# MySQL Performance Tuning

**Kristian Köhntopp,**
**Senior Consultant,**
**MySQL GmBH**

**<kris@mysql.com>**
**http://blog.koehntopp.de**
**http://mysqldump.azundris.com**

Handy aus?

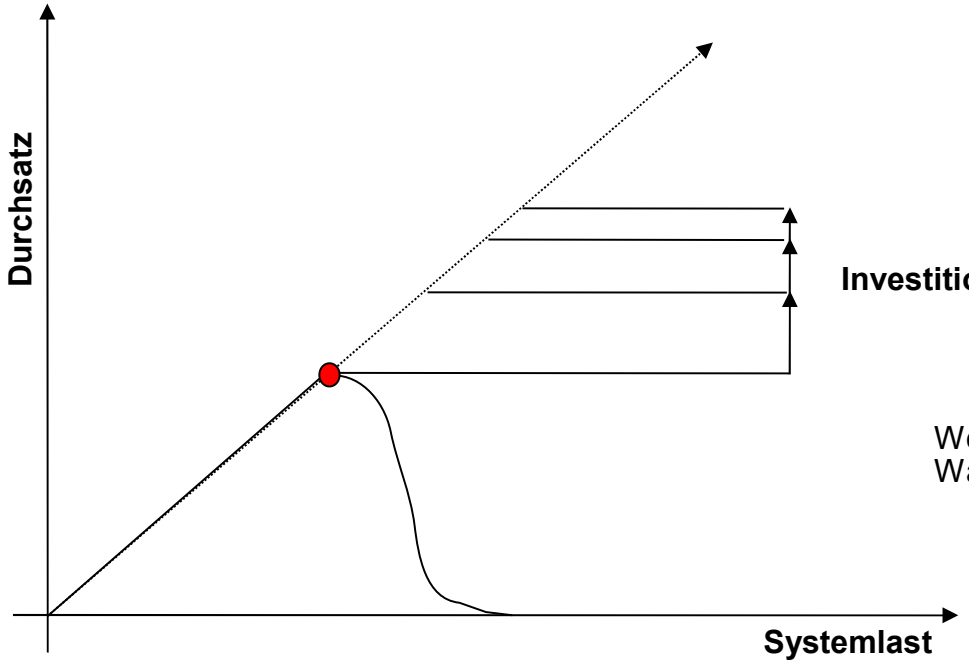
# Performance und Warteschlangen

- **Performance und Warteschlangen**
- Richtig benchmarken
- Ansatzpunkte für Optimierungen
  - Architektur
  - Schema Design
  - Indizierung
  - Server Tuning
    - Globale Puffer
    - Threadlokale Puffer
    - Andere Parameter
  - Hardware und Betriebssystem
- MySQL 5.1

# Was ist Performance

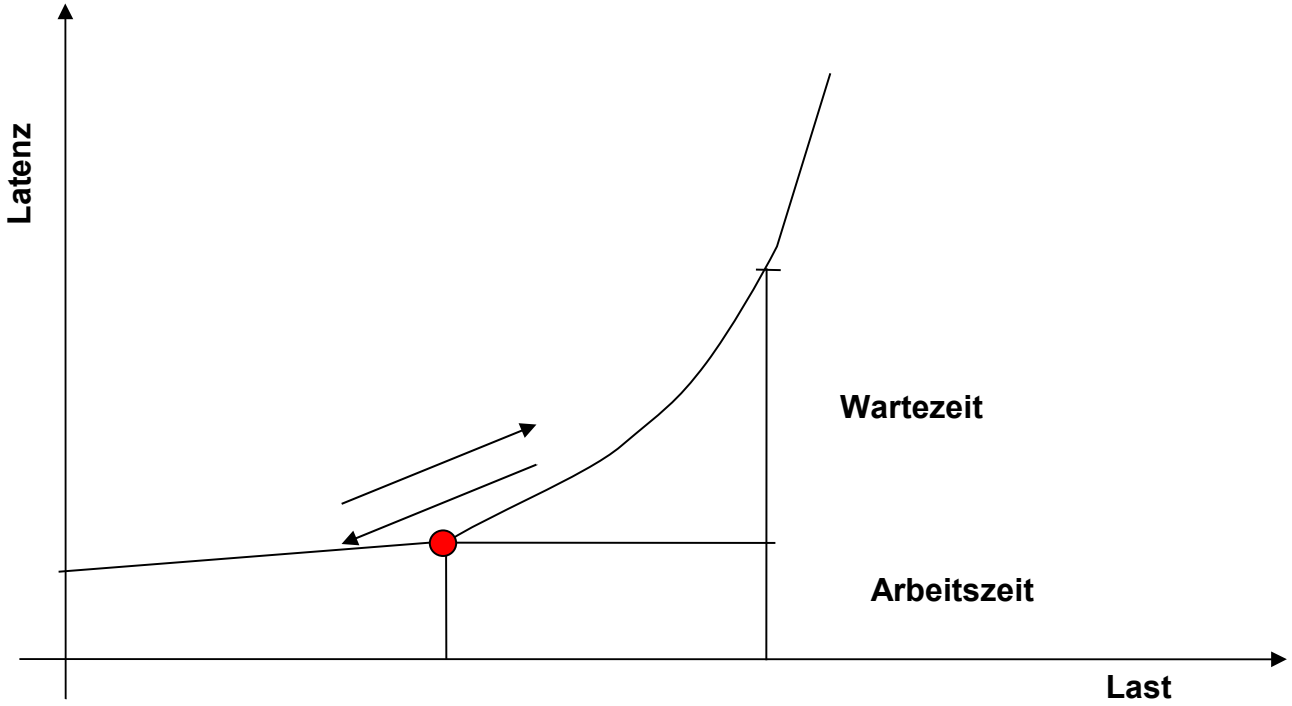
- “Systemauslastung herstellen”
  - Wenn das System ausgelastet ist und man dennoch mehr Leistung braucht, dann ist Scalability gefragt
- Metriken:
  - Latenz (Die Antwort soll möglichst schnell da sein)
  - Durchsatz (Es sollen möglichst viele Antworten pro Zeit geliefert werden)
  - Skalierbarkeit (Beibehaltung der Latenz bei Erhöhung der Parallelität)

