MySQL Optimization
MySQL User Conference

Jeremy Zawodny
Yahoo!

April 12th, 2003
San Jose, California

http://jeremy.zawodny.com/mysql/

Copyright 2003, Jeremy Zawodny
About Me

- Engineer in Y! Search (prev. Y! Finance)
- MySQL user for over 5 years
- Active in MySQL community
- Write about LAMP for Linux Magazine
- MySQL advocacy & support at Yahoo!

Home: Jeremy@Zawodny.com
Work: jzawodn@yahoo-inc.com
http://jeremy.zawodny.com/mysql/
Outline

• Introduction
• Why Optimize?
• Goals
• Database Design
• Application Design
• Writing Fast Queries
• MySQL Server Tuning
• Operating System Tuning
• Hardware Tuning
• Network & Replication
• Where to Learn More
• Questions and Answers

Copyright 2003, Jeremy Zawodny
Starting Questions

• What version of MySQL are you using?
• What languages are being used?
• Which operating systems?
• Familiarity with other RDBMS servers?
• Role? DBA? Developer? SysAdmin?
• MySQL dedicated or shared servers?
• How fast is your growth?
  – Transaction rates
  – Data volume
What you Need to Know

• You should ask questions at any time
  – There should be sufficient time

• MySQL usage
  – Basic queries (SELECT, UPDATE, INSERT)
  – Installation or where files are located

• Basic programming concepts
  – Any language will do

• Operating system basics
  – Memory usage, swapping, etc.
MySQL at Yahoo!

- Roughly 200-400 servers world-wide
- FreeBSD and Linux
- Commodity hardware
- Replaces home-grown “database” systems
- Replaces Oracle in a few cases
- Typical install uses between 1-20GB
- Used both “live” and in batch processing
- Replication and load-balancing
Why Optimize?

• You can do more with less
  – MySQL on “normal” hardware scales well
  – A little time can save thousands in hardware
  – The classic story goes…

• As you data grows, you’ll need to
  – Performance will degrade over time
  – You’re probably not monitoring it anyway

• It is easier than re-coding your apps

• Your users will notice if you don’t!
MySQL’s Defaults

• Tuned for small and medium data sets
• Uses very little memory even if available
• Suitable for use in a shared environment
• Assumes little about your hardware
• Begins to slow as growth continues
• Uses non-transactional tables (MyISAM)
  – That’s what most people need (90%)
  – Very low overhead
Scaling MySQL

• Like Linux, MySQL scales up and down
• Can run many MySQL instances at once
• Can run one very big MySQL instance
• Can run with only a few MB of memory
  – Suitable for small devices
  – Will be disk-bound
• Can embed using libmysqld (MySQL 4.x)
• Can recompile to add/remove features
  – Table types, query cache, etc.
Using Less Hardware

• Hardware is rarely the bottleneck
  – Well-tuned servers are often disk-bound
• MySQL isn’t using it aggressively
  – You must configure it
• Modern CPUs are very fast
  – What you have is probably sufficient
• Memory is plentiful
  – You’re probably not using what you have
• Upgrades do little to solve most problems!
Goals

• Learn to write fast queries and applications
• Learn to design and use the right tables
• Know where to look for bottlenecks
• Predict behavior as load increases
• Understand what to monitor over time
• Understand how MySQL uses system resources
• Learn what settings you can adjust
  – In your operating system
  – In MySQL
  – In your applications
• Know where to learn more…
Database Design

• Normalize your data by default
  – Sometime you need to de-normalize
  – When in doubt, benchmark
    • MySQL super-smack
    • MySQL benchmark suite
    • Home-grown tools
    • Use your real apps!
Database Design

• Select the right column types
  – No bigger than you need
  – MySQL provides a ton of column types
  – Use NOT NULL where it makes sense
  – Use fixed column sizes if you can
    • MyISAM tables with fixed rows are faster
    • Concurrency improvements
  – Store compressed data when possible

