

# MySQL Performance Tuning

A large, faint, light gray watermark of the MySQL logo is visible on the left side of the slide, partially overlapping the text.

Ryusuke Kajiyama  
MySQL Senior Evangelist  
Sun Microsystems / MySQL APAC Business

# Performance, Reliability, Ease of Use

## Oracle9i and MySQL top throughput

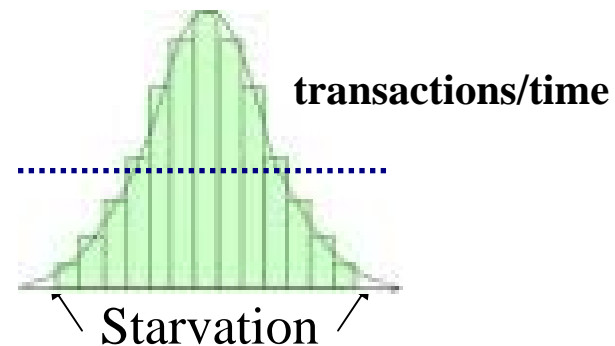


Throughput is in returned Web pages per second from the application server. Number of users is number of concurrent Web clients driving the load. Response time is the time to complete the six bookstore user action sequences, weighted by frequency of each sequence in the mix. All tests were conducted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

- *eWeek's* Database Benchmark Test
  - MySQL has the best overall performance and scalability (matching Oracle)
  - MySQL excelled in stability, ease of tuning, and connectivity
  - MySQL offered the highest throughput

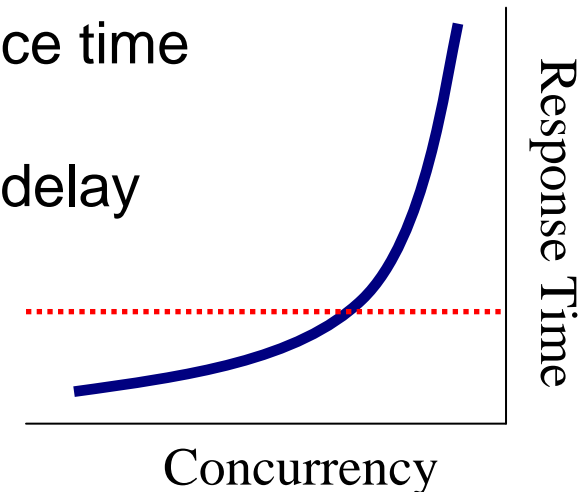
# Defining Performance

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



# 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” - queueing delay grows rapidly when system getting close to saturation
- Need to improve queueing delay or service time to improve performance
- Improving service time reduces queueing 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
  - Disk IO latency
  - Network traffic
  - Load Average
  - Number of running queries
  - 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
  - Need to be able to read results correctly
- Typical Errors
  - Testing with 1GB size with 100G in production
  - Using uniform distribution
    - “Harry Potter” ordered as frequent as Zulu dictionary
  - Testing in single user scenario

