Advanced MySQL Performance Optimization

Peter Zaitsev, Tobias Asplund
MySQL AB

MySQL Users Conference 2005
Santa Clara,CA April 18-21

Introductions

- Peter Zaitsev, MySQL Inc
  - Senior Performance Engineer
  - MySQL Performance Group Manager
  - MySQL Performance consulting and partner relationships
- Tobias Asplund
  - MySQL Training Class Instructor
  - MySQL Performance Tuning Consultant

Table of Contents

- A bit of Performance/Benchmarking theory
- Application Architecture issues
- Schema design and query optimization
- Sever Settings Optimizations
- Storage Engine Optimizations
- Replication
- Clustering
- Hardware and OS optimizations
- Real world application problems

Question Policy

- Interrupt us if something is unclear
- Keep long generic questions to the end
- Approach us during the conference
- Write us: peter@mysql.com, tobias@mysql.com

Audience Quick Poll

- Who are you ?
  - Developers
  - DBAs
  - System Administrators
  - Managers
- How long have you been using MySQL ?
- Did you ever have performance issues with MySQL ?
- What is your previous database background ?

Defining Performance

- Simple world but many meanings
- Main objective:
  - Users (direct or indirect) should be satisfied
- Most typical performance metrics
  - Throughput
  - latency/response time
  - Scalability
  - Combined metrics

Throughput

- Metric: Transactions per time (second/min/hour)
  - Only some transactions from the mix can be counted
- Example: TPC-C
- When to use
  - Interactive multi user applications
- Problems:
  - “starvation” - some users can be waiting too long
  - single user may rather need his request served fast

Response Time/Latency

- Metric: Time (milliseconds, seconds, minutes)
  - derived: average/min/max response time
  - derived 90 percentile response time
- Example: sql-bench, SetQuery
- When to use
  - Batch jobs
  - together with throughput in interactive applications
- Problems:
  - Counts wall clock time, does not take into account what else is happening

Scalability

- Metric: Ability to maintain performance with changing
  - load (incoming requests)
  - database size
  - concurrent connections
  - hardware
- Different performance metric
- “maintain performance” typically defined as response time
- When to use
  - Capacity planning

Queuing Theory

- Multi User applications
- Request waits in queue before being processed
- User response time = queueing delay + service time
  - Non high tech example – support call center.
- “Hockey Stick” - queuing delay grows rapidly when system getting close to saturation
- Need to improve queueing delay or service time to improve performance
- Improving service time reduces queuing delay

Service Time: Key to the hotspot

- Main Question – where does service time comes from ?
  - network, cpu, disk, locks...
- Direct Measurements
  - Sum up all query times from web pages
- Indirect measurements
  - CPU usage
  - Number of active queries
  - Disk IO latency
  - Network traffic
  - loadavg
  - etc

Benchmarks

- Great tool to:
  - Quantify application performance
  - Measure performance effect of the changes
  - Validate Scalability
  - Plan deployment
- But
  - Can be very misleading if done wrong

Planning proper Benchmarks

- Often Can't repeat application real world usage 100%
  - consider most important properties
  - can you record transaction log and replay it in parallel ?
- Representative data population
  - If everyone has the name “john” your result may be different
  - watch for time sensitive information
- Real database size
- Real input values
- Similar number of connections
- Similar think times
- Effect of caching
- Similar Server Settings,Hardware, OS, Network etc

Typical Benchmarking Errors

- Testing with 1GB size with 100G in production
- Using uniform distribution
  - "Harry Porter" ordered as frequent as Zulu dictionary
- Testing in single user scenario
- Going without think times
- Benchmarking on single host
  - While in real world application works across the ocean
- Running the same queries in the loop
  - query cache loves it
- Ignoring warm up
- Using default MySQL server settings

Estimating Growth needs

- Typically database grows as well as load
- Different tables grow differently
- Theoretical Blog site
  - Number of users – N new per day
    - may be non-linear
  - Number of posts - M per day for each user
- Query complexity growth may be different
- Different transactions may have different growth ratio
- Watch for user behavior changes
- Does database still fit in memory ?
  - 20% increase in size may slow things down 10 times.

Getting good results

- Make sure you measurements matches your goals
  - Are you looking at throughput ? scalability ?
- Make sure benchmark matches your problem
  - Do not use TPC-H for eCommerce benchmarks
- Gather all data you might need
  - CPU usage, disk IO, database performance counters etc
- Use right benchmarking methodology
  - Warm up ?
  - Test run length ?
  - Result filtering ?
- Compute margin of error

Business side of performance optimization

- Performance costs money, whatever road you take
- Investigate different possibilities
  - Better hardware could be cheaper than a major rewrite
- How much performance/scalability/reliability do you need ?
  - 99.999% could be a lot more expensive than 99.9%
- Take a look at whole picture
  - Is this the largest risk/bottleneck ?
- Identify which optimizations are critical for business
  - Optimization of “everything” is often waste of resources
  - What is the cost of suboptimal performance ?

Application Architecture

- Designing Scalable application Architecture
- Role of Caching
- Replication/Partition/Clustering
- Architectural notes for C/Perl/PHP/Java/.Net
- Application level performance analyses

Architecture – Key Decision

- Architecture is hard to change
- Scalable architecture often more complex to implement
  - How much performance do you need ?
  - To which level do you expect to scale ?
  - How long do you expect application to live ?
- Performance is not only requirement
  - extensibility
  - ease of maintainable
  - reliability, availability
  - integration with other applications
- Compromises may be needed

Architecture Design

- Try to localize database operations
  - "to change this we need to fix 15000 queries we need"
- Write code in "black boxes"
  - control side effects
  - be able to do local re-architecturing
- Think a bit ahead,
  - 1 hour of work today may be a week in a year
- Do not trust claims and your guts
  - run benchmarks early to check you're on the right way.
- Scale Out
  - 32 CPU box vs 20 2 CPU boxes
  - The "Google Way"

Take a look at big picture

- What functionality does this system use and provide ?
  - Can changing this affect performance ?
  - Minor use case behavior changes can give great boost
- Do not be limited to database server
  - storing large data in files (SAN, MogiloFS etc)
  - caching data in memcache
  - external FullText indexing
- Custom MySQL extensions
  - UDF, Storage Engines
- Non SQL data processing
  - process data in application instead of using complex query

Magic of Caching

- Most applications benefit from some form of caching
- For many caching is the only optimization needed
- Many forms of caching
  - HTTP Server side proxy cache
  - Pre-parsed template cache
  - Object cache in the application
  - Network distributed cache
  - Cache on file system
  - Query cache in MySQL
  - HEAP/MyISAM tables as cache
  - Database buffers cache

Proxy Cache

- External request-response cache
- Useful when data does not change
- Must have for static, semi-static web sites
- Can be just overhead for dynamic only
- Problems with cache invalidation
  - Protocol level control may not suite application
- Too high level
  - Can't cache even if difference minimal
- Security issues
  - storing sensitive data on the disk
  - disclosing data to wrong user

Parsed template cache

- Do not cache response itself, cache template for it
  - With all static data already parsed
  - request specific data added for response
- Is not the same as template language cache.
- Many different variants and tools available.
- Need to identify which data is “static” ? - which is not ?
- Example:
  - A static homepage, with the exception of rotating success stories

Object/Functional cache

- Cache results of functions or objects
  - for example user profile
- Will work for different templates and data presentations
  - Post in LiveJournal appears in a lot of “friend” pages
- Caching in application – simple
  - address space limit on 32bit systems
  - Limited to memory on single system
  - Multiple copies of same object

Cache on Network instead

- Instead of process cache on multiple nodes
  - no size limits
  - no double caching
  - use spare resources
  - network latency
    - better to be large objects
- Example tool:
  - memcached
- Can compliment local short term cache

Cache on the disk

- Can cache on file system
  - NFS, SAN, Local
- Well known “file” type interface
  - Can access cached objects from other applications
- Space is cheap
- Even larger latency
- Good for large objects
  - ie image generated from database data
- Good for objects costly to generate
  - Mnogosearch cache - example