# Sättigung
# (Durchsatz nach Last)

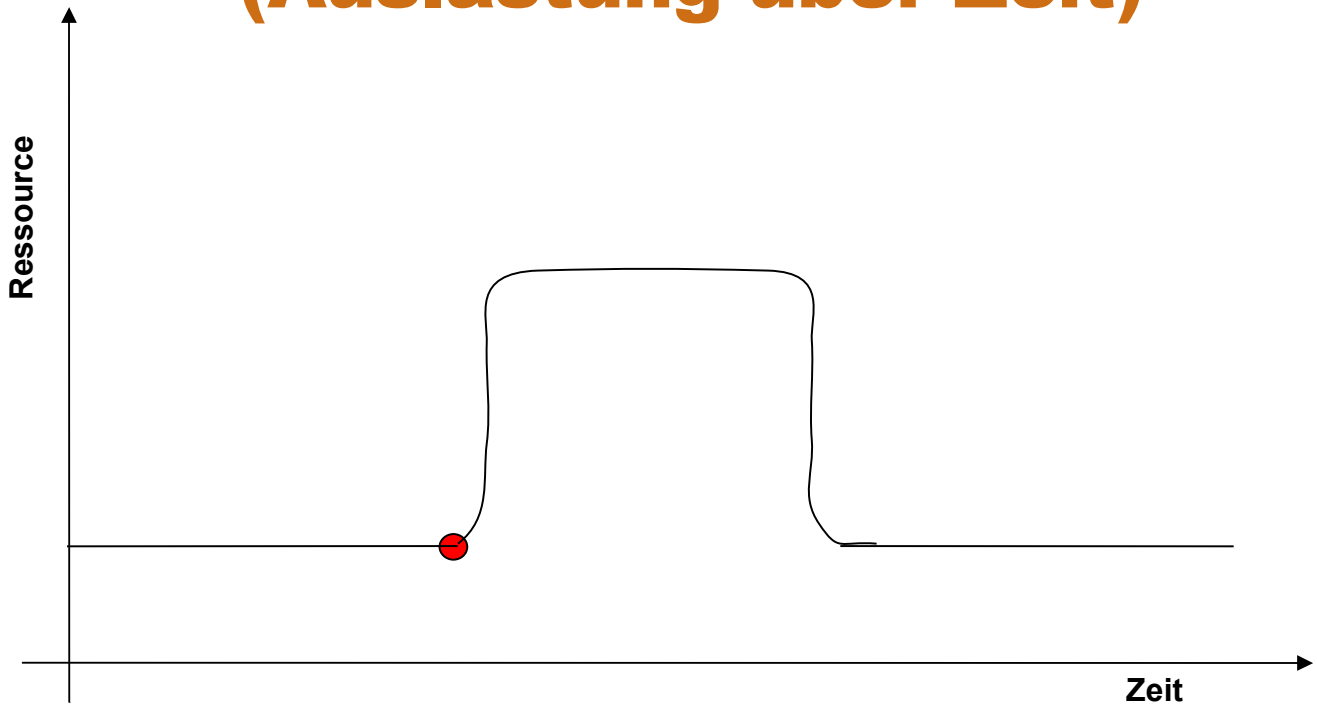



Wo liegt dieser Punkt?
Was ist der Grund für die Sättigung?

# Sättigung
# (Latenz nach Last)

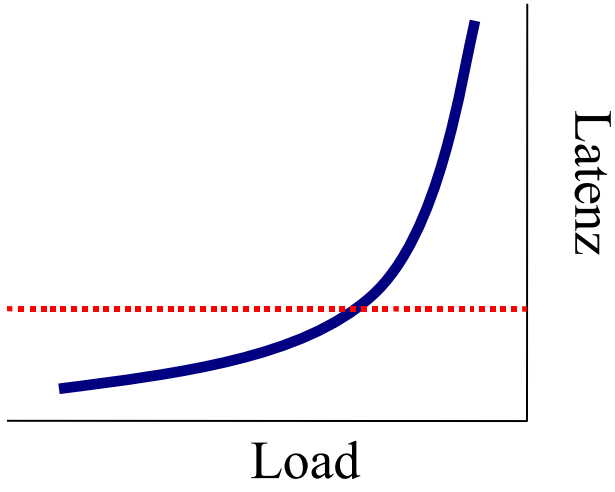


# Sättigung
# (Auslastung über Zeit)



# Warteschlangentheorie

- Latenz = Wartezeit + Bearbeitungszeit
  - Telefonzelle:
    - Alle drei Minuten trifft ein Kunde ein
    - Ein Gespräch dauert im Schnitt etwa drei Minuten
    - → Die Schlange wächst und wächst. Wieso?
- Systemverhalten am Sättigungspunkt ist nicht linear!
- Das System stirbt an Wartezeit.
  - → Wer wartet wann worauf?
  - → Kürzere Verarbeitungszeit
  - → Parallelisierung



# Servicezeit/Wartezeit

- Die wichtigste Frage: Womit verbrennen wir Zeit?
  - Festplatte, Locks, Netzwerk, CPU...
- Direkte Messung
  - Summierung aller Abfrage-Zeiten von Internetseiten
- Indirekte Messungen
  - Performancecounter
    - SHOW GLOBAL STATUS, SHOW MUTEX STATUS, SHOW FULL PROCESSLIST
  - Profiling
    - oprofile, gprof
  - System-Instrumentation
    - iostat, vmstat, sar, dstat, dtrace

# Also wie viel jetzt?

- Hardware definiert die erreichbaren Leistungsgrenzen
  - RAM: Access Time im ns ($10^{-9}$) Bereich, Granularität 1 Byte
  - Disk: Access Time in ms ($10^{-3}$) Bereich, Granularität 1 KB
  - → RAM ist 1 000 bis 10 000 mal schneller als eine Platte.
  - → Speicher gut. Mehr Speicher besser.
- Seek: Eine Platte liefert zwischen 125 und 200 Seeks/s.
  - Ein Row Access kann im schlimmsten Fall mehrere Seeks kosten.
- Scan: Eine Platte liefert bis zu 50 MB/sec an Daten.
  - Das sind ca. 100 000 Rows/sec.

