

# Scaling and Performance Issues in MySQL



Peter Zaitsev,  
MySQL AB

DBF 2003  
03-10-2003

# About the Speaker

## **Peter Zaitsev, Benchmark Project Coordinator**

- 4 Years experience using MySQL on mission-critical systems,
- Co-founder of the Myrix/SpyLOG web project,
  - One of the largest MySQL users in Russia
  - Over 1Tb of data, over 100,000,000,000 rows
- Joined MySQL AB in April 2002
- Wear many hats, involved in
  - Support,
  - Development,
  - Consulting,
  - Certification

# MySQL AB - the Company Behind MySQL

- MySQL, is Open Source AND owned/developed commercially
  - paid developers, centrally managed
- MySQL AB owns the trademark MySQL and the copyright to all code
- Lots of community contributions
- Virtual company, Offices in USA, Sweden, Germany
- Many (mostly developers) working from home, 15+ countries
- Incorporated in Sweden

# MySQL Popularity & Users

- MySQL is popular
  - Over 35.000 downloads per day
  - Estimated 4.000.000+ installations
  - Shipped with Linux, Solaris, Novell
  - Mentions by Google: 15.1 mio. (vs 11.9 Oracle)
- MySQL is used by
  - Yahoo, Google, Slashdot, Linux.com
  - Ericsson, Alcatel, Nokia
  - Cisco, Compaq, Hypperion
  - Danish Center for Biological Sequence Analysis

# MySQL Products & Services

- MySQL Server
  - 4.0 Stable
  - 4.1 Alpha (Unicode & Subselects)
  - 5.0 Pre-Alpha (Stored Procedures)
- MaxDB Server
  - Result of MySQL partnership with SAP
  - New name for SAP DB, Q4 2003
- MySQLCC, Connector/J, MyODBC
- Embedded server
- Support, Consulting, Training, Certification

# Table of Contents

- Benchmarks used by MySQL AB
- Measuring performance
- Optimizing for MySQL
- Migrating to MySQL
- Badger project

# MySQL Benchmark Project at a Glance

- Regression test: mysql-test
- Database functionality & limits test: Crash-me
- Single-user benchmark: sql-bench
- Multi-user benchmarks
- Stress load benchmarks

*And a lot of extension plans*

# Types of Database Benchmarks

Many splits possible, this is just one of them

- Synthetic benchmarks
- Real application emulation

Both have their applications

We do Synthetic benchmarks mostly, helping customers to run their application benchmarks

# Synthetic Benchmarks

- Pros
  - Gives overall impression of performance
  - Many ready-made benchmarks available
  - Easy to modify
  - Simple to analyze - great for server developers
- Cons
  - Hard to ensure it is good to represent your load

# Real Application Emulation Benchmarks

- Best to check application performance & scalability
- Complex to create
- Complex & expensive to setup
- Results are harder to analyze
- Load pattern creations are critical for accuracy

As these are customers specific, customers usually run them, sharing results

# Single User Benchmarks

There are reasons to have single user benchmarks and we use them actively in MySQL

- Easy to develop
- Easier to get repeatable results
- Great to test database optimizations.
  - Is `count(distinct x)` faster with temporary table or merge sort?
- Easy to profile server
- Show peak performance

# Database Benchmark Architectural Choices

- Various architectures possible.
- Each architecture serves a particular purpose
- The question you want to answer defines architecture

# Single vs Multiple Physical Clients

- Single client is easy to setup & run
- Single client does not need complex hardware
- Multiple clients is typical real world scenario
- Multiple clients also tests network infrastructure
- Same host for database & benchmarks
  - No network overhead
  - Client load affects performance if it is large

We run mostly single client benchmarks at the moment, on the same box and over network

# Purely Database vs Application Stack

- Application stack benchmarks - all bottlenecks
  - client code, application level locks etc
- Database benchmarks - database part in total performance.
- More actual result vs simple analysis & setup
- Database benchmark - check possible optimizations before implementation

We sometimes run Application Stack benchmarks to check our Client API implementations

# Zero Delay vs Real Delay Terminal Emulators

- Zero delay terminal emulators are simpler
- Think time delays are what we have in the real world
- Think times affect performance
  - Locks
  - Many connections
  - Different amount of queries running at the time
- Running without delays to see the peak speed

We use zero delays in most cases now

# Limited Time vs Limited Set of Work

- Limited time - TPM or max clients are measured
- Limited set of work - time is measured
- Some benchmarks you can run in both modes
- Limited set of work - for batch jobs, non uniform load
- Limited time - good for stable query mix.

