Einführung in das Tuning von MySQL

DOAG-Konferenz 2016, Nürnberg

Jörg Brühe

Senior Support Engineer, FromDual GmbH

joerg.bruehe@fromdual.com

Über FromDual GmbH

- FromDual bietet neutral und unabhängig:
  - Support für MySQL, Galera Cluster und MariaDB
  - remote-DBA Dienstleistungen
  - Beratung für MySQL, Galera Cluster und MariaDB
  - MySQL und MariaDB Schulungen
- Oracle Silber Partner (OPN)
- Mitglied bei DOAG, SOUG, /ch/open und OSBA

www.fromdual.com

Zur Person: Jörg Brühe

- **Entwicklung verteiltes SQL-DBMS:**
  Unix-Portierung, Tests, Diagnose, ...
- **MySQL Build Team:**
  Release-Builds inkl. Tests, Paketierung, Skripte, ...
- **DBA:**
  MySQL für Web-Plattform (Master-Master-Replikation)
- **Support-Ingenieur (FromDual):**
  Support + Remote-DBA für MySQL / MariaDB / Percona mit oder ohne Galera Cluster; Beratung, Schulung

Inhalt

1) Definition Tuning, Ansatzpunkte
2) Möglichkeiten und Strategien
3) Anwendung und Schema tunen
4) SQL-Statements und Strategien tunen
5) Allgemeines zum Thema Index
6) Server-Konfiguration tunen, Hardware
7) Ausblick

1) Definition Tuning, Ansatzpunkte

2) Möglichkeiten und Strategien

3) Anwendung und Schema tunen

4) SQL-Statements und Strategien tunen

5) Allgemeines zum Thema Index

6) Server-Konfiguration tunen, Hardware

7) Ausblick

Definition (1)

”Tuning steht für

- ...
- bei Computern die gegenseitige Feinabstimmung von Hardware-Komponenten oder Software, siehe Konfiguration (Computer)
- Tuning (Datenbank), die Feinabstimmung von Parametern einer Datenbankinstallation
- ...“

Wikipedia: ”Tuning“

Definition (2)

”Unter Tuning versteht man die Feinabstimmung von Parametern einer Datenbank-Installation oder beeinflussbarer Eigenschaften einer Datenbankanwendung mit dem Ziel einer Performance-Verbesserung.“

Wikipedia: ”Tuning (Datenbank)“

Performance-Verbesserung heißt ...

… schneller
(für den Einzelnen)

… besser nutzen
(mehr gleichzeitige Nutzer)

Datenbank-Tuning

… mehr erledigen
(für alle zusammen)

… weniger Maschinen-Last
(CPU, IO, RAM)

Einheiten / Ausdrucksweise

- ”schneller“ = Ausführungsdauer
  ”Latenz“, Zeit in ms oder s
- ”mehr erledigen“ = Transaktionen je Zeit
  ”Durchsatz“, TX / s
- ”besser nutzen“ = Parallelität
  ”Skalierung“, Anzahl gleichzeitige Benutzer
- ”Maschinen-Last“ = Ressourcen-Nutzung
  ”CPU- / IO-bound“, Prozent

Tuning-Bedarf erkennen (1)

- Einzel-Nutzer: System reagiert (zu) langsam
  = Latenz zu hoch
- Mehrere Nutzer: Weniger Abfragen / Trans. je Zeit als erwartet/benötigt
  = Durchsatz zu gering
- Viele Nutzer: Durchsatz nimmt ab
  = System skaliert nicht
- Ressourcen: CPU / IO komplett ausgelastet
  = Ressource ist begrenzender Faktor

Tuning-Bedarf erkennen (2)

- **Latenz:**
  Benutzer-Empfinden, Stoppuhr
- **Durchsatz:**
  Monitoring (Anwendung oder DB)
- **Skalierung:**
  Durchsatz als Funktion der Nutzer-Zahl
- **Ressourcen:**
  Monitoring (Maschine / Betriebssystem),
  Tools wie vmstat, mpstat, iostat