See: http://www.mysql.com/doc/S/t/Storage_requirements.html
Database Design

• Select the right table types
  – What locking model do you need?
    • Table (MyISAM)
    • Row (InnoDB)
    • Page (BDB)
  – Consider ratio of reads to writes
  – Foreign key constraints?
  – Do you need transactions?
  – Can you afford to lose records in a crash?
  – Do you know MySQL’s table types?
Database Design

• MyISAM Tables
  – Very efficient
  – Compact storage
  – In-memory key cache for index data
  – Table locking
  – No transactions
  – Good for
    • High volume logging (write)
    • High volume reads
    • Not both
  – Variations: Compressed, RAID, Merge
Database Design

• Compressed MyISAM Tables
  – Read-only
  – Good for CD-ROMs and archives

• MyISAM RAID Tables
  – Break the 2GB/4GB/whatever barrier

• MyISAM Merge Tables
  – Many physically identical MyISAM tables
  – Can treat as a single table (or not)
Database Design

• HEAP Tables
  – Stored in memory
    • They will vanish at server shutdown
  – Very fast hash-based lookups
    • Limited index use
    • Range queries are slower
  – B-Tree available in 4.1
  – Table locking
  – Great for static lookups
  – Size can be limited to prevent disaster
Database Design

• BDB Tables
  – Transactional
  – Automatic recovery
  – Tables grow as needed
  – Page-level locking (8KB page)
    • Single READ-COMMITTED isolation level
  – Uses Berkeley DB under the hood
  – Few users actually use BDB
  – Works well for small - medium transaction rate
  – Locking on the last page can be a problem
Database Design

• InnoDB Tables
  – Modeled after Oracle
    • Row-level locking
    • Non-locking SELECTs
    • Uses pre-allocated tablespace files
  – Multiple isolation levels
    • Easily changed with a SET command
  – Referential integrity - foreign keys
  – High performance
  – Very high concurrency
  – Automatic recovery after crash
Database Design

• Use Indexes wisely
  – Don’t use several indexes when one will do
  – Understand the “leftmost prefix” rule
    • Index on (col1, col2, col3) vs. 3 indexes
  – Don’t index columns until you need to
  – Verify that indexes are used (difficult)
  – Use partial indexes on large (text) fields
  – Index a hash rather than a large value (URL)
    • MD5 is an excellent choice
    • It’s even built-in
Database Design

• Use full-text indexing if you need it
  – MyISAM tables only
  – Very fast
  – Excellent in MySQL 4.x
  – Results are ranked (like a search engine might)
  – Boolean queries
    • Flexible
    • Mostly feature-complete
  – Works on any textual data
    • Other character sets will need 4.1 or 5.0
Full-Text Search

• Use 4.0 if possible
  – Indexing is much faster
  – Stop word list customization
  – Min word size easily changed
    • Remember to rebuild indexes after changing

• In 5.0 we should see
  – Per-table stop word lists
  – Per-table word length options
  – Per-table word characters lists
  – These might be per-index!

Copyright 2003, Jeremy Zawodny
Application Design

• Don’t store data you don’t need
  – Compress it
  – Get rid of it

• Don’t store computable data
  – MySQL can do it
  – Your app can do it

• Don’t ask for data you don’t need…
  – Do you really need all fields?
    SELECT * FROM...
Application Design

• Use MySQL extensions for speed
  – REPLACE queries
  – Bundled INSERT queries
  – Multi-table deletes
  – User variables

• Use logging to track bottlenecks

• Don’t perform unnecessary queries
  – Cache data (static lookup tables)
  – Use the Query Cache if you must

• Benchmark your application
  – Know where the bottlenecks are
  – Know how a slow db affects your application
Application Design

• Use transactions
  – Prevents data loss
  – Server does less random I/O
  – Performance and reliability

• Keep the clients “near” the server
  – Network latency is a killer
  – Replication can solve geography problems
  – Can also help solve geology problems (quake)
  – Running app and MySQL on same hardware
Application Design

• Think about growth
  – There are size limits that you might hit
  – InnoDB and MyISAM both have them (sort of)