# Richtig benchmarken

- Performance und Warteschlangen
- ➔ **Richtig benchmarken**
- Ansatzpunkte für Optimierungen
  - Architektur
  - Schema Design
  - Indizierung
  - Server Tuning
    - Globale Puffer
    - Threadlokale Puffer
    - Andere Parameter
  - Hardware und Betriebssystem
- MySQL 5.1

# Benchmarks

- Anwendungsgebiete:
  - Den zu messenden Systemzustand gezielt herstellen
  - Vergleichszahlen generieren um Fortschritt zu vergleichen
- Aussagekräftige Benchmarks sind nicht leicht:
  - Tatsächliche Systemlast muß genau reproduziert werden
  - Die Ergebnisse müssen korrekt gelesen werden

# Typische Fehler

- Falsche Datenmenge (1GB statt 100 GB)
  - Kleine Datenmenge paßt in den Cache, System ist mit Speicher gesättigt.
- Falsche Locality (Gleichverteilung statt “long tail”)
  - Test mit dem Postleitzahlen 7xxxx, Abfragen nach Postleitzahlbereichen sind immer “wahr” oder “falsch”
  - Alle Titel im Buchladen werden gleich häufig bestellt, es gibt keinen “Harry Potter”
- Falsche Concurrency (1 Thread statt 500 Threads)
  - So kann man keine Locks erzeugen
- Falsches Locking (keine Think Times statt Denkpausen)
  - So erzeugt man haufenweise Locks

# Richtig Benchmarken

- Was soll erreicht werden?
- Konfiguration reproduzierbar machen, Ergebnisse sichern
  - *Schema Dump, my.cnf, Konfiguration des Lastgenerators*
  - Hardware/OS Konfigurationsdateien wenn benötigt
  - Meßwerte
- Ein Problem zur Zeit bearbeiten
  - Nur eine Änderung zur Zeit testen
  - Mit warmen Caches testen
  - Welche Caches testen wir? (Application Caches, Query Cache, Key Cache, OS Buffer Cache, Controllercache)

# Ansatzpunkte für Optimierungen

- Performance und Warteschlangen
- Richtig benchmarken
- Ansatzpunkte für Optimierungen
  - ➔ **Architektur**
  - Schema Design
  - Indizierung
  - Server Tuning
    - Globale Puffer
    - Threadlokale Puffer
    - Andere Parameter
  - Hardware und Betriebssystem
- MySQL 5.1

# Architektur

- Datenbanken traditionell:
  - Garantierte Korrektheit
    - 2PC
    - ACID
- “Jedes asynchrone System kann
  durch Einführen von Waits
  in ein synchrones System umgewandelt werden.”
- “Best Effort” Ansatz
  - siehe TCP/IP
  - Behandle die Fehlerfälle nicht, sondern eskaliere sie zur nächsthöheren Schicht.
  - Konzentration auf die Optimierung der häufigen Fälle

# Asynchrone Systeme

- Vertikal Skalieren
  - Kauf größerer Rechner mit mehr CPUs
  - Bau eng gekoppelter Systeme, die traditionelle Datenbankparadigmen umsetzen
- Horizontal Skalieren
  - Kauf vieler kleiner Systeme
  - Bau lose gekoppelter Systeme, die Fehler akzeptieren und eskalieren

# Schema Design

- Performance und Warteschlangen
- Richtig benchmarken
- Ansatzpunkte für Optimierungen
  - Architektur
  - **Schema Design**
  - Indizierung
  - Server Tuning
    - Globale Puffer
    - Threadlokale Puffer
    - Andere Parameter
  - Hardware und Betriebssystem
- MySQL 5.1

# OLTP: Normalisierte Schemata

- Vorteilhaft für OLTP-Anwendungen
  - Transaktionen
  - viele Schreiboperationen
- Minderung von Redundanz
  - Weniger Speicherverbrauch
  - Viele Joins (kann teuer werden)
  - Viele Indices
- Gezielte Denormalisierungen
  - Trigger, die Aggregate pflegen
- Saubere Übersetzung in E/R Diagramm wird ermöglicht

# OLAP: Denormalisierte Schemata

- DWH/OLAP, Berichtserstellung
  - Historisch korrekte Daten:
    - Nicht der aktuelle Preis interessiert, sondern der damalige Verkaufspreis
    - Nicht die aktuelle Kundenanschrift interessiert, sondern die PLZ der damaligen Lieferadresse
  - Keine Joins mehr: Übergang vom Seek zum Scan
- DWH NormalStandardformen:
  - Star
  - Snowflake

# Schema Design - Datentypen

- Kleinere Datentypen: Weniger Blöcke
  - INTEGER statt DECIMAL
  - UNSIGNED wo immer möglich
  - NOT NULL wo immer möglich
  - TINYINT, SMALLINT, MEDIUMINT
- VARCHAR statt CHAR
- VARCHAR(64) statt VARCHAR(255)

- Für Join-Spalten sollten identische Datentypen verwendet werden
  - Sonst Join ohne Index

# Indizierung

- Performance und Warteschlangen
- Richtig benchmarken
- Ansatzpunkte für Optimierungen
  - Architektur
  - Schema Design
  - ➔ **Indizierung**
  - Server Tuning
    - Globale Puffer
    - Threadlokale Puffer
    - Andere Parameter
  - Hardware und Betriebssystem
- MySQL 5.1

# Schema Design - Indizierung

- Ein Index pro Tabelle wird verwendet
  ➔Multicolumn-Indices!
- Präfix-Indices sparen Speicher: Weniger Blöcke
- `create table t (c char(10), index (c(5)));`

- Covering Indices:
  - Die gesuchten Daten stehen im Index
  - Keine Seeks!

    - `select a from t where b = ?`

    - normaler Index wäre (b),
    - „Covering Index“ ist (b,a)

# Schema Design – BTREE Index

- **BTREE Indices**
  - Default Index (außer für MEMORY-Engine)
  - Beschleunigt “=” Zugriffe und Ranges, Sortierung
- **Ein BTREE-Index auf (a, b, id) beschleunigt**
  - `select id from table where a = ? and b = ?`
  - `select id from table where a = ? order by b`
  - `select id from table where a = ? and b between ? and ?`
  - `select id from table where a = ? and b > ?`
  - `select id from table where a = ? and b like 'prefix%'`
  - `select id from table where a = ?`
  - `select id from table order by a`

# Schema Design – Mehr über Indices

- Überindizierung kostet Leistung
  - Aber Unterindizierung ist tödlich für Performance
- Index sollte selektiv sein
  - Index auf Geschlecht ist keine gute Idee, benötigt Verhältnis besser als 1/3
  - Präfix-Indices sparen Speicher, solange Selektivität gegeben ist
- UNIQUE Indices sind effizienter als Multivalue-Indices (“INDEX”)

