+Prepare Cached etwas schneller als die Bind-Columns: gut 1600 Requests/Sekunde

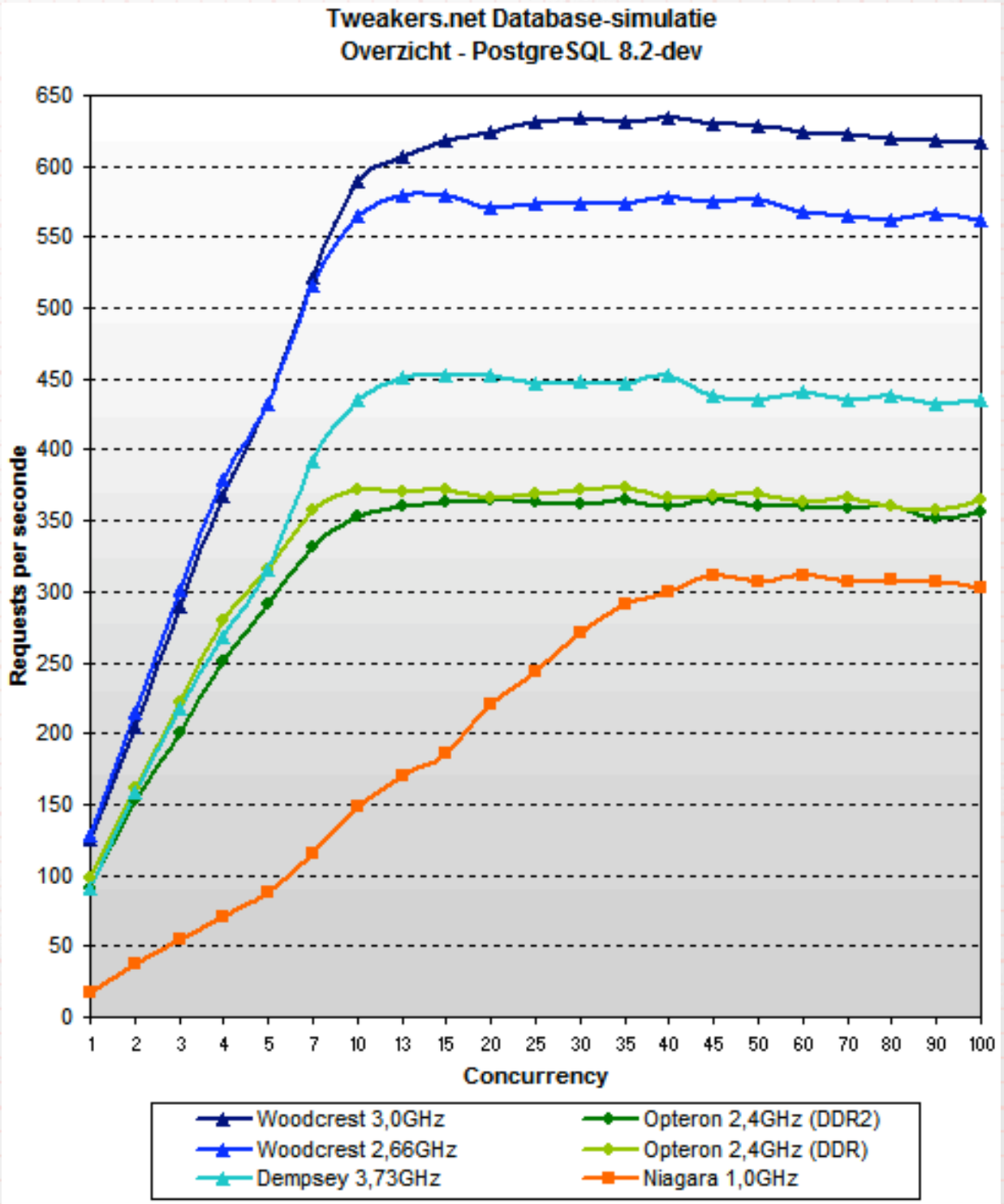
Am Rande: MySQL vs. PostgreSQL

* Dass ist PostgreSQL bei komplexen Aufgaben schneller als MySQL ist, das ist ja bekannt.
* Aber hier zeigt sich auch: es ist erstaunlicherweise ca. 50% schneller bei simplen Queries!
* Ein Kurztest zeigte auch:
  Bei einfachen Index-Zugriffen via Primary Key war es 70% schneller (alles aus dem RAM/Cache)
* Bei Bedarf sind natürlich angepasste Tests sinnvoll

MySQL skaliert schlecht!