• Keep primary keys short for InnoDB
Application Design

- Use prepared queries and placeholders
  - MySQL doesn’t yet support them
  - Your API may
  - When MySQL does, you benefit!
  - The API may be more efficient anyway
  - MySQL 4.1 and PHP 5.0 benefit

```
SELECT name, address, state, zip
    FROM customers
WHERE id = ?
```
Application Design

• Web apps
  – Use (but don’t over-use) connection pooling
  – Use middleware to abstract the database
    • May also provide caching and pooling
  – Don’t keep everything in the database!
    • Images can live on the file system
    • But you might want to replicate them
  – Pick the fastest driver you can
    • Java has several, Perl has two
    • On Windows, use the “most native”
Break!
Writing Fast Queries

- Use Indexes
- Use `EXPLAIN` on SELECT
- Simplify where clause
- Watch Slow query log
- Bundle INSERTs
- UNIONs
Writing Fast Queries

• Understanding how MySQL runs queries
• You need to think like MySQL does
• Some of its goals are…
  – Eliminate as many rows as possible
  – Use indexes where possible
  – Avoid table scans
  – Consider many join orders
  – Avoid hitting the disk
  – Avoid using the data records if the index has it
Writing Fast Queries

• **EXPLAIN SELECT**
  – Tells you what MySQL is thinking
  – Which keys (indexes) can it use
  – Which keys will it use
  – How many rows must it examine (roughly)
  • **ANALYZE TABLE** can help
  – How hard must MySQL work?
Writing Fast Queries

- **EXPLAIN SELECT**

```sql
mysql > EXPLAIN SELECT * FROM Headlines H, S2H S WHERE S.Symbol = 'YHOO' and H.Id = S.HeadlineId;
```

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>ref</td>
<td>HeadlineId,Symbol</td>
<td>Symbol</td>
<td>75</td>
<td>const</td>
<td>383</td>
<td>where used; Using index</td>
</tr>
<tr>
<td>H</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>4</td>
<td>S.HeadlineId</td>
<td>1</td>
<td>where used</td>
</tr>
</tbody>
</table>

2 rows in set (0.00 sec)

```sql
mysql > EXPLAIN SELECT * FROM Headlines H, S2H S WHERE S.Symbol = 'YHOO' and H.Id = S.HeadlineId ORDER BY Time DESC;
```

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>ref</td>
<td>HeadlineId,Symbol</td>
<td>Symbol</td>
<td>75</td>
<td>const</td>
<td>383</td>
<td>where used; Using index; Using temporary; Using filesort</td>
</tr>
<tr>
<td>H</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>4</td>
<td>S.HeadlineId</td>
<td>1</td>
<td>where used</td>
</tr>
</tbody>
</table>

2 rows in set (0.00 sec)
Writing Fast Queries

• EXPLAIN SELECT
  – Table
    • Order is significant
    • Aliases appear
  – Type
    • System
      – Table has one row
      – Easily optimized
    • Const
      – Only a single row matches
      – Read once
Writing Fast Queries

• EXPLAIN SELECT (continued)
  – Type (continued)
    • eq_ref
      – One row matches per combination
      – Unique index match
    • ref
      – Several matching rows per combination
      – Non-unique index
    • range
      – A range of rows will be retrieved
    • index
      – Index will be scanned for matches
      – Like a table scan, but faster
    • all
      – Full table scan
      – Worst case
Writing Fast Queries

• EXPLAIN SELECT (continued)
  – Possible keys
    • What MySQL had to choose from
  – Key
    • What it decided to use
  – Key length
    • Length (in bytes) of the longest key
  – Ref
    • Which column it will match with
  – Rows
    • Approximately how many rows must be examined
Writing Fast Queries

• EXPLAIN SELECT (continued)
  – Extra information
  • Using filesort
    – An extra pass is required to sort the records
    – This can be slow at times
  • Using index
    – Data will come from the index rather than rows
    – This can speed things up
  • Using temporary
    – MySQL will create a temporary table
    – It’ll be a disk-based table if it’s too large
  • Where used
    – The where clause will be applied to this table
Writing Fast Queries