MySQL Query Cache

- Caches query result
  - queries must be absolutely the same
- Caches in MySQL server process memory
- Fully transparent for application
  - activated just by server setting **query_cache_size=64M**
- No invalidation control
  - query invalidated when involved table is updated
- Does not work with prepared statements
- Works great for many read intensive web applications
  - As it is typically the only data cache used

HEAP/MyISAM Tables as cache

- HEAP Tables
  - Very fast, fully in memory
  - Limited by memory size
  - No BLOB support
- MyISAM
  - Fast disk based tables
- TEMPORARY
  - Result caching for single session
  - Caching “subquery” common for many queries
- Global
  - Caching data shared across session
  - Caching search results

Database/OS Buffers

- Data and indexes cached in Database or OS buffers
- Provided automatically, usually presents
  - MySQL server and OS Server settings.
- Fully transparent
- Very important to take into account
  - Access to data in memory up to 1000s times faster than on disk.
- Working set should fit in memory
  - Meaning load should be CPU bound
  - Often great way to ensure performance
  - Not always possible

Number of Connections

- Many Established connections take resources
- Frequent connection creation take resources
  - not as much as people tend to think
- Peak performance reached at small amount of running queries
  - CPU cache, disk thrashing, available resources per thread
  - Limit concurrency for complex queries
    - **SELECT GET_LOCK(“search”,10)**
- Use connection pool of limited size
- Limit number of connections can be established at the time
  - FastCGI, Server side proxy for web world

Replication

- Board sense – getting multiple copies of your data
- Very powerful tool, especially for read mostly applications
- MySQL Replication (Will discuss later)
- Manual replication
  - more control, tricky to code, can be synchronous
- Replication from other RDBMS
  - GoldenGate, used ie at Sabre
- Just copy MyISAM tables
  - Great for processed data which needs to be distributed
- Many copies: Good for HA, Waste of resources, expensive to update

Partitioning

- Local partitioning: MERGE Tables
  - Logs, each day in its own table.
- Remote partitioning – several hosts
  - example: by hash on user name
  - very application dependent
- Manual partitioning across many tables
  - Easy to grow to remote partitioning
  - Easy to manage (ie OPTIMIZE table)
  - Fight MyISAM table locks.
- May need copies of data partitioned different way
- No waste of resources. Efficient caching
- Can be mixed with replication for HA

Clustering

- Clustering – something automatic to get me HA, performance
- Manual clustering with MySQL Replication (more later)
- Clustering with shared/replicated disk/file system
  - Products from Veritas/Sun/Novell
  - Build your own using Heartbeat
  - Innodb, Read-only MyISAM
  - Does not save from data corruption
  - Active-Passive – waste of resources
  - Share Standby box to reduce overhead
  - Switch time can be significant
  - ACID guaranties – no transaction loss

Clustering2

- MySQL Cluster (Storage Engine)
  - Available in MySQL 4.1-max binaries
    - MySQL 5.0 will have a lot improved version
  - Shared nothing architecture
    - Replication + automatic hash partition
  - Many MySQL servers using many storage nodes
  - Synchronous replication, row level
  - Requires fast system network for good performance
  - Very much into providing uptime
    - including online software update
  - In memory only at this point. With disk backup.
  - Fixed cluster setup – can't add more nodes as you grow

Clustering3

- Third party solutions – EMIC Application Cluster
  - Nice convenient tools, easy to use
  - Commercial,
  - Patched MySQL version required
  - Synchronous replication, Statement level
  - Full data copy on each node
  - Limited scalability for writes, good for reads
  - Very transparent. Only need to reconnect
  - No multi statement transactions support
  - Some minor features are not supported
    - ie server variables
  - Quickly developing check with EMIC Networks

C/C++ considerations

- Native C interface is the fastest interface
  - "reference" interface which Java and .NET reimplement
  - Most tested. Used in main test suite, Perl DBI. PHP etc
  - very simple. May like some fancy wrapper around
  - Make sure to use threaded library version if using threads
  - Only one thread can use connection at the same time
    - use proper locking
    - connection pool shared by threads is good solution
  - Better to use same as server major version of client library
  - Prepared statements can be faster and safer
- ODBC – great for compatibility
  - performance overhead
  - harder to use

Perl

- Use latest DBD driver it supports prepared statements
- Using with HTTP server use mod_perl or FastCGI
- Do not forget to free result, statement resources
  - This is very frequently forgotten in scripting languages
- Beware of large result sets.
  - Set **mysql_use_result=0** for these
- Pure Perl DBD driver for MySQL exists
  - Platforms you can't make DBI/DBD compiled
  - Has lower performance
- Special presentation on Perl topic by Patrick

PHP

- Standard MySQL Interface
  - compatibility
- mysqli interface in PHP 5
  - Object mode
  - prepared statements like interface
    - safer
  - Support for prepared statements
  - Faster
- PEAR DB
  - Slower
  - Compatibility, support multiple databases
  - Object interface, prepared statements like interface
  - PHP5 Presentation

Java

- Centralize code that deals with data base
  - Change persistence strategies without rewrite
- Keep SQL out of your code
  - Makes changes/tuning possible without recompiling
- Use connection pooling, do not set pool size too large
- Do not use “autoReconnect=true”, catch exceptions
  - it can lead to hard to catch problems
- Use Connector/J's 'logSlowQueries'
  - It shows slow queries from client perspective
- Try to use Prepared Statements exclusively
  - Normally faster
  - More Secure (harder to do SQL Injection)

.NET

- Try to use prepared statements as much as possible.
- Close all connection you open
  - Simple but very typical problem
- Use ExecuteReader for all queries where you are just iterating over the rows
  - DataSets are slow and should only be used when you really need access to all of the rows on the client
- Handle Disconnect and other exceptions
  - No auto-reconnect support so less room for error

Application level performance profiling

- Application profiling is more accurate
  - Includes reading and processling result
- Gathering statistics on application level objects
- Can be combined with server data for deep analyses
  - MySQL 5.0 – **SHOW LOCAL STATUS**
- JDBC, PHP mysqli has great features build in
- Ideas:
  - How large portion of response time is taken by database ?
  - List all queries run to generate web page with their times and number of rows in debug mode.
  - Run EXPLAIN for slow queries

Shema design

- Optimal schema depends on queries you will run
- Data size and cardinality matters
- Storing data outside of database or in serialized for
  - XML, Images etc
- Main aspects of schema design:
  - Normalization
  - Data types
  - Indexing

Normalization

- Normalized in simple terms
  - all “objects” in their own tables, no redundancy
  - Simple to generate from ER diagram
  - Compact, single update to modify object property
  - Joins are expensive
  - Limited optimizer choices for selection, sorting
    - **select * from customer, orders where customer_id=order_id and order_date=”2004-01-01” and customer_name=”John Smith”**
  - Generally good for OLTP with simple queries

Non-Normalized

- Non-Normalized
  - Store all customer data with each order
  - Huge size overhead
  - Data updates are complex
    - To change customer name may need to update many rows.
  - Careful with data loss
    - deleted last order no data about customer any more
  - No join overhead, more optimizer choices
    - **select * from orders where order_date=”2004-01-01” and customer_name=”John Smith**”

Normalisation: Mixed

- Using Normalised for OLTP and non-normalised for DSS
- Materialized Views
  - No direct support in MySQL but can create MyISAM table
- Caching some static data in the table
  - both “city” and “city_id” columns
- Keep some data non-normalized and pay for updates
- Use value as key for simple objects
  - IP Address, State
- Reference by PRIMARY/UNIQUE KEY
  - MySQL can optimize these by pre-reading constant values
    - **select city_name from city,state where state_id=state.id and state.code=”CA”** converted to **select city_name from city where state_id=12**