# Schema Design - Indizierung

- Typische Überindizierung:
  - INDEX (A)
  - INDEX (A,B)
- OPTIMIZE TABLE … um Indizes zusammenzuführen und zu sortieren
- ANALYZE TABLE ... um dem Optimizier ordentliche Statistiken bereitzustellen
  - Immer dann notwendig, wenn sich die Verteilung der Daten geändert hat

# Server-Funktionen zur Optimierung

- EXPLAIN everything!
- --log-slow-queries und mysqldumpslow sind gute Freunde!
  - –log-slow-queries
  - –long-query-time=2
- Entwicklungsrechner:
  - Full Query Log?
  - –log-queries-not-using-indexes
- “SHOW PROCESSLIST”
  - Besser: mytop
- Was eine Abfrage “kostet”
  - **FLUSH STATUS**; <run query>; **SHOW STATUS;**

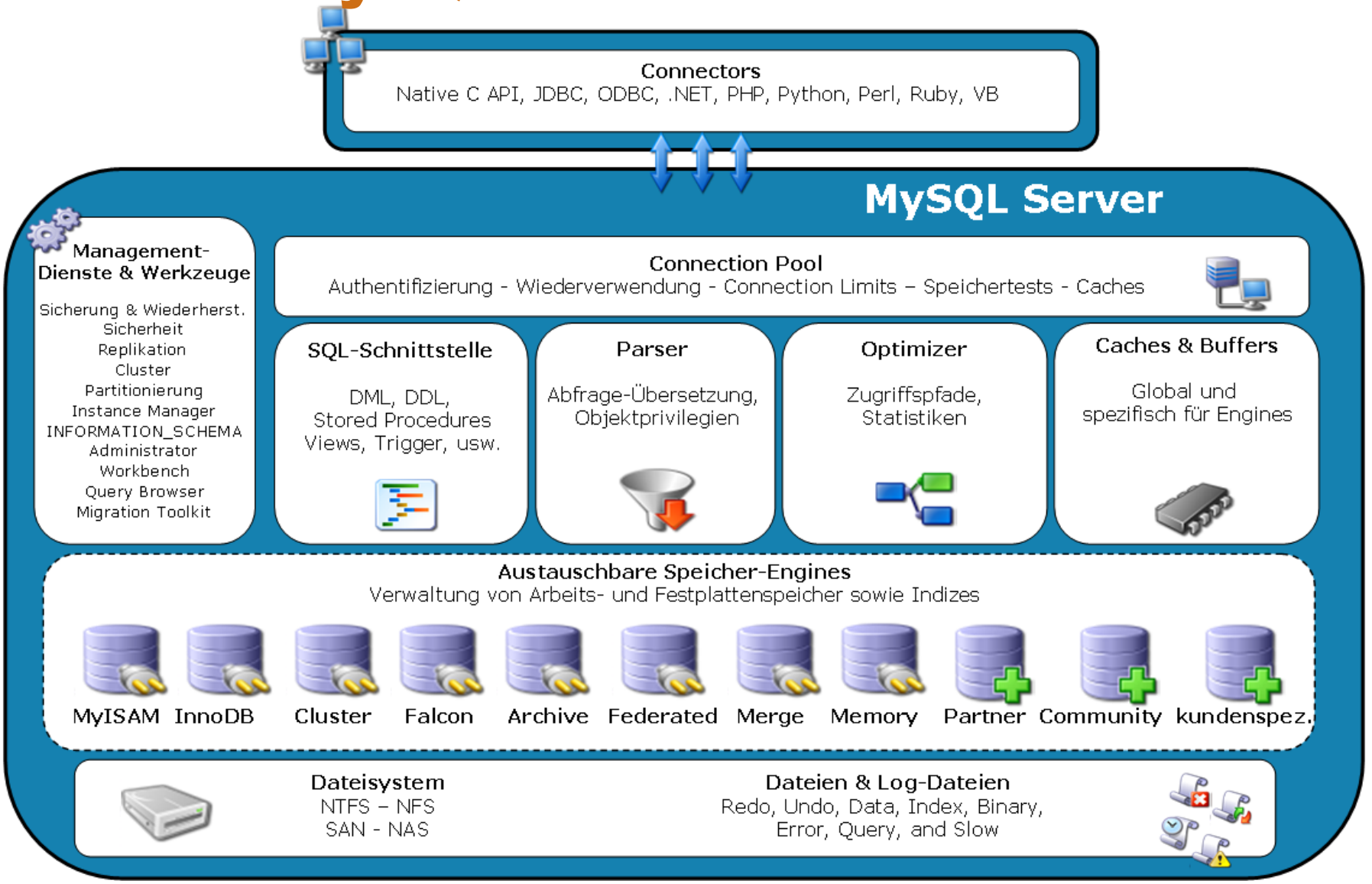
# Steuerung des SQL-Optimizers

- SELECT **STRAIGHT_JOIN** * FROM tbl1,tbl2 ...
  - Erzwingen der Tabellenreihenfolge wie in der Liste angegeben
  - In MySQL 4.0/4.1 essentiell für “große” Joins mit vielen Tabellen
- **USE INDEX** / FORCE INDEX / IGNORE INDEX
  - SELECT * FROM Country **USE INDEX**(PRIMARY)
  - In MySQL selten verwende

# Server Tuning

- Performance und Warteschlangen
- Richtig benchmarken
- Ansatzpunkte für Optimierungen
  - Architektur
  - Schema Design
  - Indizierung
  - **Server Tuning**
    - **Globale Puffer**
    - **Threadlokale Puffer**
    - **Andere Parameter**
  - Hardware und Betriebssystem
- MySQL 5.1