Aus http://tweakers.net/reviews/657/6





Explain nutzen!

- Zur SQL-Optimierung:
  - EXPLAIN zeigt den Query-Plan
  - EXPLAIN ANALYSE zeigt auch Ausführungszeit (PgSQL)
- MySQL hat keinen echten Query Plan, braucht daher bei komplexen Sachen meistens mehr manuelles Feintuning
- Testen auch unter „echter“ Last: wenn die Caches belegt sind, ist das Verhalten oft anders

Queries zusammenfassen

* Möglichst wenig Queries machen: zusammenfassen!
* Wenn ein Webseitenaufruf 600 SQL-Queries hat, dann ist da was faul! (Log des RDBMS anschauen!)
* Aber auch möglichst einfache Queries machen
  * Oder schauen, dass das RDBMS einen guten Query-Plan erstellen kann und schnell ist
* Benchmarks! Geht auch mit *Time::HiRes*

Wieviel Queries brauchen wir hier?

Ahmet Iyidirli, SPD: Wahlkreis Berlin-Friedrichshain — Kreuzberg — Prenzlauer Berg Ost, Kandidat bei der Bundestagswahl 2005

http://www.wen-waehlen.de/kandidaten/ahmet-iyidirl_1268.html

Google

Wen soll ich wählen?

WEN WÄHLEN?

Ihre Plattform zur Kandidatensuche bei der Bundestagswahl 2005

Startseite | Kandidaten | Antworten | Über uns | Presse | Impressum

Wahlkreis: Berlin-Friedrichshain — Kreuzberg — Prenzlauer Berg Ost

Ahmet Iyidirli (SPD)

51 Jahre, Kinder: eine Tochter

+Zähler, wie oft aufgerufen

Weitere Kandidaten im Wahlkreis

- Christo Großmann (APPD)
- Norbert Hähnel (Die Partei)
- Stefan Liesegang (NPD)
- Christopher Paun (FDP)
- Cornelia Reinauer (Die Linke.)
- Figen Sanlik (BüSo)
- Michael Steinbach (HP)
- Hauke Stiewe (Parteilos)
- Hans-Christian Ströbele (GRÜNE)
- Kurt Wansner (CDU)
- Christian Wendt (Parteilos)

Vergleichen

Beruf

Derzeitiger Beruf: selbstständig
Ausbildung: Volkswirt

Politik

Die Interessen der folgenden Gruppen liegen mir *besonders* am Herzen (begrenzte Auswahl)

- Arbeitnehmer
- Arbeitslose
- Familien
- Alle Bürger

Folgende Werte und Ziele sind mir besonders wichtig (begrenzte Auswahl)

- Menschenwürde und Einhaltung der Menschenrechte

Ahmet Iyidirli
Sozialdemokratische Partei Deutschlands (SPD)

Kontaktdaten

ahmet@iyidirli.de

Einen!

```
SELECT * FROM kk WHERE id = 1268;
```

OK, es geht über einen View ...

Einen!

```
CREATE OR REPLACE VIEW kk AS        -- kk steht fuer kandidaten_komplett
    SELECT k.id AS id, k.titel, k.vorname, k.name AS name, geschlecht,
           p.kuerzel AS partei, p.name AS partei_lang, p.url AS partei_url,
           extract(years FROM age(geburtstag)) AS alter, geburtsort,
           ARRAY(SELECT details
                   FROM wahl_fragen f, wahl_antworten_kandidaten ak
                  WHERE f.id = ak.fragen_id AND ak.kandidat_id = k.id AND typ=1) AS gruppen,
           ARRAY(SELECT details
                   FROM wahl_fragen f, wahl_antworten_kandidaten ak
                  WHERE f.id = ak.fragen_id AND ak.kandidat_id = k.id AND typ=2) AS werte,
           ARRAY(SELECT ik.vorname || ' ' || ik.name || '>>' || ip.kuerzel || '>>' || ik.id
                   FROM wahl_kandidaten ik
              LEFT JOIN wahl_parteien ip ON ip.id = ik.partei_id
                  WHERE wahlkreis = k.wahlkreis
                    AND ik.id != k.id
                    AND offiziell = true
               ORDER BY ik.name) AS kandidaten,
           wahlkreis, w.name AS wahlkreis_name, get_bundesland(bundesland) AS wahlkreis_land
           listenplatz, land_liste, get_bundesland(land_liste) AS land_liste_name,
           f.width, f.height, bild_id, familienstand, kinder, ausbildung, beruf,
           aemter, nebentaet, ziele, warum_mich,  email, k.url, rss_url,
           strasse, plz, ort, telefon, fax, changes, visits, mailcount,
           wahl_kandidaten_count(k.id) AS counter
      FROM wahl_kandidaten AS k
 LEFT JOIN wahl_parteien p ON p.id = k.partei_id
 LEFT JOIN wahlkreise     w ON w.id = k.wahlkreis
 LEFT JOIN files          f ON f.id = k.bild_id
     WHERE offiziell = true;
```