Data Types

- Use appropriate data type – do not store number as string
  - "09" and "9" are same number but different strings
- Use appropriate length.
  - tinyint is perhaps enough for person age
- Use **NOT NULL** if do not plan to store NULLs
- Use appropriate char length. **VARCHAR(64)** for name
  - some buffers are fixed size in memory
  - sorting files, temporary tables are fixed length
- Check on automaticly converted schema
  - DECIMAL can be placed instead of INT etc

Indexing

- Index helps to speed up retrieval but expensive to maintain
- MySQL can only use prefix of index
  - key (a,b) .... where b=5 will not use index.
- Index should be selective to be helpful
  - index on gender is not a good idea
- Define UNIQUE indexes as UNIQUE
- Make sure to avoid dead indexes
  - never used by any query
- Order of columns in BTREE index matters
- Avoid duplicated - two indexes on the same column(s)
- Index, being prefix of other index is rarely good idea
  - remove index on (a) if you have index on (a,b)

Indexing

- Covering index – save data read, faster scans with long rows
  - **select name from person where name like “%et%”**
- Prefix index for data selective by first few chars
  - **key(name(8))**
- Short keys are better, Integer best
- Close key values are better than random
  - access locality is much better
  - **auto_increment** better than **uuid()**
- **OPTIMIZE TABLE** – compact and sort indexes
- **ANALYZE TABLE** - update statistics

Index Types

- BTREE
  - default key type for all but HEAP
  - helps “=” lookups as well as ranges, sorting
  - supported by all storage engines
- HASH
  - Fast, smaller footprint
  - only exists for HEAP storage engine
    - slow with many non-unique values
  - Only helpful for full “=” lookups (no prefix)
  - can be “emulated” by CRC32() in other storage engines
    - **select * from log where url=”http://www.mysql.com” and url_crc=crc32(“http://www.mysql.com”);**

More Index types

- RTREE
  - MyISAM only.
  - Works with GIS data
  - Speeds up multi dimensional lookups
- FULLTEXT
  - MyISAM only.
  - Speed up natural language search
  - Very slow to update for long texts
- More to come

Designing queries

- General notes
- Reading EXPLAIN output
- Understanding how optimizer works
- What exactly happens when query is executed
- Finding problematic queries
- Checking up against server performance data

General notes

- Know how your queries are executed
  - On the real data, not on the 10 rows pet table.
- Watch for query plan changes with upgrades, data change
- Do not assume a query that executes fast on other databases will do so on MySQL.
- Use proper types in text mode queries
  - int_col=123 and char_col='123'
- Use temporary table for caching
- Sometimes many queries works better than one
  - and easier to debug when 70K query joining 25 tables

Reading EXPLAIN

- Retrieved by **EXPLAIN** keyword before SELECT

```
mysql> explain select Country.Name, count(*) cnt from City,Country  where Country.Code=City.CountryCode and City.Population>(select max(population) from City
where CountryCode=Country.Code)/10  group by Country.Code order by cnt des;
+----+--------------------+---------+--------+---------------+---------+---------+-------------------------+------+---------------------------------+
| id | select_type        | table   | type   | possible_keys | key     | key_len | ref                     | rows | Extra                           |
+----+--------------------+---------+--------+---------------+---------+---------+-------------------------+------+---------------------------------+
|  1 | PRIMARY            | City    | ALL    | NULL          | NULL    |    NULL | NULL                    | 4079 | Using temporary; Using filesort |
|  1 | PRIMARY            | Country | eq_ref | PRIMARY       | PRIMARY |       9 | world.City.CountryCode  |    1 | Using where                     |
|  2 | DEPENDENT SUBQUERY | City    | ALL    | NULL          | NULL    |    NULL | NULL                    | 4079 | Using where                     |
+----+--------------------+---------+--------+---------------+---------+---------+-------------------------+------+---------------------------------+
3 rows in set (0.00 sec)
```

- UPDATE, DELETE need to be converted to SELECT
- For each SELECT MySQL executes from one table looking up in others
- id, select_type - to which query does this row corresponds
- table - which table is being accessed
  - Order of tables is significant.

Reading EXPLAIN

- type – how table is accessed (most frequent)
  - "ALL" - full table scan
  - "eq_ref" - "=" reference by primary or unique key (1 row)
  - "ref" - "=" by non-unique key (multiple rows)
  - "range" - reference by ">", "<" or compex ranges
- possible_keys - indexes MySQL could use for this table
  - check their list matches what you expect
- key – index MySQL sellected to use
  - only one index per table in MySQL 4.1 (fixed in 5.0)
  - Make sure it is correct one(s)
- key_length - Used key length in bytes
  - Check expected length is used for multiple column indexes

Reading EXPLAIN

- "ref" - The column or constant this key is matched against
- "rows" - How many rows will be looked up in this table
  - Multiply number or rows for tables in single select to estimate complexity
- "extra" - Extra Information
  - "Using Temporary" - temporary table will be used
  - "Using Filesort" - external sort is used
  - "Using where" - some where clause will be resolved with this table read

```
mysql> explain select * from t1,t2 where t1.i=t2.i order by t1.i+t2.i;
+----+-------------+-------+------+---------------+------+---------+------+-------+-----------------------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows  | Extra                       |
+----+-------------+-------+------+---------------+------+---------+------+-------+-----------------------------+
|  1 | SIMPLE      | t1    | ALL  | NULL          | NULL |    NULL | NULL | 36864 | Using temporary; Using filesort |
|  1 | SIMPLE      | t2    | ALL  | NULL          | NULL |    NULL | NULL | 36864 | Using where                 |
+----+-------------+-------+------+---------------+------+---------+------+-------+-----------------------------+
2 rows in set (0.00 sec)
```

MySQL Optimizer Basics

- Optimizer Goal – find the “best” plan for the query
- “Best” in optimizer cost model, not always fastest
- Optimizer uses statistics for its decision
  - number of rows in table, row size
  - cardinality – index selectivity
  - number of rows in constant range
  - different properties of storage engines
- Some assumptions are being made for missing statistics
- Optimizer has execution methods to use
  - full table scan, index scan, range, ref etc
- New versions: improved cost model, stats, methods

Simple Example

- **SELECT City.Population/Country.Population FROM City,Country WHERE CountryCode=Code;**
- MySQL need to select table order
  - Scanning **City** and checking **Country** for each
  - Scanning **Country** and checking all **City**es for it
- In each table orders different keys can be used
- Search set too large – not all possibilities tested
- Next Step: Optimize order by/group by if present
  - Should use index to perform sort ? Filesort ?
  - Should use temporary table or sort for group by ?

How is my query executed ?

- Scan table **City**
  - For each row, read row from **Country** by **PRIMARY** index
    - matching it to **City.CountryCode** column
  - Compute row values result values
  - Row is now buffered to be sent to client
    - as soon as network buffer is full it is sent to client

```
mysql> explain select City.Population/Country.Population from City,Country where CountryCode=Code;
+----+-------------+---------+--------+---------------+---------+---------+----------------------+------+-------+
| id | select_type | table   | type   | possible_keys | key     | key_len | ref                  | rows | Extra |
+----+-------------+---------+--------+---------------+---------+---------+----------------------+------+-------+
|  1 | SIMPLE      | City    | ALL    | CountryCode   | NULL    |    NULL | NULL                 | 4079 |       |
|  1 | SIMPLE      | Country | eq_ref | PRIMARY       | PRIMARY |       9 | world.City.CountryCode |    1 |       |
+----+-------------+---------+--------+---------------+---------+---------+----------------------+------+-------+
2 rows in set (0.00 sec)
```

Adjusting Optimizer behavior

- SELECT **STRAIGHT_JOIN** * from tbl1,tbl2 ...
  - Force table order as they're specified in the list
- USE INDEX/FORCE INDEX/IGNORE INDEX
  - SELECT * FROM Country **IGNORE INDEX**(PRIMARY)
  - advice using index/force using index/do not use index for table access.
- **SQL_BUFFER_RESULT**
  - Result will be buffered in temporary table before sending
    - handle slow clients, unlock MyISAM tables faster
- **SQL_BIG_RESULT/SQL_SMALL_RESULT**
  - Set if you're expecting large or small result set
  - Affects how group by is optimized, temporary table created

