

# The MySQL Benchmark and Testing Project



Peter Zaitsev,  
MySQL AB

MySQL User Conference 2003  
2003-04-10

# About Speaker

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

- 4 Years of mission critical MySQL experience.
- CoFounder of Myrix/SpyLOG web project,
  - One of the largest MySQL users in Russia
  - Over 1Tb of data, over 100,000,000,000 rows
- Joined MySQL in April 2002
- Wears many hats, involved in
  - Support,
  - Development,
  - Consulting,
  - Certification

# Project Team

- Peter Zaitsev
- Alexey Kishkin (Walrus)
- Alexey Stroganov (Ranger)
- Rest of MySQL team
- External contributors

# Alexey Kishkin (Walrus)

- sql-bench suite development
- Crash-Me tool development
- Quality Assurance tools development
- Database Expert
- Documentation development

# Alexey Stroganov (Ranger)

- Benchmark project web pages development
- Multi User tests development (AS3AP)
- Porting OSDL suite to MySQL
- DBI/DBD Maintenance
- Data Generation scripts developer

# Informal Members: MySQL Team & External contributors

- Professional Services
- Feedback
- Test Case Ideas
- Code contributions
- Contributions of hardware for tests
- Running the tests by themselves

You are welcome to Join !

# Goals of the Project

- Tools for fair comparison of different databases
  - performance, features, stability
- Tools to help database selection and optimization to match customer load
- Provide tools for generic system performance comparison.
  - To help kernel and system software developers
- Spotting bottlenecks databases might have
- Eliminating bottlenecks in MySQL server
- Quality Assurance for MySQL Server, release testing.

# **We can't totally deny Bias but we try to keep it down**

- Our tests are open source
- We also use industry standard test cases
- We benchmark databases in default configuration
- We specify configuration clearly so tests can be repeated
- We're open for discussions and keep them public
- You're free to run your own benchmarks and publish them.

# Goals of MySQL product development

- Performance
- Stability/Reliability
- Usability (Ease of Use / Convenience)
- Features

We target the two top items with our project.

# Project Role in MySQL product development

- Performance
  - No performance bugs with new changes
  - New features are fast enough
  - Optimizations made really improve things
- Stability
  - No bugs are added during code improvement
  - New code is properly tested
  - Release successfully handles stress load

# Performance

- Regression performance testing by sql-bench suite.
- Adding test cases for new features
- Stress testing
- Finding bottlenecks
- Search for possible optimizations
- Research the best options for different load types

# Stability

- Regression testing by MySQL test suite
- Improving MySQL test suite to allow wider range of tests.
- Code coverage to make sure code is covered properly
- Valgrind/Purify runs to catch floating errors
  - Many thanks to Julian Seward for Valgrind
- Stress testing to measure behavior under load
- Testing MySQL in critical situations
  - Out of Memory condition
  - Disk full
  - Unexpected Power loss
- Crash-Me run

# MySQL Benchmarks Project Structure

- mysql-test - Regression testing suite
- sql-bench - MySQL benchmark suite
  - single-user tests
  - Multi-user tests (AS3AP)
  - Real load emulation tests
- crash-me - Feature comparison tool
- Code coverage monitoring tools
- Result Database
- Benchmarks Web Pages
- Benchmarks Mailing list

# MySQL-Test current state

The suite is designed to automate testing for MySQL server. We keep it simple

- Test-case is written in simple text form
- Test-case is executed and result compared to one in the database.
- Some required scripting features present
- Difference in expected and actual result is displayed.

# MySQL-Test future development

- More tests are added for better code coverage.
- Test Definition Language is being extended
- Add Windows Support
  - Do not use /bin/sh
- To be extended to test client code and programs as well
  - (done via separate test suites now)

# SQL-Bench current state

- Single user benchmark
- Small data set
  - Usually fits in system memory
- Measuring peak performance of particular operations
  - Insert, select on index, join, table scan
- Checking database internal algorithms and overhead
  - CPU efficiency
- Testing which optimizations database has in place
- Most tests do not represent typical real life load

# SQL-Bench Future Development

- More intelligent and extensible engine
- Multi-User tests.
- Real life inspired test scenarios
- Scalable tests
  - Database size
  - Concurrent number of clients
- User Adjustable tests
  - data distribution
  - operations
  - operations mix (ie readers/writers ratio)

# SQL-Bench planned Engine changes

- High resolution timer
- Support of different types of tests and results
- Performance data gathering
  - CPU load
  - Disk IO
  - Operation System activity
- Possibility to use prepared statements
- Improved result accuracy

# SQL-Bench planned New Tests

- Benchmarks for New MySQL features
  - Subqueries
  - Stored procedures
  - Foreign keys
  - Unions
  - Full text Search
  - GIS Functionality
- - Tests for new MySQL optimizations
- - Tests for discovered bottlenecks
- - Multi user tests for existing functionality
- - Real-life test scenarios