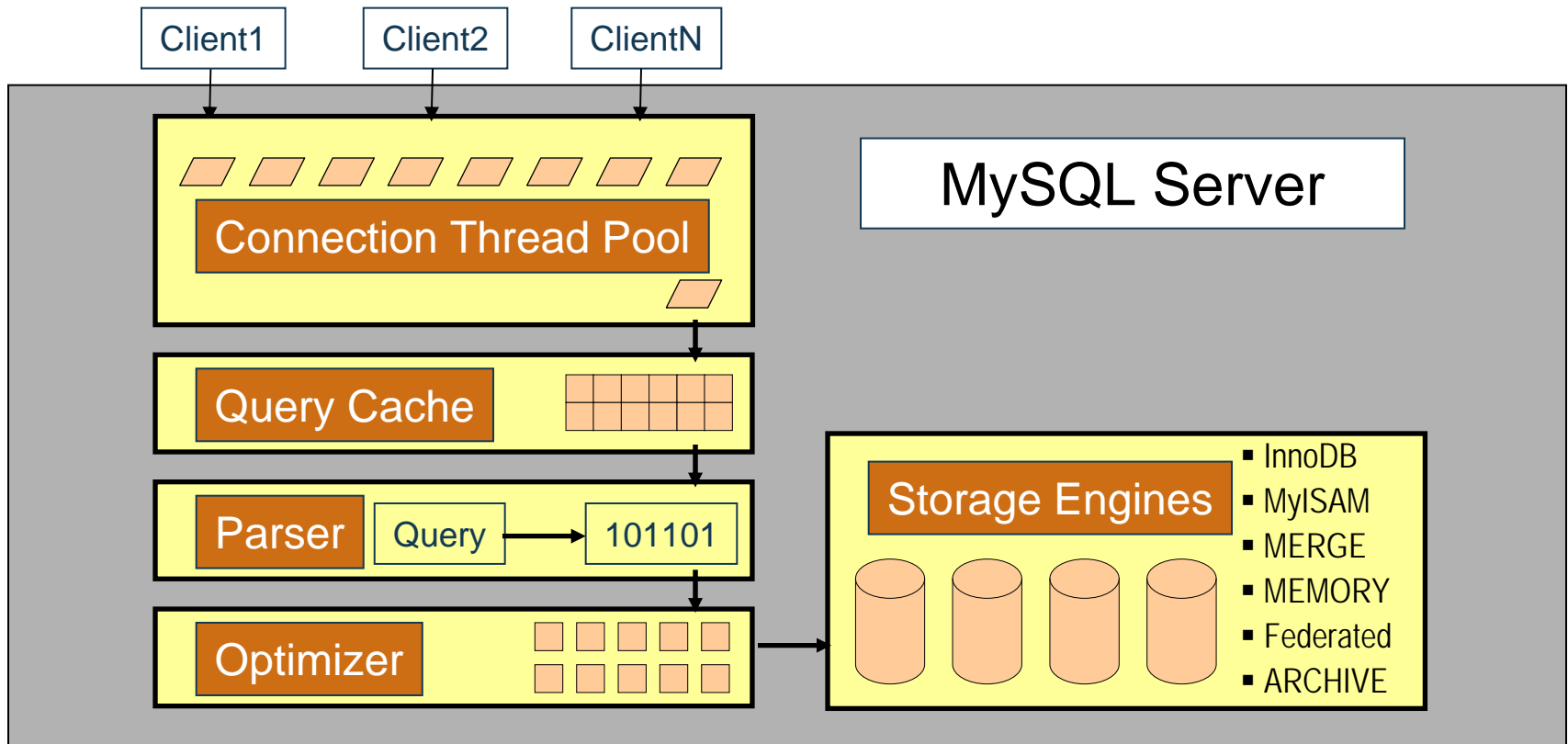
# Business side of optimization

- 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%
  - Is peak traffic requirements 100x average or just 3x ?
- 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 ?

# Performance Tuning Tips

1. Server Configuration
2. Storage Engine Selection & Tuning
3. Schema & Query Optimization
4. Server Operation Tuning
5. Hardware Selection & Tuning

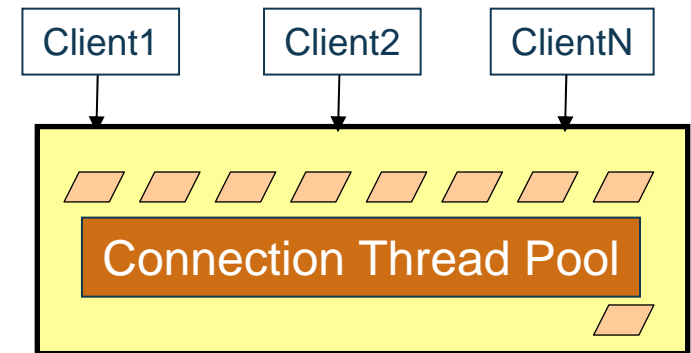
# MySQL Server Architecture



# Server Connections & Threads

## my.cnf settings:

- **max\_connections (100)**
  - number of connections server will allow. May run out of memory if too high
  - Typically **1000+** for web clusters, leave **2x** room to grow, burst
- **thread\_cache\_size (8)**
  - Keep up to this amount of threads “cached” after disconnect
  - Typical setting **max\_connections/3**



## mysql> show status;

- **Max\_used\_connections**
  - check if it matches **max\_connections**, too low value or sign of overload
- **Threads\_created**
  - thread\_cache misses
  - should be low.