Finding problematic queries

- Run **EXPLAIN** on your queries
- Enable slow query log
  - **--log-slow-queries –long-query-time=2 –log-long-format**
  - **mysql_explain_log** - check explains for slow log
  - **mysqldumpslow** – aggregate slow query log data
- Use general query log on development boxes
  - query duplicates, too many queries – typical issue
- Run "**SHOW PROCESSLIST**"
  - catch frequent, slow and never ending queries
- What query actually does
  - **FLUSH STATUS**; <run query> **SHOW STATUS**
    - Idle server or MySQL 5.0 **SHOW LOCAL STATUS**

MySQL Server Optimization

- MySQL Server Architecture
- How MySQL Server uses memory
- MySQL Server General options
- Reading server run time status data

MySQL Server Architecture

- Single process, multiple threads
  - address space limits on 32bit
  - OS should have good thread support
  - No shared memory usage
- Each connection gets its own thread
  - 1000 connections will require 1000 threads
- Some helper threads can be used
  - signal thread, alarm thread, Innodb IO threads etc
- Thread “caching” to avoid thread creation for each connect
  - establishing connection is relatively cheap
- Client-Server communication TCP/IP, Unix Socket, Named Pipes

Memory usage in MySQL

- MySQL Server code (minor)
- Global buffers
  - key_buffer, query_cache, innodb_buffer_pool,table_cache
  - Allocated once and shared among threads
- Kernel Objects
  - sockets, kernel stacks, file descriptor table
  - File System Cache
- Thread Memory
  - thread stacks,
  - sort_buffer_size, tmp_table_size, read_buffer_size etc
- Do mix global buffers and per thread buffers.

General MySQL Server Tuning

- Tune queries,schema first
  - different queries need different tuning
- What hardware are you using ?
  - CPUs, Number of disks, memory size
- How much resources do you want MySQL to use ?
- How many connections are you expecting
  - thread buffers should not run you out of memory
- Which Storage Engine(s) are you using ?
- Load scenario
  - read/write mix, query complexity
- Special Requirements
  - Replication ? Audit ? Point in time recovery ?

General Settings

- **--character-set**
  - use simple character set (ie latin1) if single language
- **--join_buffer_size**
  - buffer used for executing joins with no keys. Avoid these
- **--binlog_cache_size**
  - when --**log-bin** enabled. Should fit most transactions
- **--memlock**
  - lock MySQL in memory to avoid swapping
- **--max_allowed_packet**
  - should be large enough to fit lagest query
- **--max_connections**
  - number of connections server will allow. May run out of memory if too high

More General Settings

- **--sort-buffer-size**
  - Memory to allocate for sort. Will use disk based sort for larger data sets
- **--sync_binlog**
  - Flush binary log to disk. Reduce corruption chances
- **--table_cache**
  - Number of tables MySQL can keep open at the same time. Reopening table is expensive
- **--thread_cache_size**
  - Keep up to this amount of thread “cached” after disconnect
- **--tmp_table_size**
  - Max size of memory hash table. Will use disk table for larger sets

Query Cache Settings

- **--query_cache_size**
  - Amount of memory to use for query cache
- **--query_cache_type**
  - Should query cache be disabled/enabled or on demand ?
- **--query_cache_limit**
  - Maximum result set size to cache. Avoid erasing all query cache by result of large query
- **--query_cache_wlock_invalidate**
  - Should query cache be invalidated on LOCK TABLES ... WRITE

MySQL Status data

- "**SHOW STATUS**"
  - MySQL 5.0 has **SHOW [LOCAL|GLOBAL] STATUS**,
  - "mysqladmin extended" command
  - shows status counters since last flush (or startup)
  - **FLUSH STATUS** to reset most of them
  - On 32bit systems counters may wrap around
  - Can be affected by rate bulk jobs (ie nightly backup)
- "mysqladmin extended -i10 -r"
  - Shows difference between counters over 10 sec interval
  - Shows what is happening now
  - Can show weird data:
    - | Threads_running | **-5** |

MySQL Status

- **Aborted_clients** - are you closing your connections ?
  - if no check network and **max_allowed_packet**
- **Aborted_connects –** should be zero
  - Network problems, wrong host,password, invalid database
- **Binlog_cache_disk_use** (1), **Binlog_cache_use** (2)
  - If ½ is large, increase **binlog_cache_size**
- **Bytes_received/Bytes_sent** - Traffic to/from server
  - Can network handling it ? Is it expected ?
- **Com_*** - Different commands server is executing
  - **Com_select –** number of selects, excluding served from query cache
  - Shows load information on query basics
  - Are all of them expected ? ie **Com_rollback**

MySQL Status

- **Connections –** number of new connections established
  - way to high number may ask for connection pooling.
- **Created_tmp_tables** - internal temporary tables created for some queries executions.
  - sometimes can be avoided with proper indexes
- **Created_tmp_disk_tables –** table taking more than **tmp_table_size** will be converted to MyISAM disk table
  - if BLOB/TEXT is sellected disk based table is used from start
  - look at increasing **tmp_table_size** if value is large
- **Created_tmp_files –** temporary files used for sort and other needs.

MySQL Status "Handlers"

- **Handler_*** - Storage engine level operations
  - Show which load do your queries generate
  - **Handler_read_key -** retrieve first value by key (ie a>5)
  - **Handler_read_next –** retrieve next matching row for clause
    - large if index scans, large ranges are used.
  - **Handler_read_next –** reverse index scan, rare
  - **Handler_read_rnd –** retrieve row by position
  - **Handler_read_rnd_next –** "physically" next row
    - Corresponds to full table scans.
    - **Handler_read_rnd_next/Handler_read_rnd –** approximate average size of full table scan
  - **Handler_update –** update existing row
  - **Handler_write** - insert new row

Status: Key Buffer

- **Key_blocks_not_flushed** - dirty key blocks in keycache
  - need to be flushed on shutdown, will be lost on crash
- **Key_blocks_used –** maximum number of key blocks used
  - decrease **key_buffer_size** if it is much lower than it after warm up
- **Key_blocks_unused –** number of free keyblocks now
- **Key_read_requests**, **Key_reads –** logical and physical key block reads
  - **Key_reads/Key_read_requests –** miss ratio
  - watch for **Key_reads/**sec - match against your io system
- **Key_write_requests**,**Key_writes –** logical and physical key block writes
  - miss ratio is typically much larger, some ways to improve.

Status

- **Max_used_connections** – maximum number of connections used
  - check if it matched **max_connections**
    - too low value or sign of overload.
- **Open_files** - number of files opened, watch for the limits
  - Storage engines (ie Innodb may have more)
- **Open_tables** - number of currently open tables
  - single table opened two times is counted as two
  - check it against **table_cache**, it should be large enough
- **Opened_tables** – number of times table was opened (**table_cache** miss)
  - check how many opens per second are happening, increase **table_cache** if many

Query Cache Status

- **Qcache_free_blocks,Qcache_free_memory -** number of free blocks and total free memory in Query Cache
  - many small blocks could be due to fragmentation
    - **FLUSH QUERY CACHE** to defragment. Can add to **cron**
    - increase **query_cache_min_res_unit**
- **Qcache_hits –** times result was served from query cache
- **Qcache_inserts** – times query was stored in query cache
  - This is overhead. **hits/inserts** should be large
- **Qcache_lowmem_prunes –** times older queries were removed due to low memory
  - increasing query cache makes sense in such case

Query Cache Status II

- **Qcache_not_cached -** number of queries which was not cached
  - using rand(), now(), temporary tables etc
  - **SQL_NO_CACHE,** no hint in demand mode
  - Comment before "S" in "**SELECT ...**"
- **Qcache_queries_in_cache** – number of queries stored in the cache
- **Qcache_total_blocks –** Total number of blocks in cache
  - check against **query_cache_size** to see average size of block