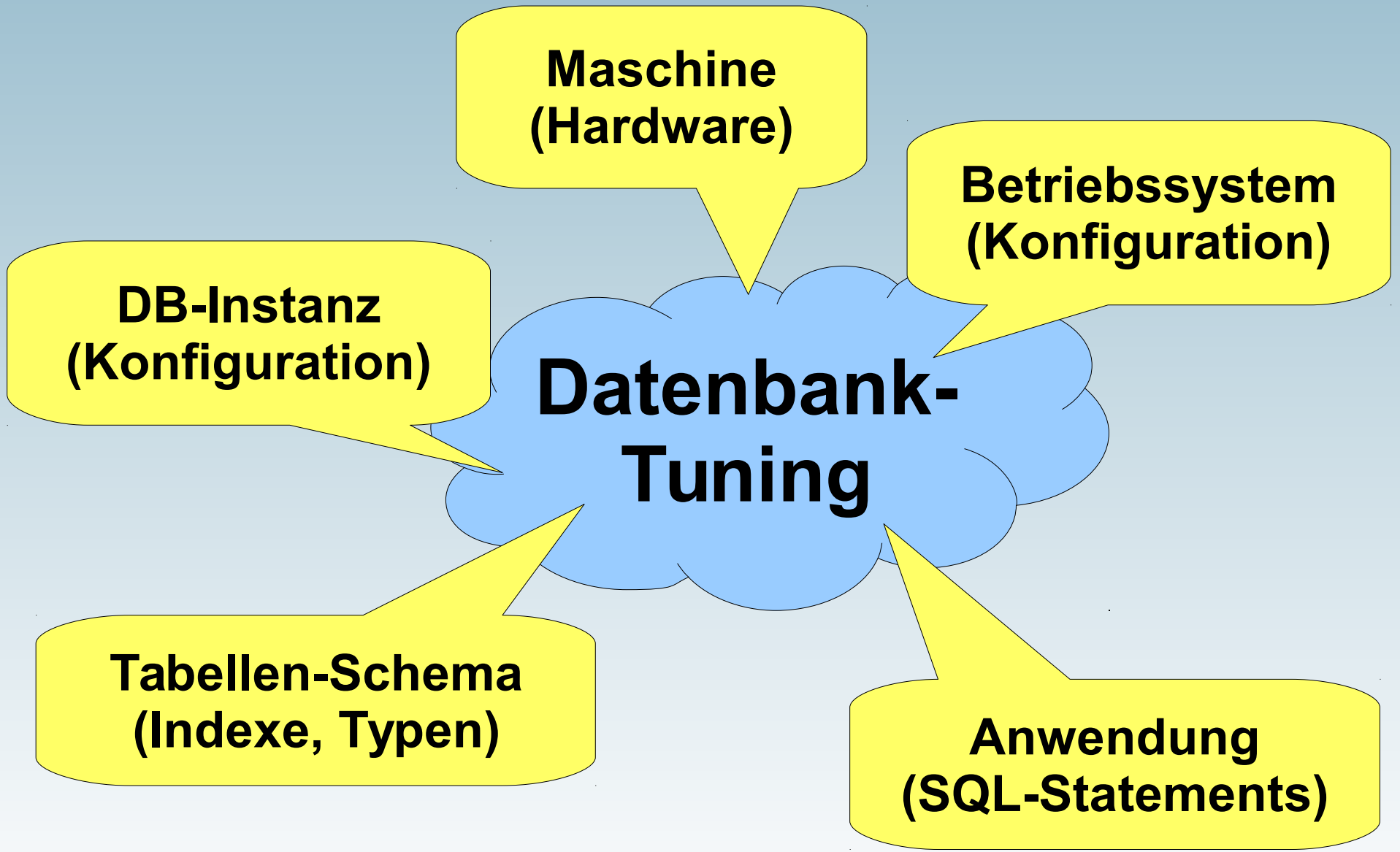
Monitoring

Monitoring liefert die Daten, ...