# Same Operation vs Query Mix

- AS3AP & TPC-C are examples
- Mixes are reality
- Single operations are easier to understand and analyze
- Both are great for different needs

We use both

# Dependent Benchmarks vs Independent Benchmarks

- Dependent: using previous benchmark result
  - Expose side effects
  - You can't rerun single benchmark
  - Save your time generating data
  - Usually used in multiple-tests suites.
- Independent - Each benchmark on its own

We have both in our suite. Large portion of single user benchmarks we have is dependent

# “Cold” vs “Hot” State Benchmarks

- Hot - The normal operation
- Cold - Operation after start
  - This can be bottleneck
- How to simulate these
  - Hot - need proper warm up load
  - Cold - make sure to clean up all caches.
- For multi user benchmarks you often get both in one run

We normally use cold start for single user benchmarks

# Why do the Results Differ?

- Various OS timing/scheduling issues
- Network/IO load being uneven
- Background external or database tasks
- Measuring inaccuracy
- Differences in layout, query mix

# Getting Usable Results

- Run test several times to know accuracy.
- Measure operating system/database performance counters during the run
- Allow system to settle down before the tests
- Avoid side load
- Measure results with good accuracy
- Use decent test time

# Layers and Subsystems

- Layers
  - Application
  - Database
  - Operation system
  - Hardware
- Each having
  - Resource requirements for levels below
  - Internal resource requirements (ie locks)
  - Requests coming from previous level

*This is one of the possible models*

# Measurement Layers

- Internal measurements
  - used to judge performance of particular layer(s)
  - cache hit rating is good example for database
- External measurements
  - Is layer loaded efficiently ?
  - How does it loads other layers ?

Use measurements to look for the reason, identify the best optimization strategy

# Measurements Example for DBMS Server

- External (higher level)
  - Transaction rate
  - Latencies
  - Time to complete job
- Internal
  - Primitive operations (handlers)
  - Cache statistics
  - Wait statistics
- External (lower levels)
  - - Disk IO, CPU Load
  - - Network Load, OS Waits, System call rate

# Optimizing for Best Performance

- Application architecture
- Database Schema
- Server configuration
- Operation system
- Hardware

MySQL is Open Source so you can also optimize server itself !

# Optimizing Your Hardware for MySQL

- Memory is usually the best investment
  - Note however limits of 32bit boxes
- 64bits to address large memory
- Fast CPU or Several ones (depends on the load)
- Do not discount cache/memory/bus speed importance
- IO subsystem critical for large data sizes
- RAID with battery backup for transactional load
- Network connection - latency is critical

# Tuning the Operating System

- Select proper OS
  - Must have good threads support
    - Kernel level threads are best
  - Linux & Solaris are currently the best
- Set OS to allow large limits (threads, files, per process memory)
- Adjust network parameters
  - Socket buffers
  - Timeouts
  - Hash sizes

# Tuning the Operating System

- Delayed disk writes are good
  - Allow large amounts of dirty buffers
- Tune IO scheduler to match the load
- Try Direct IO & Raw partitions with Innodb
- Mount file system in noatime, writeback journaling mode
- Turn off swap partitions or lock MySQL in memory

# Server Optimization

- Perform after application & schema are optimized
- Check performance counters (***SHOW STATUS, SHOW INNODB STATUS***)
- Compare them to OS counters (***vmstat, mpstat***)
- Adjust memory consumption
  - Swapping would kill performance
- Use proper disk layout
- Use MySQL specific features
  - Query cache
  - Bulk insert buffering, delayed key writes

