

# MySQL Performance Optimization

By Peter Zaitsev, Percona Ltd



**OPEN SOURCE DATABASE  
CONFERENCE 2006**

Learn • Share • Network • Benefit

# About Speaker

- Peter Zaitsev
  - Co-Founder Percona Ltd
    - Company specializing in MySQL and LAMP Performance Optimization
  - Spent 4.5 years with MySQL AB
    - High Performance Group Manager



# About Presentation

- Overview of MySQL Optimization
  - Server Settings
  - Queries and Schema
- Operating System and Hardware
- Optimum application architecture design
- Deployment and Operations
- Optimization Methodology



# Question Policy

- Ask your short questions at once
- Keep longer questions to the end
- Feel free to ask them during conference breaks etc



**OPEN SOURCE DATABASE  
CONFERENCE 2006**

Learn • Share • Network • Benefit

# Start with installation

- Binary releases from MySQL are best
  - This may change with new release policies
- OS Vendors binaries may be outdated
  - Or sometimes wrongly built
- Compiling on your own make sure you do it right
  - Compiler and library bugs are common causes for problems



# Do not use Default Configuration

- Default MySQL Configuration is Tiny
  - Using some 16MB of Memory
  - Designed to run as secondary task on the system
- Start with one of MySQL Sample configs provided
  - Better than default but still need tuning



# Know your application

- How many connections are you going to have ?
- Are you using MyISAM, Innodb ? Mix ?
- Are you going to run complex sorts or group by queries ?
- Can you afford to lose data on the crash ?
- What is your working set
  - Data you access most frequently



# Look at SHOW STATUS

- SHOW [GLOBAL] STATUS great way to see what is happening on low level
- Many Status counters show what needs to be adjusted
- FLUSH STATUS; <query> SHOW STATUS
  - Great way to profile query in MySQL 5.0



# Status Variables

- Com\_XXX
  - What is happening on Command level
  - Are you getting more inserts or selects ?
  - Does NOT tell about query complexities
- Connections
  - Number of connections. Connection creation is relatively expensive
  -



# Temporary Objects

- **Created\_tmp\_disk\_tables, Created\_tmp\_files, Created\_tmp\_tables**
  - Temporary objects can be expensive
    - Check if queries can be optimized first
  - **tmp\_table\_size** can be increased to reduce number of tables created on disk
    - In some cases, ie BLOB/TEXT fields this does not help



# Handler Statistics

- Handler\_XXX
  - Low level row operations
    - Lookup by index, full table scan, inserts, deletes
  - Good measure of load
    - Rows also can be different though
  - Show if indexes are adequate
  - Check how query was really executed
    - EXPLAIN plan and estimates can be incorrect



# Storage engine related

- Innodb\_XXX
  - Innodb statistics since MySQL 5.0
  - Mainly duplicates one from SHOW INNODB STATUS
- Key\_XXX
  - Key Buffer activity metrics for MyISAM tables
  - High amount of misses  
(**Key\_reads/Key\_Writes**) - consider increasing



# Status Variables (5)

- **Max\_used\_connections**
  - Highest number of connections used
  - If matches Max\_Connections you probably ran out of connections at some time.
- **Opened\_tables**
  - Table Cache efficiency. Do not allow to grow more than 1/sec or so
  - Table opens can be rather expensive



# Query Cache

- **Qcache\_XXX**
  - Query Cache Statistics
  - Query Cache Efficiency
    - **Qcache\_hits** - Served from query cache
    - **Qcache\_inserts** - Query Cache Writes
    - **Com\_Selects** - Selects which were executed
    - First is “benefit” last two are “overhead”
  - **Qcache\_lowmem\_prunes**
    - Low on Qcache memory or fragmentation



# Status Variables (7)

- **Questions**
  - Number of questions server has handled
    - Simple load metrics, especially for constant query mix.
- **Select\_full\_join**
  - Joins done without indexes. Should be 0 in most cases
- **Select\_scan**
  - Full table scan queries



# Status Variables (8)

- **Slow\_queries**
  - Queries logged in slow query log
    - Taking longer than `long_query_time` sec or (configurable) doing full table scans
- **Sort\_merge\_passes**
  - Disk based sorts
    - Consider increasing `sort_buffer_size`
      - and look at query optimization



# Status Variables (9)

- **Table\_locks**
  - MySQL Level table locks information
  - Applies to MyISAM, MEMORY etc
  - **Table\_locks\_immediate**
    - Table lock requests succeeded without a wait
  - **Table\_locks\_waited**
    - Requests which had to wait
      - No information about wait time available in status variables.