• Optimizer tips and tricks
  – It’s smart, but not perfect
  – Only one index per table per query
    • You *may* need to de-normalize to get performance
    • You *may* need to write two queries instead of one
  – Don’t compute in the WHERE
    • MySQL doesn’t know how to optimize constant expressions

```sql
SELECT * FROM Headlines
WHERE Time > SUBDATE( NOW(), INTERVAL 7 DAY);
```
Insert Speed

• In 4.1 and beyond, use prepared statements

• In older versions
  – Single inserts are the slowest
  – Multi-rows inserts are faster
  – Bulk-loading (LOAD DATA or mysqlimport) are very, very, very fast

• Using InnoDB, use transactions wisely
  – Many inserts in AUTOCOMMIT mode are very, very slow
Query Cache

- Part of MySQL 4.0
- Can seriously boost performance
- Might save legacy apps you can’t change
- Use query cache selectively if you have lots of writes
  - SELECT SQL_CACHE ...
- Use mytop to watch query cache stats
  - Version 1.3 and 1.4 will have more stats
MySQL Server Tuning

• Watching performance
• Benchmarking
• Tunable Parameters
  – Most bang, least effort
  – Incremental gains
• Methodology
  – Iterative testing
  – Long-term monitoring
MySQL Server Tuning
Watching Performance

Copyright 2003, Jeremy Zawodny
MySQL Server Tuning

• Key Performance Numbers
  – Queries per second
    • Min, Max, Short-term, Long-Term
  – Bytes per second
    • Inbound vs. Outbound
  – New connections per second
  – Idle vs. Active clients
  – Key cache efficiency
  – Query cache efficiency
MySQL Server Tuning

• How MySQL uses memory
  – Main Global Caches and Buffers
    • Query cache
    • Key buffer
    • Table cache
    • InnoDB buffer pool
    • InnoDB log buffer
  – Main Thread-specific Caches and Buffers
    • Record buffer
    • Sort buffer
    • Join buffer
MySQL Server Tuning

- **SHOW STATUS**
  - Created_tmp_disk_tables
    - If large, increase temp table size
  - Handler_*
    - Determine key buffer effectiveness
  - Com_*
    - Find the commands that are most often run
  - Questions and Uptime
    - Compute queries/second
  - Select_*
    - How many types of each SELECT are executed
  - Qcache_*
    - Query cache performance
On-the-Fly Tuning

• Use MySQL’s **SET** syntax to change parameters on the fly (new in 4.0)
  – max_connections
  – wait_timeout
  – thread_cache
  – key_buffer_size
  – table_cache

• Don’t change too much at once
• Persistent connections aren’t always fast!
• Changes may take time to notice
MySQL Server Tuning

- **SHOW STATUS**
  - `Table_locks_*`
    - How many times are queries waiting for locks?
    - Concurrency problems show up here
  - `Bytes_*`
    - How much data are you pumping out
    - Compare with inbound traffic
  - `Qcache_*`
    - Query cache performance
    - Memory usage
MySQL Server Tuning

• my.cnf file parameters
  – key_buffer
  – tmp_table_size
  – Table_cache
  – Max_connections
  – Max_user_connections
  – Long_query_time
  – Thread_concurrency
MySQL Server Tuning

• my.cnf file parameters
  – innodb_buffer_pool_size
  – innodb_log_file_size
  – innodb_file_io_threads
  – innodb_flush_log_at_trx_commit
  – innodb_log_buffer_size
  – innodb_flush_method
    • fdatasync
    • O_DSYNC
InnoDB Performance

• Transaction log flushing has three options
  – (1) Flush on commit
  – (0) Never flush
  – (2) Flush once per second
MySQL Server Tuning

• Fileysystem Issues
  – Spread data among disks
    • Put heavily used and lightly used databases together
    • RAID-5 or RAID-10 for data (w/batter-backed cache)
    • RAID-1 for logs
    • New CREATE TABLE makes this easier
  – Logs separate from data
    • Logs are mostly serialized writes
    • Tables are updated and used in mostly random fashion
  – If you have a lot of tables in a database
    • Use a filesystem designed to handle it
    • ResiserFS is a good choice
  – A journaling filesystem
    • Makes crash recovery faster
    • Better utilizes disk I/O (usually)
MySQL Server Tuning