Array-Subselects

Funktionsaufruf statt Join!

(via CASE WHEN)

Funktionsaufruf mit Update!

Der Query-Plan

```
ww=> EXPLAIN ANALYSE SELECT * FROM kk WHERE id = 1268;
                                                                                            QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop Left Join  (cost=0.00..143.68 rows=1 width=1094) (actual time=1.814..1.907 rows=1 loops=1)
   ->  Nested Loop Left Join  (cost=0.00..9.98 rows=1 width=1090) (actual time=0.156..0.242 rows=1 loops=1)
         ->  Nested Loop Left Join  (cost=0.00..6.10 rows=1 width=1068) (actual time=0.111..0.191 rows=1 loops=1)
               Join Filter: ("inner".id = "outer".partei_id)
               ->  Index Scan using wahl_kandidaten_pkey on wahl_kandidaten k  (cost=0.00..3.66 rows=1 width=997) (actual time=0.055..0.059
                     Index Cond: (id = 1268)
                     Filter: (offiziell = true)
               ->  Seq Scan on wahl_parteien p  (cost=0.00..1.36 rows=36 width=79) (actual time=0.008..0.054 rows=36 loops=1)
         ->  Index Scan using wahlkreise_pkey on wahlkreise w  (cost=0.00..3.86 rows=1 width=24) (actual time=0.022..0.026 rows=1 loops=1)
               Index Cond: (w.id = "outer".wahlkreis)
   ->  Index Scan using files_pkey on files f  (cost=0.00..3.61 rows=1 width=8) (actual time=0.015..0.020 rows=1 loops=1)
         Index Cond: (f.id = "outer".bild_id)
   SubPlan
     ->  Sort  (cost=11.72..11.88 rows=8 width=36) (actual time=0.468..0.476 rows=11 loops=1)
           Sort Key: ik.name
           ->  Hash Left Join  (cost=2.08..10.76 rows=8 width=36) (actual time=0.307..0.425 rows=11 loops=1)
                 Hash Cond: ("outer".partei_id = "inner".id)
                 ->  Index Scan using wahl_kandidaten_wahlkreis_idx on wahl_kandidaten ik  (cost=0.00..3.76 rows=8 width=30) (actual time=0.
                       Index Cond: (wahlkreis = $1)
                       Filter: ((id <> $0) AND (offiziell = true))
                 ->  Hash  (cost=1.36..1.36 rows=36 width=14) (actual time=0.124..0.124 rows=0 loops=1)
                       ->  Seq Scan on wahl_parteien ip  (cost=0.00..1.36 rows=36 width=14) (actual time=0.005..0.081 rows=36 loops=1)
     ->  Nested Loop  (cost=0.00..52.81 rows=9 width=81) (actual time=0.017..0.115 rows=4 loops=1)
           ->  Index Scan using wahl_fragen_typ_idx on wahl_fragen f  (cost=0.00..3.65 rows=12 width=85) (actual time=0.006..0.023 rows=11 l
                 Index Cond: (typ = 2)
           ->  Index Scan using wahl_antworten_kandidaten_kandidat_id_fragen_id_idx on wahl_antworten_kandidaten ak  (cost=0.00..4.07 rows=1
                 Index Cond: ((ak.kandidat_id = $0) AND ("outer".id = ak.fragen_id))
     ->  Nested Loop  (cost=0.00..65.21 rows=11 width=81) (actual time=0.043..0.177 rows=4 loops=1)
           ->  Index Scan using wahl_fragen_typ_idx on wahl_fragen f  (cost=0.00..3.77 rows=15 width=85) (actual time=0.015..0.040 rows=15 l
                 Index Cond: (typ = 1)
           ->  Index Scan using wahl_antworten_kandidaten_kandidat_id_fragen_id_idx on wahl_antworten_kandidaten ak  (cost=0.00..4.07 rows=1
                 Index Cond: ((ak.kandidat_id = $0) AND ("outer".id = ak.fragen_id))
 Total runtime: 2.418 ms
```

View macht Anwendung übersichtlich ...