Server Status

- **Questions –** number of questions server got
  - all of them including malformed queries
  - good rough load indicator for stable load mix
- **Select_full_join –** number of joins without indexes
  - should be zero, these are real performance killer
- **Select_full_range_join** - number joins with range lookup on referenced table
  - potentially slow. Good optimization candidates
- **Select_range –** number of joins with range lookup on first table
  - typically fine

Server Status

- **Select_range_check –** joins when key selection is to be performed for each row
  - large overhead, check query plan
- **Select_scan –** joins with full table scan on first table
  - check if it can be indexed
- **Slow_launch_threads –** threads took more than **slow_launch_time** to create
  - connection delay
- **Slow_queries –** queries considered to be slow
  - logged in **slow_query_log** if it is enabled
  - taking more than **long_query_time** seconds to run
  - doing full table scan, if **log_queries_not_using_indexes** is specified
  - check query plans

Server Status: Sorting

- **Sort_merge_passes** - number of passes made during file merge sort.
  - consider increasing **sort_buffer_size**
  - check if file sort needs to be done at all
    - **SELECT * FROM people ORDER BY name DESC LIMIT 1;**
- **Sort_range –** sorting of the range
- **Sort_scan –** sorting by scanning, full table scan
- **Sort_rows –** number of rows sorted
  - a clue how complex sorts are happening

Server Status Table locks, threads

- **Table_locks_immediate** - table locks with no wait
  - Table locks are taken even for **Innodb** tables, waits rare
- **Table_locks_waited –** table lock requests which required a wait
  - no information how long waits were taking
  - large values could indicate serious bottlenecks
    - Innodb tables, partitioning, query optimization, concurrent insert, lock settings tuning to fix
- **Threads_cached** - number of threads in “thread_cache”
- **Threads_connected** - number of current connections
- **Threads_created** – theads created (thread_cache misses)
  - should be low.
- **Threads_running** – currently executing queries

Storage Engines

- MyISAM specific Optimizations
- Innodb specific Optimizations
- Heap Specific Optimizations
- Power of multiple Storage Engines
- Designing your own storage engine

MyISAM

- MyISAM Properties
  - no transactions, will be corrupted on power down
  - small disk and memory footprint
  - packed indexes, works without indexes, FULLTEXT,RTEE
  - table locks, concurrent inserts
  - read-only packed version
  - only index is cached by MySQL, data by OS
- Typical MyISAM usages:
  - Logging applications
  - Read only/read mostly applications
  - Full table scan reporting
  - Bulk data loads, data crunching
  - Read/write with no transactions low concurrency

MyISAM optimization hints

- Declare columns **NOT NULL**, save some space
- Run OPTIMIZE TABLE to defragment, reclaim free space, make concurrent insert to work.
  - needed only after significant data changes
- set **bulk_insert_buffer_size** if doing massive inserts, use multiple value inserts.
- Deleting/updating/adding a lot of data disable indexes
  - **ALTER TABLE t DISABLE KEYS**
- set **myisam_max_[extra]_sort_file_size** large so **REPAIR TABLE** is done by sort, much faster
- use **--myisam_recover** do not ever run with corrupted data
- use merge tables for large historical data. Index tree should fit in cache

MyISAM Table Locks

- Avoid “holes” in tables to use concurrent inserts
- Try **INSERT DELAYED,** note such data can be lost
- Chop long blocking queries,
  - **DELETE FROM tbl WHERE status=”deleted” LIMIT 100;**
- Try optimizing blocking queries
- Try **low_priority_updates=1** – waiting updates will not block selects, but may starve forever
- Vertically partition – separate columns you typically update
- Horizontally partition - users -> users01.... users09
  - also good help for **ALTER TABLE**, **OPTIMIZE TABLE**
- If nothing helps try Innodb tables.

MyISAM Key Cache

- Size set by **key_buffer_size** variable
  - For MyISAM only server 25-33% of memory is typical
- Can have several Key caches (ie for hot data)
  - **SET GLOBAL test.key_buffer_size=512*1024;**
  - **CACHE INDEX t1.i1, t2.i1, t3 IN test;**
- Preload index in cache for further quick access
  - preloading is sequential read, so very fast
  - **LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;**
- Midpoint insertion strategy
  - helps from large index scans clearing the cache
  - **SET GLOBAL test.key_cache_division_limit=20;**

Innodb Storage Engine

- Innodb Tables
  - transactional, ACID, foreign keys, data checksums
  - row level locks with versioning, consistent reads
  - Support for different isolation modes
  - much larger memory,disk footprint
  - no key compression
  - data and indexes cached in memory, in memory hash
  - clustered by primary key (implicit if not defined)
- Good for
  - Transactional applications
  - heavy concurrency applications
  - minimizing downtime on server crash
  - faster accesses by primary keys, better in memory performance

Innodb performance hints

- Use short,integer primary key
  - Add auto_increment column and change current **PRIMARY KEY** to **UNIQUE**
- Load/Insert data in primary key order
  - better externally sort it, if it is not in order
- Do large loads in chunks
  - rollback of failed **LOAD DATA INFILE** can take days
- Use **DROP TABLE/CREATE TABLE** instead of **TRUNCATE TABLE** (before 5.0)
- Use **SET UNIQUE_CHECKS=0**, **SET FOREIGN_KEY_CHECKS=0** for data load
- Try prefix keys - especially efficient as there is no key compression

Innodb server settings

- **innodb_buffer_pool_size** - buffer pool (cache) size
  - 60-80% of memory on Innodb only system
  - especially important for write intensive workload
- **innodb_log_file_size** - size of each log file.
  - set up to 50% of **innodb_buffer_pool_size**
  - check how frequently log file changes (mtime)
  - large values increase crash recovery time
    - test how long you can afford
- **innodb_log_files_in_group** – number of log files.
  - leave default
- **innodb_additional_mem_pool_size –** dictionary cache
  - Set 8-16M increase if **SHOW INNODB STATUS** reports spills

Innodb server settings II

- **innodb_autoextend_increment** - chunks in which autoextend innodb data files grow.
  - larger values, less FS fragmentation, smaller overhead
- **innodb_file_per_table –** create each table in its own file
  - can be used to put tables to specific devices
- **innodb_flush_log_at_trx_commit**
  - 1 (slow) will flush (fsync) log at each commit. Truly ACID
  - 2 will only flush log buffer to OS cache on commit
    - transaction is not lost if only MySQL server crashes
  - 0 (fast) will flush (fsync) log every second or so
    - may lose few last comited transactions
- **innodb_log_buffer_size –** size of log buffer
  - values 1-8MB flushed once per second anyway

Innodb Server Settings III

- **innodb_flush_method –** how Innodb will perform sync IO
  - default – use fsync()
  - **O_SYNC** open file in sync mode. Usually slow
  - **O_DIRECT** - use Direct IO on Linux.
    - Can offer significant speedup, especially on RAID
    - avoid extra data copying and “double buffering”
  - Some OS have different ways to reach it
    - ie **forcedirectio** mount option on Solaris
- **innodb_thread_concurrency** maximum number of threads in Innodb kernel.
  - Set at least (num_disks+num_cpus)*2
  - Try setting to 1000 to disable at all
  - Innodb does not like too many active queries still

SHOW INNODB STATUS

- **SHOW INNODB STATUS**
  - Great way to see what is going on inside Innodb
  - File IO
    - 66.23 reads/s, 17187 avg bytes/read, 0.00 writes/s, 0.00 fsyncs/s
  - Buffer Pool
    - Buffer pool size 24576, Free buffers 0, Database pages 23467, Modified db pages 0
  - Log activity
    - 5530215 log i/o's done, 0.00 log i/o's/second
  - Row activity
    - 0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 242.44 reads/s
  - Locks information, deadlocks, transaction status, pending operations, a lot more
- In MySQL 5.0, some variables exported to **SHOW STATUS**