# Threads

- **Threads\_cached**
  - Number of threads kept in thread cache
- **Threads\_connected** - Current connections
- **Threads\_running**
  - Connections currently active. High number means overload
- **Threads\_created**
  - Threads physically created. Cache misses.



# General tuning tips

- Memory buffers are very important for performance
  - `innodb_buffer_pool_size`, `key_buffer_size`
- Balance appropriately between storage engines
  - Note InnoDB performance is more affected by buffer sizes than MyISAM.
- Do not waste memory if DB size is small



# Do not allocate too much memory

- Using less memory than available reduces performance
- Using more memory than you have can lead to even worse performance or crashes
- Never let your DB box to swap actively
  - Some swap space allocated is OK, watch IO
- Never let MySQL to run out of memory
  - Or address space on 32bit boxes.



# Use proper unit for variables

- **Table\_cache=512**
  - Not thread\_cache=512K
    - It is measured in entries rather than bytes
- **key\_buffer\_size=32M**
  - Size is given, not number of blocks



# Do not use overly large values

- Do not just scale values from sample config file
  - Config sample for 1GB has X variable =1M
  - I have 16GB so I set it to 10GB
- Use values you need. Larger values may decrease performance
  - `sort_buffer_size` of 512K will be faster than 16M for small sorts.



# Consider per session variables

- Optimize global variables for your common queries.
- If you have one query which needs to do large sort DO NOT set sort\_buffer to 16M
- Do
  - SET sort\_buffer\_size=16000000;
  - Run Query
  - SET sort\_buffer\_size=DEFAULT;



# Do not expect magic from settings tuning

- For most workloads changing 5-8 variables from defaults is enough
- Tuning rest affects performance few percents
- Changing other variables has similar effect
- Order of magnitude or better improvements normally come from Schema and Query tuning



# InnoDB Configuration

- MyISAM has decent performance with defaults, InnoDB will crawl.
- Speeding Up InnoDB in two minutes:
  - Increase `innodb_buffer_pool_size`
  - Increase `innodb_log_file_size`
  - Configure `innodb_flush_logs_at_trx_commit`
    - Set it to 2 if you can afford losing committed transactions (ie moving from MyISAM)



# Advanced Innodb configuration

- **innodb\_flush\_method=O\_DIRECT**
  - May speed up things bypassing OS buffering or may slow things down
- **innodb\_thread\_concurrency=N**
  - Number of threads to run inside Innodb kernel. Set to  $\text{NumCPUs} * 2$  or lower value for CPU bound queries. IO bound work better with higher value



# Advanced InnoDB Configuration

- **innodb\_log\_buffer\_size**
  - Increase if you have intensive log IO but not too much. Log is flushed once per sec.
- **innodb\_additional\_mem\_pool\_size**
  - Space for data dictionary. Modest benefit from increasing in most cases
- **innodb\_file\_per\_table**
  - Use its own tablespace for each table





# Queries and Schema



**OPEN SOURCE DATABASE  
CONFERENCE 2006**

Learn • Share • Network • Benefit

# Queries and Schema come together

- Design queries and schema together
  - Simple data modeling may be “right” but often slow
- Try to think in operations rather than queries
  - What result I need to provide ?
  - Rather than I need to join these two tables



# General schema considerations

- You want your lookups to be indexed
  - `SELECT id FROM user WHERE login='sam'`
    - Needs index on (sam)
- Sorts better done by index
  - `SELECT title FROM news ORDER BY created DESC limit 10`
    - Needs index on (created)
  - Full table sorts may do full table scan anyway



# Joins Must use Indexes

- `SELECT customer_name, total_price FROM order, customer WHERE order.customer_id=customer.id`
- Make sure there is index on customer(id)
- Joins without indexes are extremely slow
  - And they become slower rapidly with data growth



# Check how indexes are used

- MySQL May not use Index you expect it to use
  - Wrong optimizer stats or cost estimates
  - Lacking ability to do so
- `SELECT id FROM tbl WHERE a=5 ORDER BY C limit 10;`
  - If there is index on (A,B,C) it will not be used to do order by