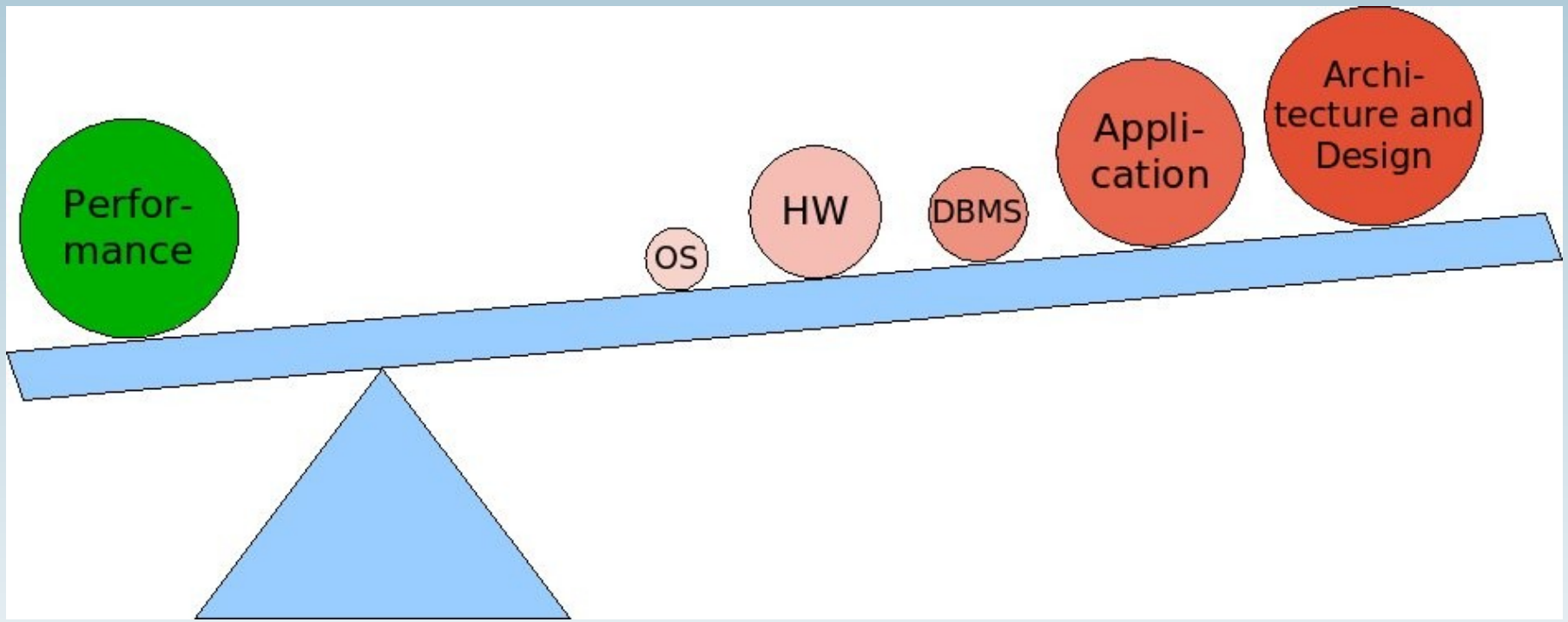
- **auf Maschinen- (HW, OS) und DB-Ebene (SQL-Kommandos, Puffer, ...)**
- **in grafischer Darstellung (Kurven)**
- **um den Tuning-Bedarf zu erkennen**
- **um das Tuning-Ergebnis zu beurteilen**

Ohne Monitoring kein (vollständiges) Tuning!