# MySQL Tune-ables

- **key\_buffer\_size** - MyISAM index cache.
- **read\_buffer\_size,read\_rnd\_buffer** - MyISAM per thread row buffers
- **sort\_buffer\_size** - Memory used for sorts, disk afterwards
- **table\_cache** - Amount of table descriptors held open at one time
  - Do not forget to increase open files limit
- **tmp\_table\_size** - In memory table size, will go to disk after
- **thread\_cache** - Cache threads on disconnect
- **query\_cache\_size** – Memory to use for caching query results
- **max\_connections** - Maximum of connections server allows
- **innodb\_buffer\_pool\_size,log settings** - critical for Innodb performance

# MySQL Performance Counters

- Performance counters from **SHOW STATUS** command
- **Com\_XXX** - commands executed
- **Handler\_XXX** - low level operations
- **Key\_XXX** - keycache statistics
- **Created\_tmp\_disk\_tables** - disk temporary tables
- **Sort\_merge\_passes** - disk based sort
- **Qcache\_XXX** - query cache statistics
- (No performance counters for Innodb yet, in ToDo)
  - Use **SHOW INNODB STATUS**

# Eweek Benchmark

- Done in January 2002
- Alpha version MySQL
  - Was optimized further
- According to eWeek, the first independent full scale test of RDBMS in 7 years
- Windows based

## Oracle9i and MySQL top throughput



Throughput is in returned Web pages per second from the application server. Number of users is number of concurrent Web clients driving the load. Response time is the time to complete the six bookstore user action sequences, weighted by frequency of each sequence in the mix. All tests were conducted on an HP NetServer LT 6000r with four 700MHz Xeon CPUs, 2GB of RAM, a Gigabit Ethernet Intel Corp. Pro/1000 F Server Adapter and 24 9.1GB Ultra3 SCSI hard drives used for database storage.

# Database Schema

- Use proper storage engine and options for tables
  - **TYPE=MYISAM/INNODB/HEAP**
- Check your indexes
  - All the ones you need, and only the ones you need  
*The key, the whole key, and nothing but the key, so help me Codd!*
- Avoid index duplicates
  - And same prefix keys: **key(a), key(a,b)**
- Avoid long primary keys for Innodb
  - Use **auto\_increment** and **UNIQUE** instead
- Consider prefix indexes
  - **key(col(20))**

# Application Optimization

- Use replication to get scalability and HA
  - Manual partition/copying is better in some cases
- Use MySQL specific features if they help
  - Multi value **REPLACE/INSERT**
  - **INSERT DELAYED**,
  - Multi-Table updates,
  - Temporary tables,
  - In-memory tables (**TYPE=HEAP**),
  - **LIMIT**
- More caching on the application side
- Summary tables can be very good help
- Normalize/de-normalize table structures

# MySQL 4.1 Performance Improvements

- New binary client-server protocol (no data conversions)
- Server level Prepared statements
- New key cache (less locks, intelligent LRU)
- Multiple key caches, Index pre-loading
- Multi-table updates
- Multiple tablespaces in Innodb
- Multiple commands in single statement
- **ON DUPLICATE .. UPDATE** in Insert

# Connecting to the MySQL Server

- APIs developed/maintained by MySQL AB
  - Native C API
  - C++ API (MySQL++)
  - MyODBC (ODBC Level 3.5)
  - JDBC
- Contributed/Commercial APIs
  - Perl DBI, PHP, Python
  - .NET, OLEDB
- MySQL APIs are available from third parties for many languages
- MySQL is supported by many database abstraction APIs

# Behavioral Differences

- No data error check, converting instead
  - -1 stored in **UNSIGNED INT** will turn to 0
  - MySQL 4.1 will generate warning
- VARCHAR removes trailing spaces (use BLOB)
  - Scheduled to be fixed
- Silent changes: VARCHAR/CHAR conversions possible
- Table names are case sensitive on Unix/Linux by default
- Transactions are for InnoDB/BDB tables only
- ALTER locks and rebuild entire table