# Use matching types

- SELECT title FROM story where id=5
  - Id is PRIMARY KEY
  - But Select does full table scan
    - Why ?
    - Because id is VARCHAR(20)
      - Data migration software created it such way
- Index can't be used for match in this case
  - Id=5 matches “5” “05” “5.0” etc



# Try to keep data shorter

- INT is better than BIGINT
  - And tiny int is even better if your data fits
- Limit maximum length for VARCHAR
  - Do not use `NAME VARCHAR(255) NOT NULL`
  - It does not full length in tables but requires in temporary tables, sort files etc.
- But use NOT NULL if possible - it saves some bits



# Use INTs

- Integers are much faster to compare, sort, group by
  - They also take much less memory
- Especially applies to JOINS
  - They cause a lot of random lookups
    - This is very SLOW with VARCHAR for MyISAM due to key compression
    - But it slower with all storage engines anyway



# Use Simple character sets

- Do not use UTF8 where you do not need it
  - Urls, emails, codes
- UTF8 has 3 times more space reserved
  - VARCHAR(100) will have 300 bytes reserved for operations if it is UTF8
- Also UTF8 comparison and sorting is much more expensive
- But DO use UTF8 for mixed charset data



# Full Table Scans

- Bad
  - If can be replaced with simple index lookup
- Good
  - Large tables and low cardinality indexes
  - Scanning even 1% of table in random order may be slower than full table scan.
  - MySQL Optimizer does not always makes the right choice



# Joins are expensive

- MySQL Uses Nested Loops for Joins
  - Meaning there are a lot of random row lookups
  - Extremely bad for IO bound workload
    - Simply bad for CPU bound workload.
- Consider having partially denormalized schema
  - Possibly in form of summary/lookup tables



# Use covering Indexes

- Data can be read from index, avoiding row read
  - Seen as “Using Index” in EXPLAIN
  - Is significantly faster
    - Especially for long row or scattered data
- `SELECT avg(age) FROM user WHERE id BETWEEN 100 and 500`
  - `INDEX(id,age)`



# Consider Prefix indexes

- Good for long fields with selective prefix
  - KEY CITY(CITY(6))
    - Assume first 6 digits are selective enough
    - Make sure there are no too frequent prefixes
      - Few cities start with “Lon” but a lot with “New”
- Using prefix for index blocks it from being used as covering index



# Hash Indexes hack

- MySQL only has native Hash indexes for MEMORY and NDB(Cluster) Tables
  - But it is quite helpful for many cases
- KEY (url\_crc)
  - SELECT hits FROM stats where URL="http://www.site.com" and url\_crc=crc32("http://www.site.com")
- Only works if you do not need ranges



# Avoid Duplicate and Redundant Indexes

- Duplicate
  - PRIMARY KEY(id), UNIQUE(id), KEY(id)
  - MySQL will create 3 physical dupe indexes
- Redundant
  - KEY(name), KEY(name,age)
  - KEY(name) can usually be removed
    - Unless you want shorter key to be available for lookups only on name.



# Fake Redundant Indexes

- But do not make a mistake
- Indexes (A,B,C) (C,B,A) and (B,C,A)
  - Are not redundant
  - Order of entries in the index is significant
- Put columns with = or IN lookup in front
- Put more selective columns in front



# Only fully matching prefix is used

- KEY(A,B,C)
  - WHERE A=5 and C=6
    - Only will use A=5 for Index lookup
  - WHERE A=5 and B>5 and C=6
    - Will only use A=5 and B=5 for index range
  - WHERE A=5 and B IN(5,6) and C=6
    - Will use full index for range lookup



# Fight Fragmentation

- Intensively updated data gets fragmented
  - This applies both to MyISAM and Innodb
  - Innodb does not fragment small rows
- Loading data in Innodb table in Random PK order can cause fragmentation
- OPTIMIZE TABLE to rebuild table
  - Do not abuse by running it too frequently
  - Locking operation even for Innodb tables



# Updating Stats

- MySQL Uses cardinality stats sometimes
  - Prefers to use actual range estimates if possible
- Statistics can become stale after massive updates
- Use ANALYZE TABLE to update it
  - OPTIMIZE TABLE Includes It
- InnoDB does not normally need it



# Cluster the data

- Make sure data accesses are as local as possible
  - Requires least amount of IO
- InnoDB automatically clusters data by primary key
- MyISAM: ALTER TABLE tbl ORDER BY <col>
  - Only does it once and updates will break it.