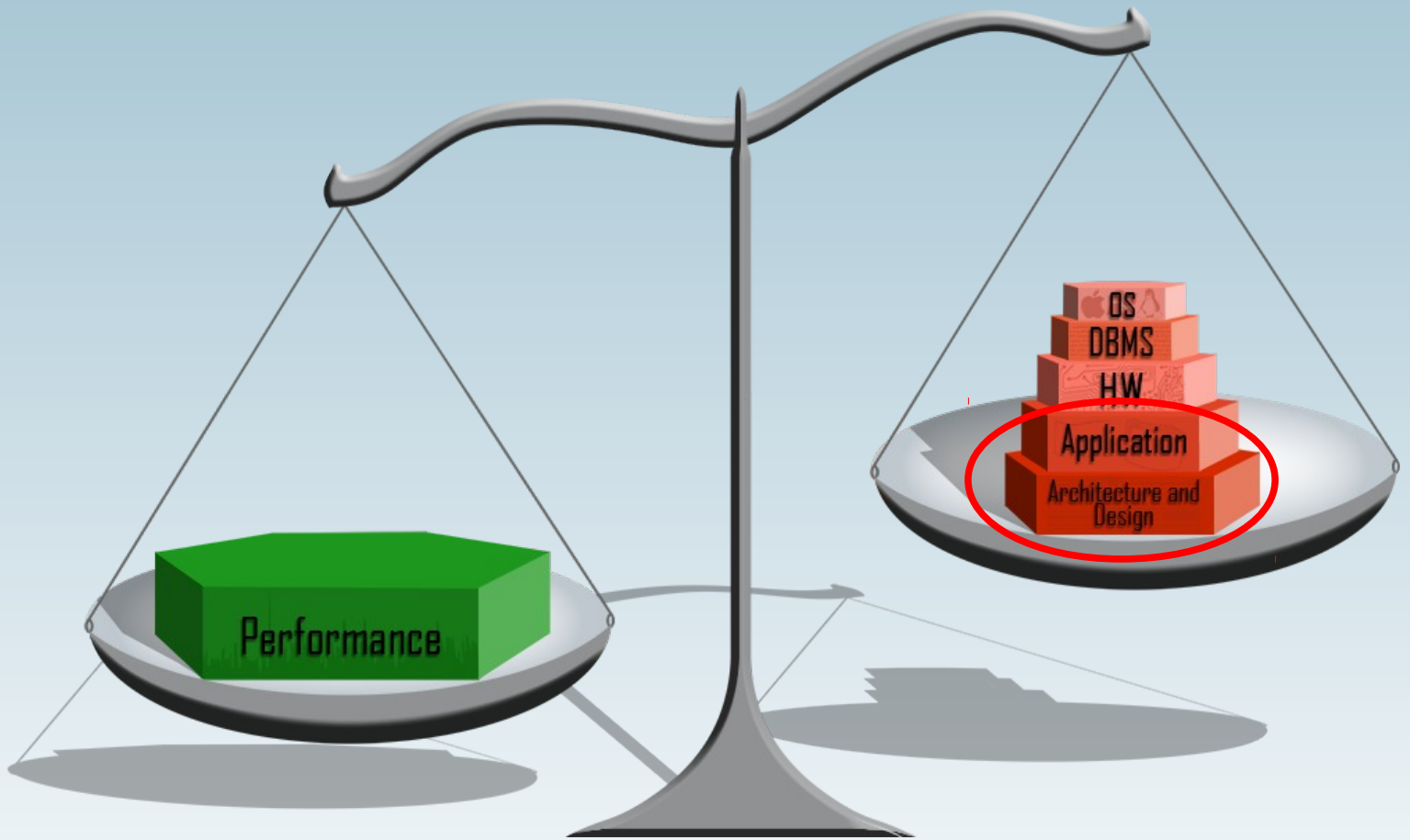
Beeinflussbare Eigenschaften



Relative Wirksamkeit



Relative Wirksamkeit



1) Definition Tuning, Ansatzpunkte

2) Möglichkeiten und Strategien

3) Anwendung und Schema tunen

4) SQL-Statements und Strategien tunen

5) Allgemeines zum Thema Index

6) Server-Konfiguration tunen, Hardware

7) Ausblick

Tuning-Ziele

- Latenz verringern (bestimmte Abfrage)
- Durchsatz steigern
- Skalierung verbessern
- Ressourcen-Nutzung verringern

- Typisch:
  von einer Verbesserung profitieren alle Ziele

Zwei Möglichkeiten (1)

- **Einzel-Aktionen beschleunigen**

  Beispiel IO: schnellere Platte, SSD

- **Einzel-Aktionen vermeiden**

  Beispiel IO: Block im Puffer halten

Zwei Möglichkeiten (2)

- **Einzel-Aktionen beschleunigen**

  Beispiel Scan: Tabelle im Puffer halten (nicht von Platte lesen)

- **Einzel-Aktionen vermeiden**

  Beispiel Scan: Gezielter Zugriff über Index (keinen Scan durchführen)

Ansatz: Beschleunigen

Hardware-Komponente beschleunigen bzw. verstärken

Z.B. schnellere Platte, SSD, schnellere CPUs

+ Ohne System-Eingriff, ohne Nebenwirkungen

+ Einfach

- Hardware-Umbau, Betriebs-Unterbrechung

- Begrenzter Spielraum

- Oft nur geringe Wirkung, skaliert nicht

Ansatz: Vermeiden

Ursache bestimmen, passend ändern

Z.B. Index statt Scan, Join statt Subquery

+ Sehr starke Wirkung möglich
+ Dauerhaft
+ Übertragbar auf alle Instanzen
- Analyse-Aufwand, Eingriff ins System
- Bei Fremd-Code nur begrenzter Spielraum
- Aufwand steigt mit Anwendungs-Komplexität

Was bringt mehr?