# MySQL Server Architektur

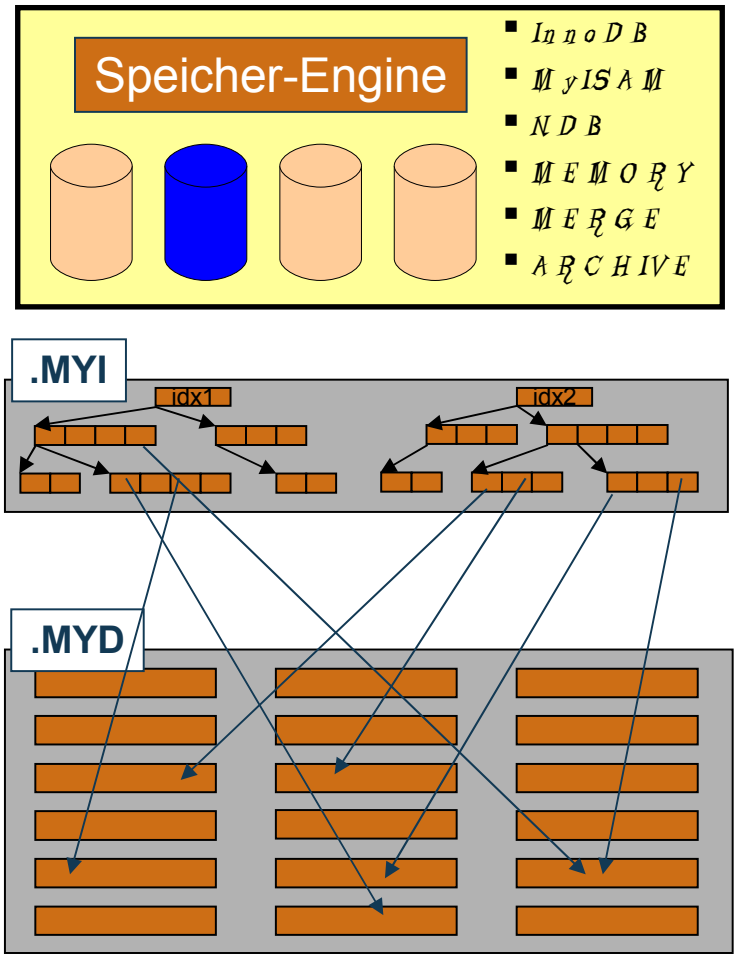


# Globale Puffer

- Wieviel Speicher haben wir zur Verfügung?
  - Serverzielgröße:
    - MySQL teilt sich den Speicher mit einer Anwendung
    - MySQL verwendet das System alleine
  - Verwendete Storage Engine:
    - Engine verwaltet Caches selber (InnoDB) (80%)
    - Engine nutzt OS Caches (MyISAM) (50%)
  - Benötigter Speicher
    - Für Daten + Indices (“RAM gesättigt”, keine Seek Times)
    - Für Indices + Teil der Daten (“OLTP”, Seek Times)
    - Für Teil der Daten/Teil der Indices (“DWH”, Scans)
    - Für Teil der Daten/Teil der Indices (“OLTP mit zu kleinem System”, Seeks + Treshing)

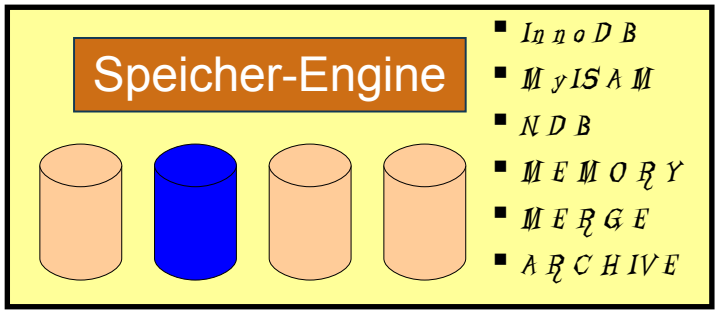
# Speicher-Engine MyISAM

- Keine ACID-Transaktionen, kein Auto-Recovery
- Geringer Footprint für Platte und Speicher, oft 25%-50% weniger als andere RDBMS
- komprimierte Indizes, arbeitet auch ohne Indizes, FULLTEXT, RTREE
- Tabellen-Sperren, gleichzeitiges Einfügen
- Komprimierte “Read-Only”-Version
- Index wird von MySQL zwischen-gespeichert, Daten vom Betriebssystem
- Schnelles Laden und Erstellen des Indexes
- Tabellen können auf Dateisystem-Ebene kopiert werden
- Unterstützt Partitionierung über MERGE



# Optimierung für MyISAM

- **key_buffer_size (8M)**
  - 25-33% des Speichers typisch, wenn nur MyISAM genutzt wird
- **myisam_sort_buffer_size (8M)**
  - Für Index-Erstellung hoch setzen, optimalerweise so groß wie der größte Index in Tabellen
- **myisam_recover=FORCE,BACKUP**
  - Um beruhigt schlafen zu können
- **delay_key_write=ALL**
  - Verzögerte Index-Erstellung für bessere Schreib-Performanz
  - Höheres Risiko von Datenkorruption bei unkontrolliertem Server-Stop