# Making MySQL Behavior More ANSI-like

- Running **--ansi** will resolve some issues
  - “||” will be used for concatenation
  - Only “ ” will be allowed for quoting
  - Serializable isolation level by default
- Running **--lower\_case\_table\_names**
  - Make tables case insensitive
- **default\_table\_type=INNODB**
  - Get transactions & Foreign keys for default table type

Options for full SQL:2003 compliance are in TODO

# Workarounds for Missing Functionality (We're Working on emptying this ASAP)

- Stored procedures
  - Use multiple queries, it is fast
  - Use embedded server library
  - UDF can help in some cases.
- Subqueries
  - Use MySQL 4.1 for new development
  - Rewrite query to **JOIN** if possible or use **TEMPORARY** tables.
- Foreign keys
  - Use **INNODB** table type
- Constraints
- - Check data on application level

# Think the MySQL Way

- Check data on application side.
  - Feel free to run many simple queries
    - Dual Opteron can do more than 55.000 trivial queries/sec
  - Working with BLOB data is simple & fast!
  - Use **LIMIT** instead of **CURSOR**, request only the data you need
  - Use the correct storage engine for the task at hand.
    - They can be mixed at will!
  - Use MySQL specific features if they make your life easier
- MySQL: Practical solutions, simple to use

# Badger

## Performance of MySQL on Modern Computing Platform

(Itanium-2, Multiprocessors, Clusters, SAN)

1. Analyzing Performance on a given Linux platform
2. Improving Performance (when possible)
  - a. How to configure/tune on a given platform?
  - b. New tuning parameters (e.g., prefetch size)
  - c. New mechanisms (e.g., read ahead prefetching)

Member of the Gelato Foundation – <http://www.gelato.org/>  
A collaboration of DIKU, MySQL, HP, Dell/EMC

# Analyzing Performances

- Internal Analysis
  - Instrumenting MySQL: Where is the time spent?
    - Physical, logical IOs
    - Locks
  - Program Path Analysis
- External Analysis:
  - IO Subsystem: Number of pending requests, Disk Scheduler Performance
  - CPU: Inspecting performance registers (Vtune, Oprofile)
- Simple set of experiments for micro-benchmarking MySQL:
  - Disk Subsystem Utilization
  - Ratio cost of physical vs. logical IOs
  - Cost of operations in-memory
  - Cost of parsing vs. query optimization vs. query execution
  - CPU Utilization
  - Cost of latching / locking

# Disk Subsystem Utilization

- Asynchronous IOs on Linux. Optimal disk performance (PIII, 10000rpm SCSI):
  - Sequential ops: 4 outstanding requests
  - Random ops: 2000 outstanding requests
- Micro-benchmarking of MySQL shows:
  - Sequential reads: Nb of outstanding requests falls to 0 repeatedly. Read ahead prefetching is the culprit.
  - Random reads/writes: Nb of outstanding requests not higher than 4.
- Improving Performances:
  - Eager read ahead mechanism (with prefetch size as a tuning option)
  - Prioritized IOs in Linux to control latency while achieving high throughput when increasing the number of outstanding requests.

# Badger Activities

- **Disk Subsystem Utilization**
  - Prefetching
  - Communication between SAN front-end and MySQL/InnoDB
- **MySQL on Cluster**
  - MySQL client for load balancing over replicated MySQL servers
  - Replicated MySQL over 100+ servers
- **Workload Generation**
  - Given an app, how to synthesize a representative workload for commercial benchmarking?
- **Specific Platforms**
  - Optimizing register utilization on HP Itanium-2
  - Optimizing multiprocessor usage on SGI Altix

## Final Word

- Thank you for Listening !
- MySQL Web site: <http://www.mysql.com>
- MySQL Documentation <http://www.mysql.com/doc>
- MySQL Products: <http://www.mysql.com/products>

Have questions ?

Contact me during the conference or  
at [peter@mysql.com](mailto:peter@mysql.com)