- Beschleunigung geht nur in HW
- Beschleunigung in SW ist Vermeidung (IO durch Pufferung, Scan durch Index, ...)
- Vermeidung kann andere Ressourcen belegen (Puffer: RAM)
- Vermeidung bringt den größeren Gewinn

1) Hardware-Upgrade

Schnell und einfach

- Auslastung bestimmen: `vmstat free ...`
- Ist ein Engpass erkennbar?

  %usr + %sys nah an 100 => CPU,

  %iowait nah an (100 / #CPU) => Platte,

  Paging => RAM

  Transfers langsam => Netzwerk

2) MySQL-Konfiguration

Ohne HW-Änderung, ohne SQL-Analyse

- MySQL führt Status-Variable (Zähler, ...)
- Differenz nachher – vorher zeigt Aktionen
- High-Level-Aktionen (Statements) beschreiben Anwendungs-Last
- Low-Level-A. (Puffer-Zugriffe, IO, ...) prüfen: kann Puffer/Cache-Vergrößerung helfen?

  InnoDB Buffer Pool, Table Definition Cache, Table Open Cache, Query Cache, Thread Cache

3) Anwendung und SQL-Statements

Gründlich: Top-down vorgehen,

- **von der Anwendung**
- **über das Schema**
- **und die Statements**
- **zur Ausführungs-Strategie**
- **und zur DB-Konfiguration**

Prinzip: Teure Aktionen identifizieren,
möglichst vermeiden,
sonst beschleunigen

Warum top-down?

- Was oben vermieden wurde,
  braucht unten nicht beschleunigt zu werden

- Änderung der ausgeführten Aktionen
  bewirkt Änderung des Ressourcen-Bedarfs

1) Definition Tuning, Ansatzpunkte

2) Möglichkeiten und Strategien

3) Anwendung und Schema tunen

4) SQL-Statements und Strategien tunen

5) Allgemeines zum Thema Index

6) Server-Konfiguration tunen, Hardware

7) Ausblick

Eigentlich ist es zu spät ...

Anwendung ist geplant:

- **SW-Architektur, Komponenten, ...**
- **Datenfluss**
- **Datenhaltung wie, Maschinen, SAN ja/nein**
- **Server-Architektur (Read-Slave, ...)**
- **Schema-Entwurf**
- **...**

Änderung nur als Entwicklungs-Aufgabe

Manchmal geht noch was ...

Spätere Änderungen nur sehr begrenzt:

- **Datentypen**
- **Table Handler**
- **Primärschlüssel, Indexe**
- **SQL-Statements (nur bei Quell-Zugriff)**

Intensive Tests nötig!

Problem-Punkte finden (1)

”Slow Query Log“ konfigurieren:

- `long_query_time = 0.0`
  `log_queries_not_using_indexes = 1`
  `min_examined_row_limit = 0`
  `log_slow_admin_statements = 1`
  `slow_query_log_file = LOGFILE`

- `slow_query_log = 1|0` für An/Aus

- `log_output = NONE` verhindert das Log!

Problem-Punkte finden (2)

”Slow Query Log“ auswerten:

- `mysqldumpslow -s t LOGFILE \`
  `> LOGEVAL`

  (Oracle)
  oder
- `pt-query-digest LOGFILE > LOGEVAL`
  (Percona)

Pt-query-digest: mehr Statistiken, separat installieren

Mysqldumpslow: Bug#83777 in 5.7

Auswertung

LOGEVAL enthält die SQL-Anweisungen mit der höchsten Gesamt-Laufzeit, absteigend sortiert.

Daraus Hinweise

- auf langdauernde SQL-Anweisungen
- auf lange gehaltene Locks
- auf Code-Fehler (Statement zu häufig)

Anweisungs-Tuning -> nächster Abschnitt

1) Definition Tuning, Ansatzpunkte

2) Möglichkeiten und Strategien

3) Anwendung und Schema tunen

4) SQL-Statements und Strategien tunen

5) Allgemeines zum Thema Index

6) Server-Konfiguration tunen, Hardware

7) Ausblick

Statement-Tuning

- Kandidaten aus dem Slow Query Log
- Ziel: gleiche Semantik, aber billiger
- Wege:
  - Statement ersetzen
    (Anwendungs-Code ändern)
  - Strategie verbessern
    (Indexe anlegen oder ändern)
  - Statement modifizieren
    (”Query Rewrite Plugin“)