* ... und schnell!
* SELECT name FROM kk WHERE id = 1

  ➡ Trotz View: Nur die nötige Spalte wird berechnet
* leicht erweiterbar
* Komplexität gut versteckt
* Alles an einer Stelle im RDBMS, nicht in der Applikation

Web-Applikationen

Nur ein paar Tipps

Web-Applikationen

* Der Aufbau einer Seite sollte im Normalfall nie länger als 100 ms dauern. Standard-Seiten lieber ca. 10 ms!
* Je häufiger eine Seite aufgerufen wird, desto schneller sollte sie erzeugt werden
* Wenn langsamer: Analysieren woran es liegt!
* Nicht nur bei leerer Datenbank die Performance testen!

mod_perl richtig nutzen

* Eigene Handler nutzen, nicht Apache::Registry-Scripts
* Beim Start alle Module laden
  * Dadurch werden die Module alle beim Apache-Start geladen und belegen nur ein mal Speicher
* persistente Datenbankverbindungen *(Apache::DBI)*
* Frontend/Backend-Lösung (Reverse Proxy) spart viel Speicher und gibt dicke Perl-Backends schnell frei: *mod_proxy* oder *Perlbal* (http://www.danga.com/perlbal/)

Caching-Strategien

(Beispiele)

- Caching-Strategien sind oft ein gutes Werkzeug zur Beschleunigung
- *mod_cache* für häufig benutzte Seiten, evtl. auch mit kurzen Zeitabständen
- In Applikation Cachen
  - globale Variablen/Datenstrukturen mit *mod_perl*
  - *memcached* für ändernde Daten (⇒Vortrag Wolfgang)

Beispiel: c‘t DB-Contest

Festplatte mit Caching schonen
Datenbankzugriffe zusammenfassen

Grundlegendes

- Nicht gewertete Lösung, da ich die Dokumentation vergessen hatte mitzuschicken und erst nachlieferte :-(
- Aufgabe: ein DVD-Shop nach Musterlösung optimieren
- Meine Lösung: Drei Varianten:
  - Unabhängig vom RDBMS normal
  - Unabhängig vom RDBMS mit Caching
  - PostgreSQL-Spezial-Version

Normale Lösung

- mod_perl
- Unabhängig vom RDBMS
- Im Wesentlichen 1:1 wie Original
- Aber ein paar SQL-Optimierungen
  - Zusammenfassung von Queries, Subselects
  - $db_h->prepare_cached
- PostgreSQL ca. 50% schneller als MySQL

```
SELECT c.customerid,
       (SELECT prod_ids
          FROM orders_full
         WHERE customerid = c.customerid
      ORDER BY orderid DESC LIMIT 1) AS history
  FROM customers c
 WHERE username=? AND password=?;
```

Queries mit Subselects zusammenfassen

Lösung mit Caching

- Viele Daten sind statisch und ändern sich nicht
  - Sie können daher beim Start vom Apache in eine Perl-Datenstruktur (Hash) geladen werden
    - Damit stehen sie nachher allen Apaches zur Verfügung
- Das entlastet die Datenbank, insbesondere bei der großen Version (ca. 1 GB)
- Da nur Hashzugriff: deutlich schneller als *memcached*

```perl
# ergibt eine Produkt-ID
my $prod_id  = $title_cache{$suche_titel};

# ergibt eine Arrayreferenz auf die Produkt-IDs
my $prod_ids = $actor_part_cache{$suche_schauspieler};

# ergibt den passenden Titel als String
my $title    = $prod_cache[$prod_id][TITLE];
```

Sehr einfacher und schneller Zugriff auf die Daten

Speicherbedarf

* Der Speicherverbrauch hält sich in Grenzen: unter OS X 10.4 braucht der gesamte Apache bei der mittleren Datenbank (100000 Produkte, DB-Größe 1 GB) rund 70 MB.
* Die gecachten Daten sind davon ca. 60 MB, der Rest Apache, mod_perl, compilierte Perl-Module usw.