# Short Primary Keys w Innodb

- Innodb references rows by PK from all rows, so long PK blow up index size
- Have primary keys with Innodb
  - They are created internally anyway and 6 bytes length rather than typical value of 4
- Sequential primary keys are best
  - Btree will not get fragmented with inserts



# Queries Causing Problems

- Slow Queries
  - Easy to find by using Slow query Log
- Frequent queries
  - Causing significant load due to their number
  - Query taking 0.01 sec but you need to run 1000 of them per second
  - Hard to find in stock MySQL
  - Our patch: <http://www.mysqlperformanceblog.com/2006/09/06/slow-query-log-analyzes-tools/>



# Other way to find queries

- General query log
  - `--log`
- Processlist pooling
  - `mysqladmin processlist -i1 | grep -v Sleep`
- Application level profiling
  - Application can summarize and log queries



# Is query Inefficient ?

- How much data application uses ?
- How many rows query returns
  - If you show 10 rows, 3 columns each it is about as much as query should return
  - Application may do extra processing but it should be justified
- Do not run **SELECT \* FROM TABLE**
  - To only show couple of rows



# Is query complicated

- Look at information query must use to generate result set
  - `SELECT count(*),color FROM cars GROUP BY color`
    - Query must use every row in the column to create result set
- Summary tables are way to fix such queries



# Is all filtering done by index

- `SELECT count(*),color FROM cars WHERE year=1990 GROUP BY color`
  - `KEY(year)`
  - All where clauses are resolved by Index lookup
  - Can't be improved from number of rows standpoint
    - Extending key to `(year,color)` will improve performance due to covering index



# Dealing with LIMIT

- Great way to result result set
  - LIMIT 0,10 - first 10 rows
- Makes it important to be efficient to stop after first few rows are generated
  - Using index for ORDER BY/GROUP BY
- Beware of Large Limit
  - LIMIT 1000000,10 can be slow
  - Even Google does not let you to page 100000



# GROUP BY Optimization

- If large number of groups use SQL\_BIG\_RESULT hint
  - Use FileSort instead of temporary table
- Skip-Index GROUP BY in MySQL 5.0
  - SELECT MAX(B) FROM TBL GROUP BY A
    - KEY(A,B)
- Add ORDER BY NULL if do not need order
  - MySQL Sorts GROUP BY result by default



# ORDER BY Optimization

- Index based Sort
  - Great with LIMIT or for index covered queries
    - Otherwise can be slower than extra sort
- External Sort
  - In memory (**sort\_buffer\_size**)
    - Using quicksort
  - External, using temporary files
    - Multi-way merge sort



# ORDER BY Optimization

- **read\_rnd\_buffer\_size**
  - Helps MyISAM tables to speed up read after sort
- **max\_sort\_length**
  - Prefix used to sort BLOB columns
- **max\_length\_for\_sort\_data**
  - Row size to store instead of row pointer in sort file. Allows to avoid random table reads



# Forcing Join Order

- Optimizer Needs to select order of tables to perform join
  - `SELECT * FROM A,B WHERE A.COL1=B.COL2`
    - Can start from A and look match in B or in reverse order
- Sometimes optimizer does the wrong choice
  - `SELECT STRAIGHT_JOIN * FROM A,B WHERE A.COL1=B.COL2`



# Forcing Index

- Sometimes there are multiple index choices any MySQL Optimizer is confusing
  - May select index (A) if index (A,B) is possible
- Hints for Index usage
  - USE INDEX (I1,I2,I3) - Use these indexes
  - FORCE INDEX(I1) - Force using this index
  - IGNORE INDEX (I1,I2,I3) - Ignore these indexes
- Applies to Lookup, not ORDER BY



# Be careful with Subqueries

- MySQL Optimizer is very limited in Subquery Handling
  - Subqueries in FROM Clause are always materialized
    - And never get any indexes
  - Many other subquery types are optimized suboptimal way
    - Optimization effort is on a way



# Newer Features

- Optimizer is always improved
- Optimizer limits are overtaken, new execution methods added.
- MySQL 5.0 Adds many smaller optimizations and
  - Greedy Optimizer - Fast handling of many tables
  - Index Merge - Using Multiple Indexer per table



# EXPLAIN

- Describes how query is executed
  - Not overly detailed but gives enough info for simple queries
- Shows Estimated row numbers
  - Can be very different in real life
- Sometimes can show wrong plan
  - Profiling with SHOW STATUS is good way to check if it is correct.