# Crash-me, mysterious tool

- Database independent, most well known databases supported
- Testing present database features
- Spotting implementation differences
- Discovering database limits, including
  - query/result size limit
  - maximum row length
  - maximum number of indexes
- Database stability on unusual queries.
- Other database properties identification

## Crash-Me future

- Extending test set to test more features
- Unexpected result debugging feature
  - Basically logging queries and responses for test
- - ANSI SQL-99 standard compliance testing
  - Based on Peter Gulutzan work
- Results for more databases with various run options
- New convenient Web based result viewer

# Code Coverage analyses tool

- Currently used only within the company
- Code covered by the tests has much less chance to be buggy
- Code coverage generated by mysql-test suite is tested
- Code which can't be covered can be marked.
- Integration with BitKeeper
- Code coverage may be reported
  - by developer
  - by file
  - for code created particular time
- Policy: All new code shall be covered by tests

# Results Database

- Open Results database
- Web viewer is under development
- Submit your tests & results
- Main results to be found in the database
  - MySQL vs Other RDBMSs
  - MySQL performance on various platforms
  - System configuration profile
    - kernel
    - file system
    - disk subsystem configuration
  - MySQL options.

# Benchmarks Web Pages: future

- Convenient Result Viewing
- Crash-Me results viewer
- Articles section
- Benchmarks forum
- Links section
- Visitors will show us the way for future development.

# Benchmarks Mailing list announce [benchmarks@lists.mysql.com](mailto:benchmarks@lists.mysql.com)

- Subscribe at <http://lists.mysql.com>
- Main topics for the list:
  - Benchmarking Databases
  - Performance differences between databases
  - Databases performance tuning
- Benchmarks sources and results welcome
- Discussions about any databases are welcome

# How to run MySQL benchmark suite?

- Download and install MySQL binary distribution
  - If you prefer RPM – make sure MySQL-bench-<version>.rpm is installed
- Make sure Perl, DBI and DBI-mysql are installed
- You can download them from <http://www.mysql.com/downloads/api-dbi.html>
- Locate "sql-bench" directory in your installation directory
- Run `./run-all-tests`

# Performing Benchmarks

- System should be as idle as possible
- Use dedicated partition or at least a lot of free space
- Several runs performed to estimate accuracy
- MySQL server is restarted after each run
  - To flush internally buffered data
- Partition is mounted/unmounted to clean file cache.
  - Different OS may require different tricks for it.

# Different results for different goals

- Time taken to perform the test
- Transaction rate (transactions per second/minute)
- Max and average time taken per transaction
  - (critical for interactive applications)
- Max Number of concurrent clients while meeting response time requirements

# Performing benchmarks: Is more complex than you might think

- Very many unexpected variables affecting performance
  - ReiserFS journaling options can give major difference
- Very detailed descriptions needed to make results repeatable
- Some tests may have too bad relative accuracy making hard to compare
- Time for some tests may differ 100+ times
  - Depending if database optimizes some cases or not
- Making it too long for one database and too fast and inaccurate for others

# Making results repeatable

- Detailed description of test system is required
- Very small points might matter
  - Compiler or options which were used for kernel compilation
- Using official MySQL binary.
  - Compiler and GLIBC version may give 10%+ difference
- Consistent benchmarking policy
  - Be aware of data caching by Operation system
  - Be aware of data caching by database
  - Be aware of storage degradation effects
    - File system fragmentation
    - Tablespace fragmentation
    - Table fragmentation

# Interpreting Benchmark results

- Make sure you clearly understand what is being benchmarked
- Is this the optimal way to solve the task for this database ?
- Take scale and options into account
- Think of which way this test corresponds to your application
- Take accuracy into account

# Creating your own benchmarks

- What is purpose of the benchmark ?
  - (application emulation, stress test, test of some functionality)
- Single database or multiple databases ?
  - Different databases may solve the same task in different ways
- Are you stuck with ANSI SQL or will you use database specific features ?
- What are the main variables you need to check ?
  - (data size, concurrency, data distribution)
- Select result type to match your goal
- Select length of benchmark to match required accuracy

# Benchmark project community works

- Linux Kernel
- DBI/DBD Perl API
- PHP API
- OSDL
- Academic & Research projects
- Other OpenSource projects

We're taking an active part in Benchmarking OpenSource projects and projects which can help to improve MySQL performance

# Linux Kernel

- File Systems
- Disk IO (especially Direct IO, Async IO)
- Scheduling
- Threads Support

Performance, Stability, Scalability for MySQL.  
Communication, Testing, Advice - no coding yet.