Spezielle PostgreSQL-Version

* Die Cache-Version als Basis, dazu aber Optimierungen
* Eine Tabelle (cust_hist) war unnötig, die gleichen Informationen können mit einem Subselect aus anderen Tabellen geholt werden
* Eine andere Tabelle mit 1:n-Beziehung kann durch ein Array in der referenzierenden Tabelle ersetzt werden (PostgreSQL kann mehrdimensionale Arrays)

* Dadurch nur noch ein INSERT pro Bestellung, anstatt 1+anzahl_produkte*2
* Die gesamte Bestellung braucht nur einen INSERT und für jedes Produkt ein UPDATE (denn es ist eins weniger auf Lager) und ein SELECT auf die Benutzerdaten
* Der gesamte Bestellvorgang wird in einem PL/pgSQL-Funktion durchgeführt: vermindert DBI-Overhead
* Dadurch ist eine Bestellung sehr schnell

PL/pgSQL

```
CREATE OR REPLACE FUNCTION insert_order
            (
            customerid_in INTEGER,
            netamount_in  NUMERIC(8,2),
            tax_in        NUMERIC(8,2),
            prod_in       INTEGER[],
            quan_in       SMALLINT[]
            )
  RETURNS insert_order_return
       AS $$
DECLARE
   result insert_order_return;
   failed BOOLEAN DEFAULT 'f';
   num    INTEGER := 1;

BEGIN

   result.failed := '{}';

   INSERT INTO orders_full (orderdate, customerid, netamount, tax, prod_ids, quantity)
        VALUES (NOW(), customerid_in, netamount_in, tax_in, prod_in, quan_in);

   LOOP

      UPDATE inventory
         SET quan_in_stock  = quan_in_stock - quan_in[num],
             sales          = sales        + quan_in[num]
       WHERE prod_id        = prod_in[num]
         AND quan_in_stock >= quan_in[num];

      IF NOT found THEN
         result.failed := result.failed || prod_in[num];
         failed := 't';
      END IF;

      num := num + 1;
      EXIT WHEN prod_in[num] IS NULL;

   END LOOP;

   IF NOT failed THEN
      result.orderid := currval('orders_full_orderid_seq');
      SELECT INTO result.creditcardtype, result.creditcard, result.creditcardexpiration
                        creditcardtype,        creditcard,        creditcardexpiration
             FROM customers
            WHERE customerid = customerid_in;
   END IF;

   RETURN result;

END;
$$
LANGUAGE 'plpgsql';
```

Fazit

* Wenn Datenbanken weniger zu tun haben, wird alles schneller (achwas?)
* Weniger Queries meist schneller
* Schreibzugriffe zusammenfassen
* Möglichst viel im RDBMS machen
* Aufpassen: Swapping vermeinden
* Und was sagen denn nun die Bechmarks? …

Beispiel: Forum

Bäume in SQL ohne Rekursion: Genealogical Trees

Foren mit Thread-Ansicht brauchen baumartige Datenstrukturen

odi profanum vulgus et arceo von Marc Klinger

http://odem.org/informationsfreiheit/forum-view_482.html

Google

*Kommentare

- »Ich hasse den Pöbel und distanziere mich von ihm« (✉Bastian Pophal; 26.12 2004, 00:04:43)
  - Jura (Bernhard David Ströbele; 15.02 2007, 18:54:35)
  - Hallo! (Monika Ruszewski; 09.02 2007, 18:27:00)
    - Moinsen Moni! (Basti; 11.02 2007, 14:42:04)
  - Referat (Melanie Witte; 11.01 2007, 13:26:06)
    - Tips (Tim ; 10.02 2007, 11:11:34)
      - Verwechslung (Tim; 10.02 2007, 18:27:57)
  - Halbwissen (Amelie Wendt; 08.12 2006, 14:24:30)
    - Ohne Titel (Graf Tobler (Zürich, Genf und St. Moritz); 14.12 2006, 17:32:47)
      - Ohne Titel (Oliver Jenkowsky; 24.01 2007, 20:05:57)
        - Ohne Titel (Graf Tobler (Zürich, Genf und St. Moritz); 08.02 2007, 17:42:30)
          - Wissen und Bildung (Oliver Jenkowsky; 10.02 2007, 13:18:26)
            - Ohne Titel (Graf Tobler (Zürich, Genf und St. Moritz); 13.02 2007, 16:18:12)
      - Verunglimpfung sozialistischer Errungenschaften! (Wendelin Tornow; 09.01 2007, 11:05:20)
        - Ich hasse den Pöbel trotzdem.... (Bastian W. Pophal; 24.01 2007, 13:19:43)
          - @ Bastian Pophal (Wendel Holmes; 24.01 2007, 19:53:15)
            - Ohne Titel (Graf Tobler (Zürich, Genf und St. Moritz); 08.02 2007, 17:53:58)
              - Check? (Wendel Holmes; 09.02 2007, 18:33:49)
                - Ohne Titel (Graf Tobler (Zürich, Genf und St. Moritz); 13.02 2007, 16:20:48)
                  - Falsch (Wendel Holmes; 15.02 2007, 18:51:26)
                    - Ohne Titel (Graf Tobler (Zürich, Genf und St. Moritz); 16.02 2007, 12:40:57)
  - Ohne Titel (Hans Felbert; 30.11 2006, 16:02:24)