Beispiel: Correlated Subquery

- Tabelle ”mitarbeiter“ mit 1000 Einträgen

```
select nachname, vorname,
       geschlecht, abteilung, gehalt
  from mitarbeiter as person
where person.gehalt <
   ( select avg(gehalt)
       from mitarbeiter as abt
      where abt.geschlecht =
                  person.geschlecht and
     abt.abteilung = person.abteilung )
```

Slow Query Log: Inhalt


```
# Time: 2016-11-11T14:27:07.886555Z
# User@Host: class[class] @ localhost []
Id:     2
# Query_time: 0.465798  Lock_time: 0.000710
Rows_sent: 545  Rows_examined: 1001000
SET timestamp=1478874427;
select nachname, vorname, geschlecht,
abteilung, gehalt from mitarbeiter as
person where person.gehalt < ( select
avg(gehalt) from mitarbeiter as abt where
abt.geschlecht = person.geschlecht and
abt.abteilung = person.abteilung );
```

Ergebnis mysqldumpslow


```
Count: 3  Time=0.46s (1s)
Lock=0.00s (0s)  Rows=545.0 (1635),
class[class]@localhost

  select nachname, vorname, ...
[[Statement, Platzhalter für Werte]]
```

Liefert:

- Häufigkeit, Dauer, Lock-Zeit, Zeilen, User

Fehlt: `Rows_examined`

Ergebnis pt-query-digest (1)


```
# 210ms user time, 40ms system time, 25.74M rss, 221.88M vsz
# Current date: Thu Nov 10 21:36:46 2016
# Hostname: cent7-class
# Files: /var/lib/mysql/cent7-class-slow.log
# Overall: 6 total, 4 unique, 0.26 QPS, 0.06x concurrency _________
# Time range: 2016-11-10T20:35:34 to 2016-11-10T20:35:57
# Attribute     total  min     max     avg     95%  stddev  median
# ============  =====  ====  =======  =======  =======  =======  =======
# Exec time         1s  9us    518ms    244ms    501ms    236ms    455ms
# Lock time       3ms    0      1ms    431us      1ms    550us    542us
# Rows sent     1.60k    0      545   272.67   537.02   268.35   537.02
# Rows examine  2.86M    0  977.54k  488.77k  961.27k  480.63k  961.27k
# Query size      847   27      253   141.17   246.02   108.60   246.02

# Profile
# Rank Query ID           Response time Calls R/Call V/M   Item
# ==== ================== ============= ===== ====== ===== ==========
#    1 0x222C807B7...  1.4626 99.9%     3 0.4875  0.00 SELECT mita
# MISC 0xMISC           0.0020  0.1%     3 0.0007   0.0 <3 ITEMS>
```

Ergebnis pt-query-digest (2)


```
# Query 1: 0.50 QPS, 0.24x concurrency, ID 0x222C807B... at byte 1085
# This item is included in the report because it matches –limit.
# Scores: V/M = 0.00
# Time range: 2016-11-10T20:35:46 to 2016-11-10T20:35:52
# Attribute    pct total     min     max     avg     95% stddev  median
# ============ === ===== ======= ======= ======= ======= ====== =======
# Count         50     3
# Exec time     99     1s   453ms   518ms   488ms   501ms    28ms   477ms
# Lock time    100   3ms   210us     1ms   862us     1ms   503us   881us
# Rows sent     99  1.60k     545     545     545     545       0     545
# Rows examine 100  2.86M 977.54k 977.54k 977.54k 977.54k       0 977.54k
# Query size    89    759     253     253     253     253       0     253
# String:
# Databases    FlughafenDB
# Hosts        localhost
# Users        class
```

Rows examined / sent = 977.540 / 545 = 1793 !