## DBI/DBD MySQL driver

- Is now maintained by Benchmark Team member, Ranger
- Support for true Prepared statements
- Support for Binary communication protocol
- Support for New MySQL 4.1 functionality
  - charset, multi query execution, etc

# PHP API

- PHP 5.0 to feature new MySQL API
  - Implemented mainly by Georg Richter
- Support for Prepared statements
- Support for Binary communication protocol
- New MySQL 4.1 functionality
  - charset, multi query execution etc
- A lot of fixes, ie persistent connections
- Up to 300% performance boost in some tests

# OSDL

## Open Source Development Laboratory

- OSDL Released benchmark suites which are based on
  - TPC-W
  - TPC-C
  - TPC-H
- We're to port them to MySQL to have results of independent benchmark suite

# Other OpenSource projects

- NGPT Next Generation Posix Threads
- Glibc (LinuxThreads mainly)
- NPTL Native POSIX Thread Library
- OSDB (Open source C AS3AP suite)

# Academic & Research Projects

- Started Database performance research project by University of Copenhagen.
  - lead by Philippe Bonnet
- Research is done in areas:
  - Instruction cache utilization.
  - Disk subsystem memory hierarchy.
  - Clustered, distributed databases

We welcome other research projects using MySQL

# Working together with the Benchmarks team

- We need your good ideas
- You're free to extend the existing benchmark suite
- We also welcome independent tests developed
- You may wish to run benchmarks and share results.
  - (This is not limited to MySQL database)
- Provide your hardware to run benchmark on.
  - (We just need remote access for it)
- Other partnership you might think of

**We'll be able to do more together!**

# Interesting Benchmark Results

- MySQL 4.1 improved keycache
- MySQL 4.0 bulk insert feature
- Speed of FullText indexes
- A few facts about MyISAM vs InnoDB
- innodb\_flush\_logs\_at\_trx\_commit speed effect
- SSL performance
- Various Client APIs
- Please take results as illustration, rather than hard numbers

# MySQL 4.1 KeyCache benchmark

- Improved internal concurrency
- AS3AP Random read test:
  - disk bound load,
  - Dual PIII System
- Results:
  - 5 users: 3.18 times improvement
  - 25 users: 25.8 times improvement
- Note: It is very special test case to spot the bottleneck

# MySQL 4.0 Bulk Insert Benchmark

- Bulk insert: thousands to hundreds of thousands records per single statement
- Optimization Idea is updating key blocks in sorted order to minimize disk IO.
- Bulk insert for 100,000 rows/statement
  - 10 times improvement, compared to standard insert
- Increasing 10,000 rows/statement to 100,000 rows/statement
  - performance gain: 1.6 times.
- Increasing `bulk_insert_tree_size` from 8M to 256M
  - Performance gain: 4 times

# Full Text Search performance facts

- MySQL 4.0 introduced Boolean FullText search. See the difference
- Selecting 10000 out of 1000000 rows, single keyword
  - natural language: 1 min 56.75 sec
  - boolean search: 36.31 sec
- Top 10 rows (LIMIT 10)
  - natural language: 1.09 sec
  - boolean: 0.05 sec
- Boolean search is faster, especially with LIMIT
  - Boolean search does not need to sort according to relevance

# InnoDB vs MyISAM performance

- 2Mb BLOB insertion.
  - InnoDB is about 3 times slower than MyISAM
- Natural JOIN using clustered (primary) key:
  - (Tables were sized to fit in memory completely)
  - InnoDB is 2 times faster than MyISAM.
- Result:
  - Both can be faster or slower for you. Benchmark your application.
  - There are cases where MyISAM is slower for read-only load

## Flush logs at COMMIT ?

- Some of recent Linux kernels trend to fake fsync() which can lead to data loss.
- Performance of single row update transaction.
  - innodb\_flush\_logs\_at\_trx\_commit=0 6100 tps
  - innodb\_flush\_logs\_at\_trx\_commit=2 5900 tps
  - innodb\_flush\_logs\_at\_trx\_commit=1 1650 tps
    - Disk write cache enabled
  - innodb\_flush\_logs\_at\_trx\_commit=1 **65** tps
    - Disk write cache disabled
- (battery backed up write cache can boost last number safely)

# How does using SSL affect performance of MySQL

- Speed change really depends on CPU speed, load type
- Few examples from MySQL benchmark suite run:
  - ATIS: 1.3 times slower
  - Connect: 8.4 times slower
    - (Yes. SSL handshake takes plenty of time)
  - Insert: 1.4 times slower
  - Select: 1.06 times slower
    - (Complex queries do not depend on communication speed a lot)

# MySQL performance based on client API used.

- Once again it is quite different for different loads
- We used simple selects load, time in seconds:
- MySQL C API            27/22
- Perl native MySQL 51/42
- Perl via ODBC            92/85
- Python                    80/66
- Java (JDBC)            45/-
  - (TCP/IP and UNIX socket times shown)

# Final Word

- Contacts:
  - Subscribe to [benchmarks@lists.mysql.com](mailto:benchmarks@lists.mysql.com) public list at <http://lists.mysql.com>
  - Benchmarks team private address: [benchmarks@mysql.com](mailto:benchmarks@mysql.com)
  - Contact me: [peter@mysql.com](mailto:peter@mysql.com)
  - Do not hesitate to catch me on the conference for personal talk

Time for Questions