# EXPLAIN

- Number one query optimization assistant
- Shows order of tables in join, access method, indexes used, estimated number of rows and some extra info
- Do check manual It is very well described
  - <http://dev.mysql.com/doc/refman/5.0/en/explain.html>





# Operating System Tuning



**OPEN SOURCE DATABASE  
CONFERENCE 2006**

Learn • Share • Network • Benefit

# Selecting Operating System

- MySQL Supports huge amount of platforms
  - But few are frequently used in production
- **Windows** - Mostly used in single user and smaller installations
- **Linux** - Most popular Server operation system
- **Solaris** - Was popular years ago, now gaining popularity with Sun support.



# What MySQL Needs from OS

- Good Threads Support
  - Especially with Multiple CPU scaling
- Fast System Calls
  - Especially for MyISAM with OS based cache
- Fast IO Subsystem
  - For IO Bound Workload



# OS Tuning

- In Most cases no tuning is required
- Network tuning
  - Handling many new connections from
  - Large Network Buffer
- Virtual Memory
  - Avoiding caches
- File System optimizations



# Virtual memory

- MySQL Daemon Must Not be Swapping
  - **--memlock**
    - lock it in memory
  - **--innodb-flush-method=O\_DIRECT**
    - Bypass caching so reduce VM Pressure (Linux)
  - **/proc/sys/vm/swappiness**
    - May adjust to reduce swapping



# FileSystem Tuning

- Which FileSystem to use ?
  - Different benchmarks different winners
  - Journaling is good to have
  - Linux - EXT3, ReiserFS, XFS
- Configuration
  - Journaling mode
    - Ordered or Writeback (enough for Innodb)
  - Do not update access time **-o noatime**





# Hardware tuning



**OPEN SOURCE DATABASE  
CONFERENCE 2006**

Learn • Share • Network • Benefit

# Choosing hardware

- Scale UP or Scale Out
  - Existing systems may be designed with single box it mind
- Maintainability Requirements
  - 50 cheap boxes will need more time than 10 standard ones
  - Is your system designed to minimize recovery efforts needed etc ?



# CPUs

- Use 64bit CPUs with 64bit OS
  - Especially with large amount of memory
- X86-64 based architectures offer best price performance
  - Opteron or Intel Woodcrest based are both decent
- DualCore is great HypperThreading is so-so
- CPU matters if CPU is bottleneck



# Faster CPUs or More CPUs ?

- Single Query is mainly run on single CPU
- Fast CPUs reduce response times for single query
- Many CPUs allow multiple queries to run truly in parallel
  - Good with multiple concurrent queries
  - Scalability problems depending on workload and storage engine.



# Memory

- Is workload going to be CPU Bound or IO Bound
  - Having working set in memory solves so many problems
- If you can build system with working set fitting in memory - do it.
- Having more memory than you need is a waste (will not give Any extra speed)



# IO Subsystem

- Reads, Writes and Sync IO
- Reads
  - Data does not fit in memory and warmup
- Writes
  - Happen with updates even with small data
- Synchronous IO
  - Happens with Transactional Storage Engines



# Sequential IO and Random

- Sequential IO
  - Full Table Scans, Sorts, Bulk Loads, Log writes
  - Good speed. Less frequently the problem
- Random IO
  - Index Lookups, Joins
  - Most frequently reason for bottleneck



# Parallel IO

- Are 4 7200 RPM SATA drives better than 1 15000 RPM SAS ?
- How parallel is your Disk IO
  - Look at `iostat -x 10` to see device load
- Multiple OLTP Queries - Parallel
- Batch Jobs, Replication
  - Non-Parallel
- Fast drives help response time



# Estimating IO

- Hard to predict due to caching, fragmentation, clustering
  - Benchmark on real data set
- Drive can do 100-250 IO/Sec
- 70-120MB Sequential Read/Write Speed
- RAID, SAN can't magically change physics