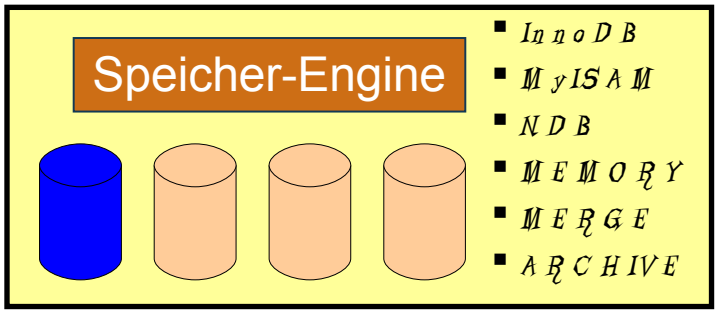


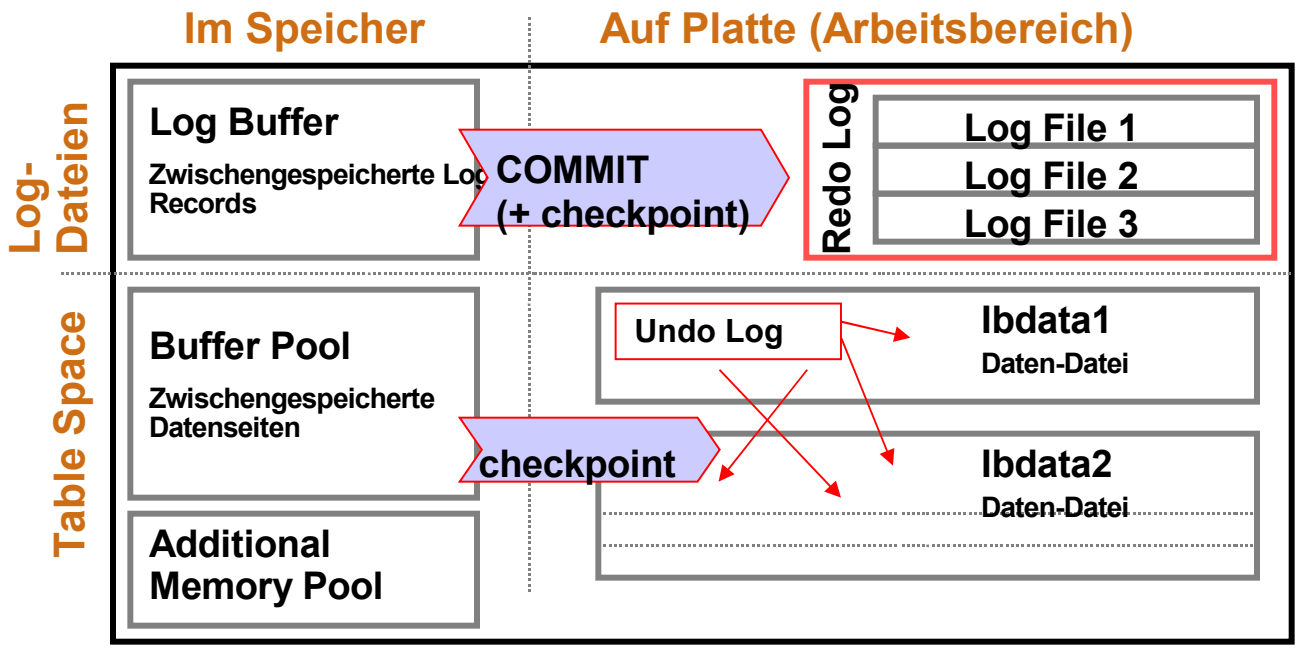

**mysql> show status like 'key%'**

- **Key_read_requests, Key_reads**
  - Key_reads/Key_read_requests – Wert für Nutzung des Key Cache
  - Überprüfen Sie Key_reads/sec, sollte zum IO System passen
- **Key_write_requests, Key_writes**
  - Miss-Verhältnis typischerweise größer
- **Key_blocks_unused**
  - Zu viel Speicher könnte allokiert worden sein

# Speicher-Engine InnoDB

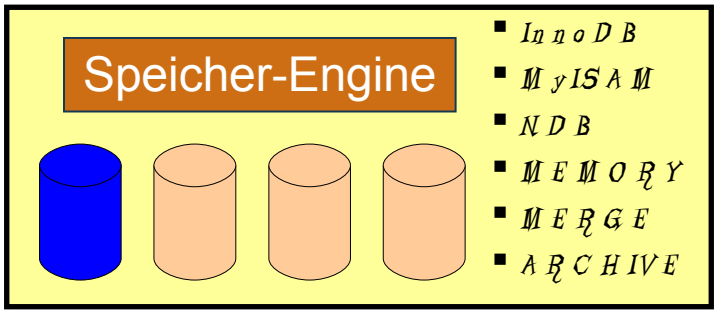
- InnoDB stellt ACID-Transaktionen zur Verfügung:
  - Row-Level- und Multi-Versioning
  - Konfigurierbare Isolation Levels
  - Sperren auf Zeilenebene



# Optimierung für InnoDB

- **innodb_buffer_pool_size**
  - Bis zu 80% des verfügbaren Speichers, wenn nur InnoDB genutzt wird
  - Zwischenspeicherung von Daten & Indizes - im Gegensatz zu MyISAM
- **innodb_log_file_size**
  - Meist zwischen 128M und 512M
  - Check innodb_log_waits; bei vielen Änderungen verschlechtert sich die Performanz durch Checkpointing
  - Recoveryzeit nach Crash direkt abhängig von der größe des ungeflushten Logs
- **innodb_flush_log_at_trx_commit**
  - 1 (langsam) schreibt (fsync) Log bei jedem Commit auf die Platte: Echtes **ACID**
  - 2 (schnell) schreibt Log bei Commit nur in den OS-Zwischenspeicher, Synchronisation auf Platte einmal pro Sekunde
  - 0 (sehr schnell) schreibt (fsync) Log ca. jede Sek.




**mysql> SHOW INNODB STATUS;**

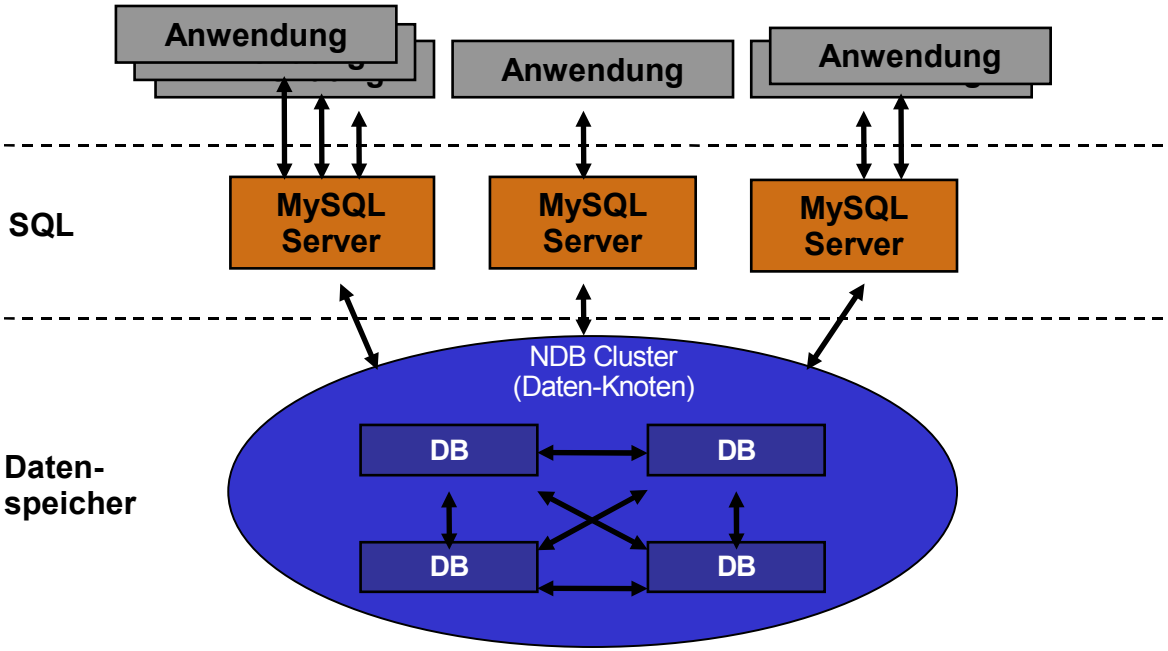
Guter Weg zu sehen, was in InnoDB passiert.