Laufzeit 453 ms .. 518 ms, Durchschnitt 488 ms

Ergebnis pt-query-digest (3)


```
# Query_time distribution
#   1us
#  10us
# 100us
#   1ms
#  10ms
# 100ms  ##############################################################
#    1s
#  10s+
# Tables
#    SHOW TABLE STATUS FROM `FlughafenDB` LIKE 'mitarbeiter'\G
#    SHOW CREATE TABLE `FlughafenDB`.`mitarbeiter`\G
# EXPLAIN /*!50100 PARTITIONS*/
select nachname, vorname, geschlecht, abteilung, gehalt
 from mitarbeiter as person
 where person.gehalt <
  ( select avg(gehalt) from mitarbeiter as abt
    where abt.geschlecht = person.geschlecht
      and abt.abteilung = person.abteilung )\G
```

Explain


```
mysql> explain select nachname, vorname, geschlecht, ... \G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: person
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1000
     filtered: 100.00
        Extra: Using where
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: abt
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1000
     filtered: 2.00
        Extra: Using where
2 rows in set, 3 warnings (0,01 sec)
```

Create Index, Explain


```
mysql> create index mitarb_abt_sex on mitarbeiter (abteilung, geschlecht);
Query OK, 0 rows affected (0,15 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain select nachname, vorname, geschlecht, ...  )\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: person
          ...  ...
         rows: 1000
     filtered: 100.00
        Extra: Using where
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: abt
   partitions: NULL
         type: ref
possible_keys: mitarb_abt_sex
          key: mitarb_abt_sex
      key_len: 4
          ref: FlughafenDB.person.abteilung,FlughafenDB.person.geschlecht
         rows: 100
     filtered: 100.00
        Extra: NULL
2 rows in set, 3 warnings (0,00 sec)
```

Pt-query-digest mit Index


```
# Attribute    pct   total ...      avg      95%  stddev  median
# ============ === ======= ... ======= ======= ======= =======
# Count         25       1
# Exec time     84    180ms ...    180ms    180ms       0    180ms
# Lock time    100      2ms ...      2ms      2ms       0      2ms
# Rows sent     99      545 ...      545      545       0      545
# Rows examine 100 137.93k ... 137.93k 137.93k       0 137.93k
# Query size    72      229 ...      229      229       0      229
```

Rows examined / sent = 137.930 / 545 = 253

Index = Ersparnis um Faktor 7

Laufzeit 180 ms (nur 1 Messung)

Beispiel: Join


```
select nachname, vorname,
       geschlecht, abteilung, gehalt
  from mitarbeiter as person
inner join
    (select geschlecht, abteilung,
            avg(gehalt) as wert
       from mitarbeiter group by
         geschlecht, abteilung) as abt
using (geschlecht, abteilung)
where person.gehalt < abt.wert
```

Pt-query-digest für Join


```
# Attribute    pct   total ...      avg      95%  stddev  median
# ============ === ======= ... ======= ======= ======= =======
# Count         50       3
# Exec time     95     43ms ...     14ms     18ms      4ms     16ms
# Lock time    100      4ms ...      1ms      2ms    467us      1ms
# Rows sent     99    1.60k ...      545      545       0      545
# Rows examine 100    8.85k ...    2.95k    2.95k       0    2.95k
# Query size    89      786 ...      262      262       0      262
```

Rows examined / sent = 2.950 / 545 = 5,4

Join = Ersparnis um Faktor 331

Laufzeit 8 ms .. 18 ms, Durchschnitt 14 ms

Fazit

- Ein Index kann eine (Sub)Query um Faktoren beschleunigen
- Die Vermeidung einer (Correlated) Subquery bringt den (deutlich) größeren Gewinn

1) Definition Tuning, Ansatzpunkte

2) Möglichkeiten und Strategien

3) Anwendung und Schema tunen

4) SQL-Statements und Strategien tunen

5) Allgemeines zum Thema Index

6) Server-Konfiguration tunen, Hardware

7) Ausblick

Primärschlüssel

- Primärschlüssel (PK) ist Pflicht!
- PK ist essentiell bei Replikation im Row-Format
- PK soll sich nicht ändern
- Kurzer PK bringt Vorteile:
  - Höherer Fan-Out im Tabellen-Baum, also geringere Baum-Höhe
  - Weniger Platzbedarf im Sekundärschlüssel

Sekundärschlüssel / Index (1)

- Sekundärschlüssel (”Index“) kann Suche beschleunigen
- Index ist logisches Mapping: Wert -> PK
  Reorganisations-frei
- Index-Nutzung nicht für Funktions-Argument:

```
... where date(zeitpunkt) = '2016-11-15'

... where zeitpunkt
    BETWEEN unix_timestamp
                  ('2016-11-15 00:00:00')
        AND unix_timestamp
                  ('2016-11-16 00:00:00')
```

Sekundärschlüssel / Index (2)

- Index ist zwingend für ”foreign key“
- Index-Nutzung nur bei hoher Selektivität:
  Zugriff über Index ist Random-IO
- (Nur) führende(s) Feld(er) nötig:
  `... on tab1 (col1)` überflüssig bei
  `... on tab1 (col1, col2)`
- Index verlangsamt Insert/Update/Delete:
  Überflüssige / unbenutzte Indexe löschen!