# RAID

- Different tasks different RAID Level
  - RAID10 is good in general
    - Implementation on some RAID cards is poor
  - RAID5
    - Decent reads but slow random writes, long recovery
  - RAID1
    - Logs, OS
  - RAID0 - Temporary files, Slaves



# Advanced RAID

- RAID Stripe Size
  - A lot depends on internal implementation
  - 64K-256K are often optimal
- Raid Read-Ahead
  - May improve performance a bit
- Writeback Cache
  - Great for transactional Storage Engines
  - Make sure you have Battery Backup



# NAS And SAN

- NAS has caused various problems
  - Especially for transactional Storage Engines
  - Try at your own risk
- SAN
  - Will not buy you better performance for money
  - May help with manageability, recovery
    - I think there are better ways to target these





# Application Architecture Design



**OPEN SOURCE DATABASE  
CONFERENCE 2006**

Learn • Share • Network • Benefit

# Designing Architecture

- Get it right from beginning
  - Often most expensive to change later on
- Defines Hardware/software purchases
  - Makes it even more expensive to change
- Database is just one component which needs attention



# Identify parameters

- Application Scale and growth ?
  - Transaction rate, Database size etc
- High availability requirements ?
- Maintainability ?
- Scalable architectures are often more complicated
  - Over engineering is expensive



# Caching

- Best way to optimize query is to skip it execution at all
  - Applies to any other operations
- Many layers of cache are normally used
  - Web Cache (is Squid)
  - Some form of Object cache
  - Database Caches etc



# Caching questions

- How large is the data set ?
  - Small data sets can be cached on each node
- How expensive is cache miss ?
  - Expensive data can be cached on disk
- How cache will be synchronized
  - TTL Based
  - Write Update/Write Invalidate



# MySQL Query Cache

- Simple and transparent
- Caches on query level (can be slow)
- No multi-get API
- Very coarse invalidation
  - Any update to the table removes all queries involving this table
- Poorly scales with large cache sizes
- Good solution for smaller projects



# MemCached

- Distributed Cache Solution
- Very fast and efficient
- Object level cache
- Can be used TTL Based or Write-Update
- Multi-Get API
- Used by LiveJournal, Wikipedia, FaceBook etc



# Distributing your data

- Single server can hold certain amount of data
  - And handle certain amount of queries
- Scaling by upgrading server is expensive
- And limited anyway
- So need to split your data and load to multiple servers



# MySQL Replication

- Copy Dataset to multiple servers
  - Can only replicate only portion of data if needed
- Single master only
  - Multi-Master to come some time
- Can be configured master-master, chain or circular way



# MySQL Replication

- Asynchronous
  - Slave gets data with (undefined) delay
  - Loss of master is loss of data
- Statement level based
  - Before MySQL 5.1
- Can be row level based
  - MySQL 5.1+



# MySQL Replication Goods

- Simple to configure and run
- Fast, low overhead on master
- Great scalability for readers
- Easy to use from application
  - Write to the master
  - Read from the slave
  - Handle replication delay somehow



# MySQL Replication Bads

- Limited write scalability
  - Can replicate fraction of write load server can handle as happening in single thread
- Waste of space
  - Think about replicating 500Gb database in 10 copies
- Waste of database cache memory
  - All nodes cache about same thing



# Partitioning

- Partition your data
  - And replicate for High Availability
- Instead of 10 copies 100GB each
  - Use 5 Master-Master pairs, 20GB each
- Number of copies per chunk depends on workload and partitioning type



# How to Partition

- Role based partition
  - “Authentication Server”, “Billing Server”
  - Isolate tables in groups which you do not run joins between
    - Light duty joins possible with FEDERATED engine
- Horizontal partitioning
  - Chop your large tables (ie by user\_id)
  - Minimize cross partitioning access



# Partitioning Problems

- Complicates code
- Need to think in advance
  - If you will need to join these tables or not
- Cross partition accesses are expensive
- May require double data store for some configurations
  - ie storing “link” between 2 users



# Single Writer Master-Master

- Bi-Directional replication  $A \leftrightarrow B$
- Both servers configured Master and Slave
- Writes go only to one of the servers
- Reads are done from both
- Can fallback to second server in case of crash
- Almost zero downtime upgrades and maintenance