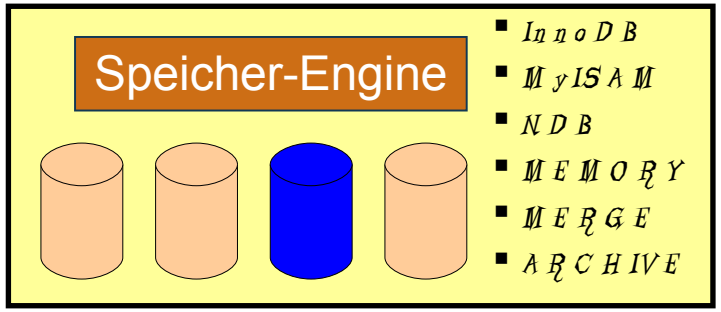
- File IO
- Buffer Pool
- Log Aktivität
- Zeilen-Aktivität

In MySQL 5.0 wurden einige Variablen nach **SHOW STATUS** exportiert

# MySQL Cluster – NDB Engine

- **Hochverfügbarkeit**
- **Skalierung mit Lastverteilung**



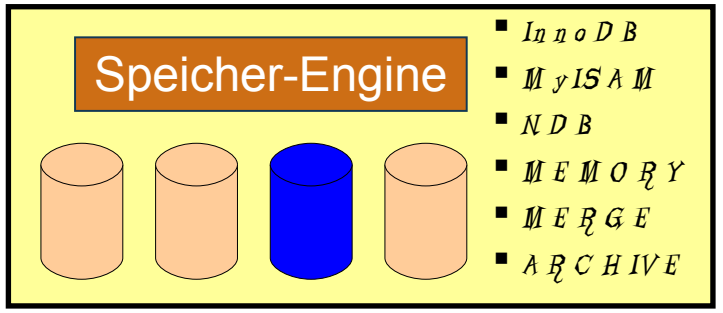




## Funktionen

- Synchrone Replikation
- Two-Phase Commit
- Automatisches Failover
- Replikation auf Ebene der Speicher-Engine

# Optimierung für MySQL Cluster

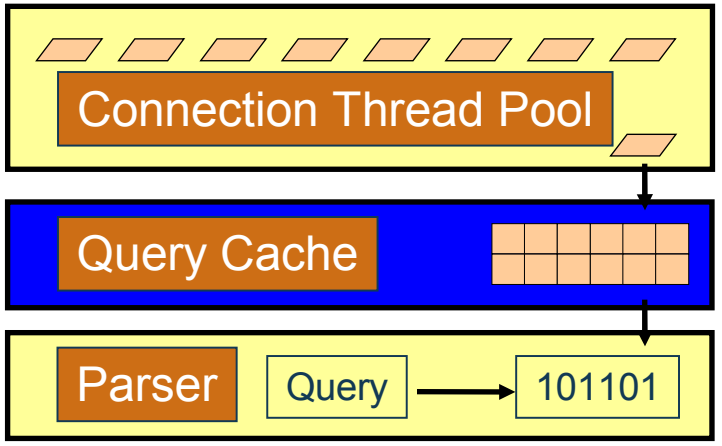
- **`ndb_force_send` (1)**
  - **ndb_force_send**=1, direkte Versendung von Nachrichten, optimiert in Bezug auf “Latency”
  - **ndb_force_send**=0, Sammeln der Nachrichten, optimiert für Durchsatz und bessere Verteilung der “Latency”, kann bei vielen gleichzeitigen Zugriffen von Vorteil sein
- **`ndb_use_exact_count` (1)**
  - **ndb_use_exact_count=0** setzen, um “Table Counts” für den Optimizer “vorzutäuschen”
- **`ndb_use_transactions` (1)**
  - **ndb_use_transactions=0** setzen, wenn eine größere Menge an Daten über eine Dump-Datei eingelesen werden



# Server “Query Cache”

## Parameter my.cnf:

- **query_cache_size (0)**
  - Speichermenge für Query Cache
  - Typisch **32M**, manche Datenbanken benötigen **128M**
- **query_cache_type (ON)**
  - Im schlechtesten Fall 13% Performanz-Overhead
  - Bevorzugt für Server mit höherem SELECT/WRITE Verhältnis




## mysql> show status;

- **Qcache_hits, Qcache_inserts**
  - Wenn Hits/Insert-Verhältnis klein ist, Deaktivierung des Query Cache
- **Qcache_lowmem_prunes**
  - Indikator, wie oft “alte” Ergebnisse wegen zu wenig Speicher gelöscht wurden. Bei hohem Wert Erhöhung von query_cache_size

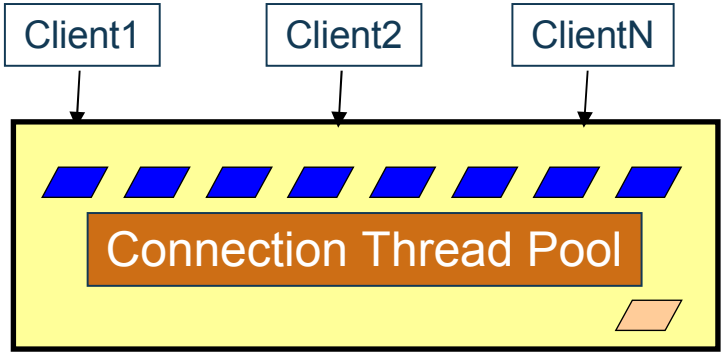
# Verbindungsbezogene Zwischenspeicher

**Parameter my.cnf:**

- **sort_buffer_size (2M)**
  - Für Sortierung zur Verfügung gestellter Speicher. Festplattenspeicher wird für größere Datenmengen genutzt. Oft reichen **512K** oder **1M**
- **andere Zwischenspeicher**
  - **read, read_rnd, etc…** kleine Standardwerte meist ok

Abschätzung: **Thread Speichernutzung** =
**max_connections** x
(connection buffers) x 1/3