• Upgrade once in a while
  – New versions are often faster
  – Better optimizations in query parser
  – New and enhanced caching

• Convert older tables to newer format
  – ISAM to MyISAM
  – BDB to InnoDB (or not)
  – `ALTER TABLE mytableName TYPE=InnoDB`

• Don’t flush the transaction logs on commit
Upgrade Testing

• It’s often a good idea to keep up-to-date
• Performance tweaks and optimizations are introduced during the maintenance process
• Be sure to test your critical queries carefully
• Always use a real load test or read the EXPLAIN output
• Without load, “slow” queries are often fast
Operating System Tuning

- Virtual Memory Use
  - FreeBSD - excellent
  - Linux - varies wildly
    - 2.4.9 good
    - >= 2.4.16 good
    - Others not good

- Per-process limits on:
  - Memory
  - File descriptors

- Network duplex settings

- Competing processes on the machine?
Operating System Tuning

• Key Metrics
  – Memory used/free/cache/buffer
    • Swapping is very bad
    • You might even disable swap
  – Paging and page faults
    • Make sure there’s no memory pressure
    • Server variables might be wrong if many page faults
  – Disk I/O
    • Make sure the I/O is where you expect
    • Disk I/O tuning (see your OS docs)
  – Processes running, sleeping, blocked/waiting
  – Actual CPU usage (might be too low)
Operating System Tuning

- Useful Unix Tools
  - top, ps, vmstat
  - iostat, sar
  - mrtg, rrdtool

- Windows Tools
  - Performance Monitor (perfmeter)
  - Task Manager
  - Others I don’t know (not a Windows guy)
Hardware Tuning

• CPU Issues
  – Speed
  – Single vs. Dual

• RAM Issues

• Disks
  – IDE vs. SCSI
  – RAID (hardware or software)
  – Battery-backed cache on controller is best
Hardware Tuning

• Network
  – The faster the better (watch latency)
  – Duplex settings

• I/O Channels
  – The more the merrier
  – Most PC motherboards suck
  – Server-class boards are better
  – High-end hardware (IBM, Sun) are best
  – You’ll be lucky to have this problem!
Network & Replication

• Put clients near servers
• Redundancy is very good
• Put slaves near master(s)
  – Unless that’s stupid
• Use load-balancing technology
  – High(er) availability MySQL
  – Easy scaling of traffic
• Pick the correct replication topology
• Backup slaves instead of the master
Network & Replication

• Replication is quite flexible
• Can build a topology to solve most problems
• Only a few nagging issues
  – Auto-increment fields
  – Automatic Fail-over
  – Need to build health checks
    • Performance/Latency
    • Slave stopped?
• Come to my replication talk to learn more!
Stupid Query Tricks

• Use SQL_CALC_ROWS and FOUND_ROWS() rather than double-queries:
  – SELECT ... LIMIT N, M
  – SELECT COUNT(*)

• Instead:
  – SELECT ... LIMIT N, M
  – SELECT FOUND_ROWS()

• Requires far less overhead on MySQL
Stupid Query Tricks

• Use a UNION to re-write a slow OR query

```
SELECT * FROM mytable
WHERE col1 = 'foo' OR col2 = 'bar'

(SELECT * FROM mytable
WHERE col1 = 'foo')
UNION
(SELECT * FROM mytable
WHERE col2 = 'bar')
```
Stupid Query Tricks

• Ordering, limiting, and ordering again

(SELECT * FROM mytable
WHERE col1 = 'foo'
ORDER BY col2 LIMIT 50)
ORDER BY col3
Final Advice

- Read
- Learn
- Test
- Ask
- Monitor
- Benchmark
For More Info...

- MySQL mailing lists
  - Visit lists.mysql.com

- Books
  - MySQL Manual
  - MySQL (Paul’s Book)
  - Managing & Using MySQL

- Web searching
Questions and Answers