Aber SQL kann keine Trees!

- Üblicherweise wird dann der Baum rekursiv durchgegangen
- Das würde im Beispiel 22 SQL-Abfragen benötigen (für jedes Element des Baumes einen)
  - Wenn man die Kind-Elemente bei den Eltern mitzählt, kann man sich Queries sparen, wenn keine Kinder da sind.
    - Das Beispiel bräuchte aber immer noch 16 Queries

Und ohne Rekursion?

- Man denke an eine Liste mit Dateinamen inkl. Pfad im Dateisystem:

```
/etc/aliases        /etc/named.conf          /etc/X11/xkb/types
/etc/passwd         /etc/monthly             /etc/X11/xkb/types/basic
/etc/fstab          /etc/periodic            /etc/X11/xkb/types/caps
```

- Die Dateien sind auch als Baum geordnet
- Und alle lassen sich mit einer Abfrage finden:
  - `name LIKE '/etc%'`

Das Prinzip lässt sich für Bäume in SQL nutzen

* Allerdings: statt Pfaden mit variabler Länge einfach eine feste Länge nehmen
* Und mit Binärdaten: bei jedem Pfad-Element wird eine laufende Nummer dieses Elements als Binärstring abgelegt. Alle zusammen bilden die *gid*, Genealogical ID
* Limitierung der Anzahl der Kinder: bei 2 Bytes 65536 Kinder, bei 3 Bytes über 16 Millionen, bei 4 Bytes über 4 Milliarden ...

Beispiel

Länge der GID: die ersten beiden zwei Bytes, dann ein Byte

id	gid (hex)	Bemerkung
1	0000	Erster Eintrag
2	0000 0000	Kommentar auf ID 1
3	0000 0001	Noch ein Kommentar auf ID 1
4	0000 0001 00	Kommentar auf 3
5	0000 0001 01	Noch ein Kommentar auf 3
…	…	…
300	0000 0001 FF	256. Kommentar auf 3
301	0001	Neuer Baum
302	0000 0000 01	Ein später Kommentar auf 2
303	0001 0001	Kommentar auf 301

Vorteile

* Durch diese Struktur ist ein sehr einfaches undeffizientes Handling möglich
* Das Auslesen eines ganzen Baums (bei Foren also Threads) ist genau so schnell möglich wie bei einer flachen Darstellung.
* Einfügen ist auch schnell, im Parent muss man sich nur die letzte vergebene Nummer merken

Genealogical Trees in SQL

```
CREATE TABLE gtree AS
   (
   id         SERIAL,                 -- zum Auffinden via URL-Parameter
   gid        bytea PRIMARY KEY,      -- gid besteht aus Binärdaten
   rgid       bytea,                  -- negativ für Rückwärts sortieren
   nchildren INTEGER                  -- Anzahl der direkten Kindelemente
   --         ...                        Bei Bedarf mehr
   );
```

* Daten können in anderer Tabelle stehen, die auf die *id* verweisen: ein Element kann mehrfach im Baum
* Auslesen einfach:

```
SELECT * FROM gtree WHERE gid LIKE '$gid%' ORDER BY gid
```

* Der Client muss dann manuell die Tiefe im Baum herausfinden, einfach anhand der Länge der *gid*

Ende

* Die Folien vom Vortrag werde ich unter der folgenden Adresse nachreichen: http://alvar.a-blast.org/vortraege/perl-workshop/
* Da werde ich dann auch noch Benchmarks zum Datenbank-Contest ablegen – sobald ich das mal irgendwann schaffe

Alvar Freude
http://alvar.a-blast.org/
alvar@a-blast.org