1) Definition Tuning, Ansatzpunkte

2) Möglichkeiten und Strategien

3) Anwendung und Schema tunen

4) SQL-Statements und Strategien tunen

5) Allgemeines zum Thema Index

6) Server-Konfiguration tunen, Hardware

7) Ausblick

Konfiguration und Status

- Konfiguration = Einstellung
  Parameter, my.cnf und Default-Werte
- `show global variables`

- Status = Ereignis-Zähler im Betrieb
- `show global status`
  (Umzug nach `performance_schema`)
- Aufwand = (Zähler danach) – (Zähler davor)

- Status zeigt, ob Konfiguration passt

Status-Auswertung: Beispiel

	davor	danach	Diff	1/s
Innodb_buffer_pool_read_requests	4977257422	4993806133	16548711	1591
Innodb_buffer_pool_reads	133117509	133567759	450250	43
...	...	...	...	
Uptime	4409226	4419624	10398	

- **InnoDB Buffer Pool Hitrate = 97,2 %**
- **43 Read je Sekunde wegen Nicht-Treffer**

- **Weitere relevante Puffer / Caches:**
  Thread Cache, Query Cache (prüfen!),
  Table Definition Cache, Table Open Cache

Puffer und Caches (1)

Zugriffe und (Miss)Erfolge in Status-Werten:

- InnoDB Buffer Pool:
  Vermeidet Disk-Read
  `Innodb_buffer_pool_read_requests`
  `Innodb_buffer_pool_reads`

- Thread Cache:
  Vermeidet pthread_create()
  `Threads_connected`
  `Threads_created`

Puffer und Caches (2)

- Table Definition Cache:
  Vermeidet Lesen + Analyse der frm-Files

```
Open_table_definitions
Opened_table_definitions
```

- Table Open Cache:
  Vermeidet open() der ibd-Files

```
Open_tables
Opened_tables
```

Puffer und Caches (3)

- Query Cache:
  Vermeidet Select-Ausführung
  `Qcache_hits`
  `Qcache_inserts`
  `Qcache_lowmem_prunes`
  (Verwaltung kann Flaschenhals werden!)

Aktions-Vermeidung benötigt RAM
=> Risiko Paging

Hardware-Prioritäten

- Platte
  - Hohe IOPS wegen Commit + Random IO
  - SSD, NVM
- CPU
  - Single-Core Leistung wichtig!
  - Multi-Core lohnt nur bei hoher Parallelität
  - Multi-Socket => NUMA-Risiko
- RAM
  - Paging ist fatal!
  - Große Puffer reduzieren IO
- Netzwerk
  - Stabilität
  - Durchsatz für Backup-Transfer

1) Definition Tuning, Ansatzpunkte
2) Möglichkeiten und Strategien
3) Anwendung und Schema tunen
4) SQL-Statements und Strategien tunen
5) Allgemeines zum Thema Index
6) Server-Konfiguration tunen, Hardware

7) Ausblick

Weitere Hilfe

- ”High-Performance MySQL“
  Schwartz, Zaitsev, Tkachenko; O'Reilly
- FreeNode: `#mysql #mysql.de #maria`
- Hersteller-Support
- Dienstleister, Schulungen, Webinare
- Google etc.
- Informationen für Hilfe:
  - Genaue Version MySQL + Betriebssystem
  - `show create table` + Statement + Explain

Alles hat ein Ende ...

- Ausschlaggebend: Anwendungs-Design
- ”Scale-Out“ für lese-intensive Last erwägen
- ”Gesetz vom abnehmenden Grenznutzen“

Q & A

Fragen ?

Diskussion?

Wir haben Zeit für ein persönliches Gespräch ...

- FromDual bietet neutral und unabhängig:
  - Beratung
  - Remote-DBA
  - Support für MySQL, Galera, Percona Server und MariaDB
  - Schulung

www.fromdual.com/presentations