- Oder einfach ~1GB für 1000 Verbindungen



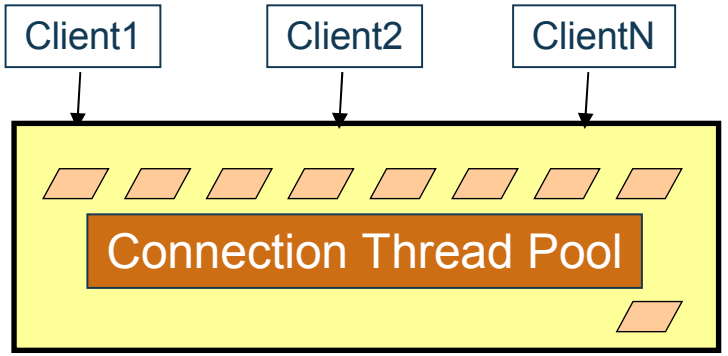

**mysql> show status;**

- **Sort_merge_passes -**
  - Anzahl Durchläufe während “File Merge”-Sortierung.
  - Überprüfung, ob “File Sort” nötig ist
  - Index nutzen, wenn möglich

# Server-Verbindungen & Threads

## Parameter my.cnf:

- **max_connections (100)**
  - Anzahl Verbindungen, die der Server erlaubt. Durch einen zu hohen Wert könnte der Speicher überschritten werden.
  - Typisch **1000**+ für Web-Cluster
  - 2-facher Wert des Durchschnitts-werts als Freiraum
- **thread_cache_size (8)**
  - Anzahl gehaltener Threads nach Verbindungsabbau
  - Typischer Wert **max_connections / 3**




## mysql> show status;

- **Max_used_connections**
  - Wird **max_connections** erreicht oder bei weitem nicht?
- **Threads_created**
  - thread_cache passend?
  - Sollte gering sein.

# Hardware und Betriebssystem

- Performance und Warteschlangen
- Richtig benchmarken
- Ansatzpunkte für Optimierungen
  - Architektur
  - Schema Design
  - Indizierung
  - Server Tuning
    - Globale Puffer
    - Threadlokale Puffer
    - Andere Parameter
  - → **Hardware und Betriebssystem**
- MySQL 5.1

# Datenbanken und CPU

- CPU – 64 Bit
  - MySQL ist eine Einprozeß/Multithread-Architektur
  - Wir haben 2006: Eine 3 GB Grenze können wir uns nicht leisten.
- Speicher-Bandbreite
  - Häufiger Engpass für CPU bezogene Last
  - Schneller Speicher, Dual Channel Speicher, dedizierter Bus für SMP
- Anzahl CPUs
  - Jede Verbindung verwendet eine CPU
  - Mehrere Abfragen skalieren gut bei Multi-CPU Maschinen
- Dual Core – Besser als Hyper Threading
- Datenbanken sind in der Regel nicht CPU-Bound
  - Wenn doch, ist die Architektur fragwürdig.

# Datenbanken und RAM

- Daten und Indizes werden auf Datenbank- oder Betriebssystem-Ebene zwischengespeichert
- Automatisch zur Verfügung, normalerweise vorhanden
  - Einstellungen für MySQL Server und Betriebssystem
- Arbeitsdaten sollten in den Speicher passen
  - Locality of Reference ist wichtig
  - RAM-Sättigung wäre noch besser

# Datenbanken und Platten

- 125-200 Seeks pro Disk und Sekunde
  - Viele Platten kaufen, nicht teure Platten kaufen!
  - RAID 10, denn RAID 5 ist Gift für Datenbanke-Writes.
  - Slaves können für bessere Performanz mit RAID0 arbeiten
- batteriegesicherter „Write Cache“
  - ACID Transaktionen ohne langsame Platten

# Datenbanken und Storage

- Wide Thin Disk Striping lohnt ab 6 Platten
  - Ist ab 10 Platten immer Vorteilhaft
  - Bei kleinen Konfigurationen gelegentlich dedizierte Disks für InnoDB Logs sinnvoll
- SAN/NAS gelegentlich fragwürdig
  - Seek-Performance oft nicht garantiert
  - lokaler Storage vielfach bevorzugt

# Datenbank und Betriebssystem

- MySQL unterstützt eine breite Palette an Plattformen
  - Linux, Windows, Solaris sind die meist genutzten
    - Alle drei funktionieren gut
- MySQL Pakete sind von Vorteil
  - RedHat, SuSE, Debian – die Häufigsten
    - Tarball oder RPM von http://www.mysql.com
- Betriebssystem-Support ist wichtig
  - MySQL lebt von gutem Thread-Support
  - Linux: NPTL
  - Ältere FreeBSD, NetBSD hatten einige Probleme
- 64 Bit Hardware, 64 Bit Betriebssystem
  - Wir haben 2006, wir können uns keine 3 GB Grenze leisten
- Treiber? Qualität der Treiber?

# MySQL 5.1

- Performance und Warteschlangen
- Richtig benchmarken
- Ansatzpunkte für Optimierungen
  - Architektur
  - Schema Design
  - Indizierung
  - Server Tuning
    - Globale Puffer
    - Threadlokale Puffer
    - Andere Parameter
  - Hardware und Betriebssystem
- **MySQL 5.1**

# Neue Performanzfunktionalitäten in MySQL 5.1

- Tabellen- und Index-Partitionierung
- Verbesserungen für
  - Volltext-Index
  - Archive Engine
- Schnelleres ALTER TABLE
- Schnelleres ADD/DROP INDEX
- Paralleler Datenimport
- Neues Lasttest-Werkzeug
- MyISAM Memory Option
- Neue Prozeß- und SQL-Diagnostik

# Quellen

- MySQL Online Handbuch – eine gute Informationsquelle
  - http://dev.mysql.com/doc/mysql/en/index.html
- SysBench - Benchmark und Stress-Test Werkzeug
  - **http://sourceforge.net/projects/sysbench**
- MySQL Benchmarks Mailing Liste
  - **benchmarks@lists.mysql.com**

# Fragen & Antworten

**Weitere Unterstützung zu Leistungsoptimierung:**

**www.mysql.de/consulting**
**www.mysql.de/training**

**Kristian Köhntopp**
**<kris@mysql.com>**
**http://blog.koehntopp.de**
**http://mysqldump.azundris.com**