HEAP Storage Engine

- HEAP storage engine properties
  - In memory, content is loss on power failure
  - HASH and BTREE indexes
  - Table locks
  - Fixed length rows
    - varchar(200) will take a lot even with empty string stored.
  - Very fast lookups
  - **max_heap_table_size** limits size
- Usage:
  - Cache tables
  - Temporary tables
  - Buffer tables (insert/update buffering)

HEAP Optimization hints

- Beware of table locks
- Fixed size rows – you may need much more memory for your data
- Do not run out of memory
  - HEAP table in swap is slower than MyISAM
- Use BTREE indexes for data with a lot of duplicates
  - deletes from HASH index with many dupes is very slow
- Use proper index types
  - HASH does not handle ranges or prefix matches.
- HEAP tables do not provide much optimizer stats
  - optimizer may chose wrong plan

Power of multiple Storage Engines

- You can mix them
  - On the same server
  - even in the single query
- Store constant data in MyISAM, dynamic critical data in Innodb and use Heap for temporary tables.
- ALTER TABLE tbl ENGINE=<engine>
  - Conversion back and forth is simple, easy to try
- Downsides
  - Mixed database configuration is more complicated
    - backup, maintenance, tuning
  - Potential of bugs while using multiple storage engines.
    - especially optimizer may have hard time.

Add your own storage engine

- You can easily add your own storage engine to MySQL
  - to solve your application specific needs
- Examples:
  - “Archive” storage engine to deal with huge log files
    - used by Yahoo
  - Special distributed storage engines
  - Storage engines for fuzzy matches
  - Storage engine for network lookup (Friendster)
  - Storage engine to read apache log files
- MySQL development and support can help with design and implementation.

MySQL Replication

- MySQL Replication Architecture
- Setting up MySQL Replication
- Replication concepts for your application
- Bidirectional, Circular replication issues
- Fallback/Recovery in MySQL Replication

MySQL Replication Architecture

- Replication done by binary log (**--log-bin**)
  - Master writes this log file
  - Slave fetches it from master and executes
- Binary log contains statements + extra information
  - Time when statement was executed if it uses **now()**
  - Can easily run out of sync without noticing it
  - Some funtionality does not work with replication – **uuid()**
- Replication is asyncronous. Slave has a bit old data.
- Slave has 2 threads
  - “IO Thread” - fetch statemenets from master, store locally
  - “SQL Thread” - get from local storage and execute
    - So if master goes down the gap between master and slave is small

Setting up MySQL Replication

- Make sure different **server_id** is set on Master and Slave
- Enable **--log-bin** on the Master.
- Create user on Master to use for replication
  - **GRANT REPLICATION SLAVE ON *.* TO 'repl'@'%.mydomain.com' IDENTIFIED BY 'slavepass';**
- Get master data snapshot to the slave, and binary log position
  - they must match exactly for replication to work properly
- **CHANGE MASTER TO MASTER_HOST='host', MASTER_USER='repl', MASTER_PASSWORD='slavepass', MASTER_LOG_FILE='recorded_log_file_name', MASTER_LOG_POS=recorded_log_position;**
- Run "**SLAVE START**"
- Run "**SHOW SLAVE STATUS**" on the slave to ensure it worked

Getting master data to the slave

- Many options
  - Shut down MySQL Server and copy data – downtime.
  - Shut down one of the slaves and clone it - need to have one
  - Use last consistent backup (how did you get this backup?)
    - Need to have all binlogs available since when
  - Use mysqldump –master-data
    - will make server read-only while it dumps data
  - Innodb: Use Innodb Hot Backup (commercial tool)
  - User LVM or other volume manager with snapshot
    - run **FLUSH TABLES WITH READ LOCK**
    - run **SHOW MASTER STATUS**, record position
    - create snapshot
    - run **UNLOCK TABLES**
    - copy snapshot to the slave

Replication Options

- **--log-slave-updates –** log updates from slave thread
  - useful for chain replication, using slave for backup
- **--read-only** - do not allow updates to the slave server
  - useful as protection from application errors.
- **--replicate-do-table**,**--replicate-wild-do-table** – specify tables, databases to replicate
  - avoid using –**replicate-do-db**
- **--slave_compressed_protocol=1** Use compressed protocol
  - useful for replication over slow networks
- **--slave-skip-errors** - continue replication with such errors
- **--sync_binlog=1** - Sync binlog on each commit
  - if you want to continue after master restart from crash

Replication concepts

- Master -> Slave
  - Most simple one, gives some HA and performance
- Master <-> Master
  - write to both nodes, simple fall back, update conflict problem
- Master -> Slave1...SlaveN
  - Great for mostly read applications, easy slave recovery
  - More complex fall back, resource waste – many copies
  - Write load does not scale well.
- Master1 -> Slave1, Master1->Slave2 ...
  - Replication together with data partition.
  - Can be used in bi-directional mode too
  - Limited resource waste, good write load scalability
  - Can have several slaves in each case

Bi-Directional Replication

- Master1 <-> Master2
  - Writing to both nodes – update conflicts, no detection
    - Due to asynchronous replication
    - auto increment values collide
      - MySQL 5.0 **--auto-increment-offset=N**
    - updates can be lost
  - Make sure no conflicting updates if both Masters writable
    - Check if queries can be executed in any order
      - **UPDATE TBL SET val=val+1 WHERE id=5**
    - Partition by tables/ objects
      - Master1 works with even IDs Master2 with odd
  - Writing to one of them at the time
    - Other protected by **--read-only**
    - Easy to fall back – no need to reconfigure

Chain,Circular Replication

- Chain Replication
  - Slave1->Slave2->Slave3
  - Can be used as “tree” replication if there are too many slaves
  - HA – if middle node fails, all below it stop getting updates
  - Complex rule to find proper position for each on recovery
- Circular Replication
  - Slave1->Slave2->Slave3->Slave1
  - Same problems as in Bi-Directional replication
  - Same HA issues as Chain Replication

Making sure Master, Slave in Sync

- Internal inconsistence detection is weak
  - you will get errors for duplicate keys, corrupted tables
- MyISAM - create table with **CHECKSUM=1**
  - some write performance penalty
  - use **CHECKSUM TABLE tbl** to retrieve checkum.
- Checksum can be computed on the fly for any table
  - full table scan is needed, could be long lock
- Master and Slave must be in sync when comparing checksum:
  - **LOCK TABLE tbl WRITE** - on master;
  - **SELECT MASTER_POS_WAIT(master_position)** – on the slave;
  - Compare checksums.

Fall back, Master goes down ?

- Some transactions can be lost as replication is async
- Having shared data active-passive clustering is option if this is unacceptable.
- If using many slaves could keep one underloaded so it is most up to date
- Have **–log-slave-updates** enabled.
- Select most up to date server from the slaves. Compare **SHOW SLAVE STATUS**
- Re-compute new position for each - tricky
- Use CHANGE MASTER STATUS to change it
  - MySQL will take care of old relay logs

Recovery, Slave Goes down

- You can't be sure data restarting replication will be consistent even if using only Innodb tables.
  - master.info, relay logs are buffered.
- Let slave run a bit and check if it is consistent with master
  - May seriously slow down/block master.
- Clone the slave from scratch
- Ignore the problem and hope to be lucky
  - most commonly used approach :)

Replication aware application

- Taking into account asynchronous nature of replication
  - Data on slaves is not guaranteed up to date. Use Master reads if last update should be visible
- Difference between masters and slaves
  - One may prefer do reads from slaves and writes and live reads from Master
- Handling update protocol
  - If Bi-Directional replication is used, make sure conflicting updates are not issued. Ie do balancing by table ID
- Load Balancing
  - Balance load across the slaves or partitions
- Fall back
  - Master or slave may day, need proper handling.