# Why not to write on both ?

- It is higher performance but more complicated
- Need to make sure writes can't cause conflicts
- Pattern: Server A is master for set of databases, Server B for other set



# Things to watch out

- Replication is asynchronous
  - Re-cloning or full comparison are only 100% secure ways to recover crashed slave
- Unintended writes to “Slave” can cause inconsistency
  - Use `--read-only` settings to limit writes to slave
- Make sure single server can handle load



# Using multiple tables

- May use multiple groups of tables inside same host
  - 100 1GB tables is better than 100GB table
- Data Clustering
- Easier to maintain
  - ALTER TABLE/OPTIMIZE TABLE are blocking
- Easier to move across servers if needed
- May require more complex code though
- MySQL 5.1 Partitioning - can help some issues.



# MySQL 5.0 new features

- Triggers
  - Reduce number of roundtrips
- Stored Procedures
  - Reduce number of roundtrips
- Views
  - Can help to abstract from database schema
    - And change it without application changes
  - Can be faster than Derived Tables.





# Deployment and Operations



**OPEN SOURCE DATABASE  
CONFERENCE 2006**

Learn • Share • Network • Benefit

# Deploying MySQL

- Use Quality hardware
  - Database corruption is not web server crash
- Limit number of hardware configurations
- Keep Standby server of best configuration
  - Even most expensive hardware can fail
- Have Hot-swap Drives if you need fast recovery



# Deploying MySQL

- Have main actions scripted
- Automate configuration
  - So you can setup server quick.
- Use LVM or other volume manager
  - Great for backups and other means
- Have backups
  - Replication and RAID do not solve all problems
- Track configuration and version changes



# Reasons for backups

- Hardware crashes and failures
- OS and MySQL Bugs can corrupt data even on most expensive RAID
  - And there is a chance for these to be replicated
- Software and User errors
  - `DELETE FROM accounts WHERE id;`
- Security breaches



# Backup Strategies

- Physical level backups are best
  - Fast to backup and recover
- Check backup
  - So you do not have unnoticed database corruption
- Perform recovery trials
- Make sure you have binlog offset stored
  - Can perform point in time recovery



# Backup Details

- Creating backup
  - **FLUSH TABLES WITH READ LOCK;**
  - **SHOW MASTER STATUS;**
  - **<create snapshot>**
  - **UNLOCK TABLES;**
- Make sure you have binary logs securely stored for point in time recovery
  - Better to cover couple of backup generations



# Backup Security

- Pull backups from Server rather than push
- Or push to specially dedicated area
- Compromising DB Server should not be able to corrupt your backups
- Have offsite backups for high security levels
- Make sure backups are running





# Methodology



**OPEN SOURCE DATABASE  
CONFERENCE 2006**

Learn • Share • Network • Benefit

# Working as consultants

- Deliver result quickly
- Save customers money
- Predict and prevent future problems



**OPEN SOURCE DATABASE  
CONFERENCE 2006**

Learn • Share • Network • Benefit

# Get global picture

- What are the problems ?
  - Response time, throughput, scaling etc
- Most critical transactions ?
- What is current application size ?
  - And how it is expected to growth ?
- What skills exist in the company ?
  - And resources clients want to apply
- What hardware is currently used ?



# Low hanging fruits

- OS and Server configuration
- Query Optimizations
- Minor schema fixes
  - Indexes, Storage engines
- Simple Caching
  - Query Cache etc
- Hardware upgrades
  - Especially on lower end



# Planning for future

- Can current schema and architecture deliver the requirements
  - And how expensive it would be ?
- How much time is left before it fails ?
- What are application improvements are possible
  - How much would they could
  - Which benefits would they bring



# Executing

- Selecting changes which will be done at this stage
- Selecting who will do these changes
  - We ? Customer ? Third Party ?
- Implementation
- Accessing results
  - Is everything implemented correctly ?
  - Do we observe improvements expected ?



# Time for Questions

- Ask your questions now !
- Or catch me during the conference
- Email me
  - [pz@mysqlperformanceblog.com](mailto:pz@mysqlperformanceblog.com)
- Consulting requests
  - [consulting@mysqlperformanceblog.com](mailto:consulting@mysqlperformanceblog.com)