Hardware,OS, Deployment

- Hardware selection for MySQL
- Hardware Configuration
- OS Selection
- OS Configuration
- Physical Deployment

Hardware Selection

- CPU: Consider 64bit CPUs
  - EM64T/Opteron are best price/performance at this point
- CPU Cache – Larger, better
  - CPU Cache benefit depends on workload
    - 1MB->2MB seen to give from 0 to 30% extra
    - Large number of threads benefit from increased size
- Memory Bandwidth – Frequent bottleneck for CPU bound workloads
  - Fast memory, dual channel memory, dedicated bus in SMP
- Number of CPUs: Single query uses single CPU
  - multiple queries scale well for multiple CPUs
    - consider logs Storage engine is setting for you
- HyperThreading – gives improvement in most cases

Hardware Selection II

- System Bus - can be overloaded on high load
  - different buses of IO, Network may make sense
- Video Card, Mouse, Keyboard
  - MySQL Server does not care :)
- Network card
  - Watch for latency, 1Gb Ethernet are good
  - CPU offloading (Checksum generation etc)
    - check for driver support
- Extension possibilities
  - Can you add more memory ? More disks ?

Disk IO Subsystem

- Need RAID to ensure data security
  - Slaves could go with RAID0 for improved performance
- RAID10 – best choice for many devices
  - RAID1 if you have only two disks
- RAID5 – very slow for random writes, slow rebuild
  - cheaper drives in RAID10 usually work better
- Battery backed up write cache
  - truly ACID transactions with small performance hit
- Multiple channels good with many devices
- Software RAID1/RAID10 typically good as well
  - random IO does not eat much of CPU time
- Use large RAID chunk (256K-1MB)

Disk IO Subsystem

- Compute your IO needs – drive can do (150-250 IO/sec)
- Test your RAID if it gives you performance it should
  - SysBench http://sourceforge.net/projects/sysbench
- Test if Hardware/OS really syncs data to disk
  - Or bad corruption may happen, especially with Innodb
- SAN – easy to manage but slower than direct disks
- NAS, NFS – Test very carefully
  - works for logs, binary logs, read only MyISAM
  - a lot of reported problems with Innodb
- Place Innodb logs on dedicated RAID1 if a lot of devices
  - otherwise sharing works well
  - OS could use the same drive

Hardware configuration

- Mainly make sure it works as it should
  - sometimes bad drivers are guilty
- Does your IO system delivers proper throughput
  - check both random and sequential read/writes
  - Cache set to proper mode ?
    - good to benchmark, settings, ie read-ahead
- Is your network is set in proper mode (ie 1GB/full duplex)
  - CPU offloading works ? Any errors ?
  - What is about interrupt rate ?
    - Some drivers seems to have problem with buffering, taking interrupt for each packet
- Test memory with memtest86 if unsure
  - broken memory frequent source of MySQL “bugs”

OS Selection

- MySQL Supports wide range of platforms
  - Linux, Windows,Solaris are most frequently used
    - all three work well
  - Better to use OS MySQL delivers packages for
  - RedHat, Fedora, SuSE, Debian, Gentoo – most frequent
    - Any decent distribution works
    - Get MySQL server from http://www.mysql.com
  - Ensure vendor can help you – we can't fix some OS bugs
- Watch for good threads support
  - Kernel level threads library for SMP support
  - Older FreeBSD, NetBSD had some issues
- Make sure your memory is addressable by OS
- Make sure all your hardware is well supported by OS

OS Configuration

- Allow large process sizes
  - MySQL Server is single process
- Allow decent number of open files, especially for MyISAM
- If possible lock MySQL in memory (ie **–memlock**)
- Make sure VM is tuned well, to avoid swapping
  - And Size MySQL buffers well
- Tune read-ahead. Too large read-ahead limits random IO performance
- Set proper IO scheduling configuration (**elevator=deadline** for Linux 2.6)
- Use large pages for MySQL process if OS allows ie
  - **--large-pages** option in 5.0 for Linux

OS Configuration

- Use Direct IO if using Innodb for Data
  - Logs and MyISAM are better with buffered
  - **O_DIRECT** in Linux "forcedirectio" in Solaris
- Set number of active commands for SCSI device
  - default is often too low
- Make sure scheduler is not switching threads too often
  - with large number of CPUs, CPU binding could help
- Use large file system block/extent size
  - tables are typically large
  - use "notail" for reiserfs

Deployment Guidelines

- Automate things, especially dealing with many systems
- Have load statistic gathering and monitoring
- Use different Database and Web (application) Server
  - different configuration, quality requirements, scaling
- Do not have MySQL servers on external network
  - Web servers with 2 network cards are good
- Have regular backup schedule
  - RAID does not solve all the problem
- Use binary log so you can do point in time recovery
- Have slow log enabled to catch slow queries.

MySQL Workloads

- MySQL in OLTP Workloads
- MySQL in DSS/Data warehouse Workloads
- Batch jobs
- Loading data
- Backup and recovery

OLTP Workloads

- Online Transaction Processing
  - Small Transactions, Queries touching few rows, random access
  - Data size may range from small to huge, not uniform access
- Make sure your schema is optimized for such queries
- If you can fit your working set in memory – great
- Watch for locks (table locks, row locks etc)
- For large databases – check random IO your disks can handle
- Configure MySQL for your number of connections
  - Large global buffers (key_buffer, innodb_buffer_pool)
  - Smaller per thread buffers   - sort_buffer, read_rnd_buffer

DSS/Data warehouse Loads

- Decision Support and Data Warehouse queries
  - Large database, few users
  - Start schema – many tables in join, or denormalized
  - Long running complex queries.
- MySQL does not have HASH/SORT MERGE Join support
  - may benefit by preloading dimension tables to HEAP table
- Great full table scan performance, especially MyISAM
  - denormalized schema often works better
- No physical order index scan
  - sort your indexes (**OPTIMIZE TABLE**) or preload them
- May need to help optimizer with **STRAIGHT_JOIN** if joining may tables

MySQL In Batch Jobs

- Long running data crunching, complex queries or many queries.
- Watch for locks (especially MyISAM) may chop task
  - **DELETE FROM TBL WHERE ts<"01-01-2005" LIMIT 100**
- Use temporary tables – result buffering, data selection
- Creating shadow tables for operation may make sense
  - ie small MyISAM table based on Innodb table
- Running batch jobs on dedicated Slave
- Periodic sleep() to avoid resource hog
- Do some data processing in application
  - beware **mysql_store_result()** with large data sets
    - **use mysql_use_result()**

Loading data in MySQL

- Creating table without indexes, loading data and creating indexes is very slow
  - MySQL recreates whole table in such case
- Do not add indexes one by one, add all of them by **ALTER TABLE**
  - if you're dropping/adding columns do it in the same command
- Parallel loading
  - **myisam_repair_threads**=N will build indexes in parallel
  - Innodb does not have matching option.
  - May load different tables at the same time
    - beware of fragmentation,random IO, increased working set

Loading data in MySQL

- Parallel Load
  - May load different tables at the same time
    - beware of fragmentation,random IO, increased working set
- MyISAM
  - loading data in empty table is much faster.
    - Workaround – use **ALTER TABLE** t **DISABLE KEYS** before loading data, **ALTER TABLE t ENABLE KEYS** after
  - Index rebuild by sort is very important
    - check it is the case in **SHOW PROCESSLIST**
    - **myisam_sort_file_size=100G, myisam_max_extra_sort_file_size=100G**
    - use large **myisam_sort_buffer_size**
    - Unique indexes are not build by sort (use large key_buffer_size)
  - Bulk_insert_buffer_size
    - Increase if doing bulk inserts in table with data

Loading data in Innodb tables

- Large **innodb_buffer_pool, innodb_log_file_size** for the time of the load
- Innodb does load row by row at this point
- Beware of crash during the load (rollback takes forever)
  - load data in chunks (ie by 10000 rows)
    - May load to MyISAM with no indexes and convert to Innodb.
- Load data in primary key order. Do external sort if needed
- May watch how load goes in **SHOW INNODB STATUS**
- If have unique keys and sure data is unique
  - **SET UNIQUE_CHECKS=0**
- If have foreign keys and sure they match
  - **SET FOREIGN_KEY_CHECKS=0**

Backup and Recovery in MySQL

- Backup is similar to slave snapshot creation
  - sometimes relaxed consistency may be required (ie for DB)
    - note you can't do point in time recovery from such backup
- Store your binary logs since at least last 2 backups
  - some people archive them forever.
- Test your backup actually restores valid data
- Test how long time restoration process takes
  - Textual backups can take very long time to restore
- Test how long time roll forward recovery takes
  - **mysqlbinlog logfile015.bin –start-position=123 | mysql**
  - It may take up to several hours for each live hour
    - roll forward recovery is done by single thread
  - set **innodb_flush_log_at_trx_commit=0** for recovery

Application problem examples

- Fulltext Search
- Random object sellection
- Logging
- Working with tree structures
- Listing navigation
- Storing large objects

Full Text Search

- Manually building FullText search ie (doc_id,word_crc)
  - used by PHPbb, database independent – very slow
- MySQL native FullText Search
  - Simple to use .. **MATCH (descr) AGAINST ('keyword')**
  - Search with relevance or in “Boolean Mode” (faster)
  - Only works for MyISAM tables
    - Can use shadow table when Innodb table is used
  - Indexed updated in live fashion (slow updates)
  - Really slow when index does not fit in memory
  - Slow with common words search
    - **MATCH (product) AGAINST (“video evita” IN BOOLEAN MODE) -> MATCH (product) AGAINST (“evita”) IN BOOLEAN MODE) AND product LIKE “%video%”;**

Full Text Searching

- MySQL native Full Text Search
  - Need multiple indexes if you want different searches
    - **MATCH (title)** ... **MATCH(title,descr)**
    - May use **MATCH(title,descr) ... and title like "%match%"**
  - Bulk updates in shadow search table for good performance
  - No native stem support – may use special field with stemmed text, same works for custom parsed text
  - Index stored in BTREE
    - fetching data requires random IO
    - OPTIMIZE table improves performance, sorting index

FullText Search: Caching

- Some searches are more frequent than others
  - cache these
  - to avoid cold start pre-fill caches on data update
  - separate MyISAM are good for caching – easy to drop
- Skip COUNT(*) computation (or **SQL_CALC_FOUND_ROWS**)
  - Very slow operation.
- Do single FullText search match, process results later
  - ie if you want to show how many matches in each subgroup
- Prefetch more results when you show on the first page
  - So you do not have to run the whole query again for second page
- Query cache works good with FT Search for small loads
  - make sure table with FT indexes rarely updated

FullText Search: Mnogosearch

- Full text search engine, initially for indexing files
  - Adapted to be able to index database
  - Stores full text index in separate tables or on file system
  - Multiple DB storage modes
  - Incremental indexing, indexing done on demand
  - Supports stop words, synonyms, morphology
  - Request caching (on file system)
  - Multiple document ranking modes
  - Boolean search
  - Large memory requirements
  - Homepage: http://www.mnogosearch.org
  - Used for Manual search at MySQL.com

FullText Search: Sphinx

- Designed specially for indexing databases
- Stem based morphology, stop words support
- Very small compressed indexes
- Index stored on file system in sorted form
  - can be fetched in single sequential read
- Very fast index speed, 5min vs 24 hours for Mnogosearch
- Modest (tunable) memory consumption
- Good relevance ranking (any,all)
- Fast retrieval from given offset, match counting
  - “displaying result 1000.1010 from 56787”
- Project Page: http://shodan.ru/projects/sphinx

FT: Performance comparison

- Database: 500MB, 3mil documents, 128M Key buffer, 512M memory
  - Sites: “url,title,description”
- “match all” mode
- MySQL native Full Text search tested just count(*)
  - count retrieval typically needed anyway
- “internet web design” - 134.000 docs matches
- Results in seconds

	FullText Index	FT Boolean	Mnogosearch	Sphinx
NonCached	392	12	3.5	0.23
Cached	272	11	1.06	0.15

Selecting random object

- **SELECT * FROM tbl ORDER BY RAND() LIMIT 1**
  - requires large scan and expensive sorting
- Add “rnd” column, index it, update periodically
  - **SELECT id FROM T tbl ORDER BY rnd LIMIT 1**
  - **UPDATE TBL SET rnd=RAND() WHERE id=<id>**
  - may use “used” column instead of rnd updating
    - **SELECT id FROM TBL WHERE USED=0 ORDER BY rnd LIMIT 1**
- Partition it into buckets
  - **SELECT * FROM TBL WHERE BUCKET=<rnd> ORDER BY RAND() LIMIT 1;**
    - if bucket is small sort is fast
- If sequential IDs with no holes – use direct lookup
  - **SELECT * FROM tbl WHERE id=<rnd 1...N>**

Logging

- Logs in database are cool – easy reporting using SQL
  - **SELECT AVG(rtime) FROM log WHERE request=”search”**
- MyISAM table with no indexes – fast logging and scans
  - “Archive” storage engine has smaller footprint
- Use “**INSERT DELAYED**” so live reporting possible
  - if “no holes” CONCURRENT insert should work as well
  - may write them to file and use separate “feeder”
- Limit indexes – these are most expensive to update
  - with index - keep tables small so index tree fits in memory
- Create multiple tables, easy, fast data purging:
  - **INSERT INTO log20050101 (...) VALUES (...)**
  - if error, **CREATE TABLE log20050101 LIKE base_table**
  - retry insert

Working with Tree Structures

- Typical tasks: Finding path to top, finding all objects in current subtree
- “Classical solution” - specially enumerate nodes so between can be used for lookup. (Joe Celko)
  - expensive - tree may need to be rebuilt on each change
- Use “group_parents” table (group_id,sub_group_id,level)
  - **SELECT GROUP_ID WHERE SUB_GROUP_ID=<N> ORDER BY LEVEL**
    - Gets you path to top
  - **SELECT SUB_GROUP_ID WHERE GROUP_ID=<N>**
    - Gets you all groups from this group subtree,
- May make sense to cache string Path in the group table
  - /Products/Electronics/VHS
    - **... LIKE “/Products/%”** will get you all subgroups

Listing navigation

- Common problem – directories, forums, blogs etc
  - "show everything from offset 2000 to 2010"
  - **SELECT * FROM tbl LIMIT ORDER BY add_time 2000,10** works but slow
    - 2000 rows has to be scanned and thrown away
- Precompute position
  - **SELECT * FROM tbl WHERE POS BETWEEN 2000 and 2010** is fast
    - hard to do live, may use delayed published
    - "new" entries can be shown out of order until position counted
- Cache - pull first 1000 entries and precompute positions
  - only few people will go further than that.
- Specific applications may have more solutions

Storing Large objects in MySQL

- Files work faster
- Why do it ?
  - Uniform access interface, transactions, replication consistent backup...
- Always full reads - can't get first 100 bytes
- MyISAM – row read done together with BLOB
  - may use separate table if BLOB is rarely accessed
  - Innodb will skip reading BLOB if it is not requested
- Watch for fragmentation, if deleting/updating
- Memory consumption – 3 times size the blob on server
- Use Binary Protocol - avoid escaping.

Resources

- MySQL Online Manual – great source for Information
  - http://dev.mysql.com/doc/mysql/en/index.html
- SysBench - Benchmark and Stress Test tool
  - http://sourceforge.net/projects/sysbench
- FullText Search systems
  - Mnogosearch: http://www.mnogosearch.org
  - Sphinx: http://www.shodan.ru/projects/sphinx
- MySQL Benchmarks mailing list
  - benchmarks@lists.mysql.com
- Write us your questions if you forgot to ask
  - peter@mysql.com tobias@mysql.com
  - Feel free to grab on the conference to discuss your problems