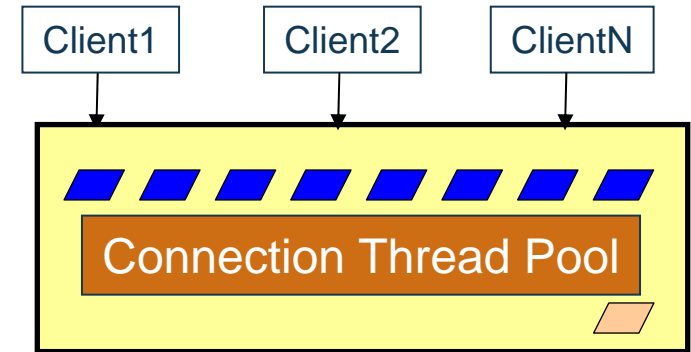
# Connection Thread Work Buffers

## my.cnf settings:

- **sort\_buffer\_size (2M)**
  - Memory to allocate for sort. Will use disk based sort for larger data sets. Often fine at **512K** or **1M**
- **other buffers, read, read\_rnd, etc...** smaller defaults often OK

Estimate, **thread memory usage** =  
**max\_connections** x  
**(thread\_buffers + thread\_stack\_size)**  
x ½ (or average buffer allocation)

- 1GB easy for 1000 connections



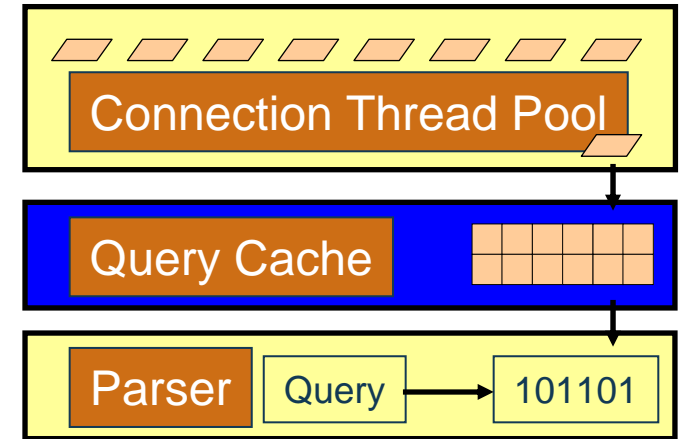
## mysql> show status;

- **Sort\_merge\_passes** -
  - number of passes made during file merge sort.
  - check if file sort needs to be done at all
  - use index if possible

# Server Query Cache

## my.cnf settings:

- **query\_cache\_size (0)**
  - Amount of memory to use for query cache
  - Typically **32M** is fine, some databases need **128M**
- **query\_cache\_type (ON)**
  - Worst case performance overhead is about 13%
  - Favors servers with higher SELECT/WRITE ratios

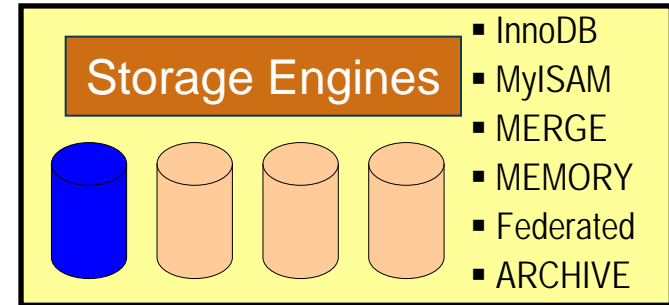


## mysql> show status;

- **Qcache\_hits, Qcache\_inserts**
  - hits/inserts cache hit ratio, if small, **maybe disable query cache**
- **Qcache\_lowmem\_prunes**
  - times older queries were removed due to low memory, need to increase query\_cache\_size if high

# InnoDB Storage Engine

- InnoDB provides ACID transactions:
  - Row-level multi-versioning
  - Configurable isolation levels
  - Row-level locking



## Tablespace In Memory

- Data and indices are available in a cache.
- Data modification happens in memory
- UNDO log maintained in case of ROLLBACK.
- TX checkpoints cause data and UNDO log to be written to disk

## Logs In Memory

- Tracks actions from all active TXs

## Tablespace On Disk

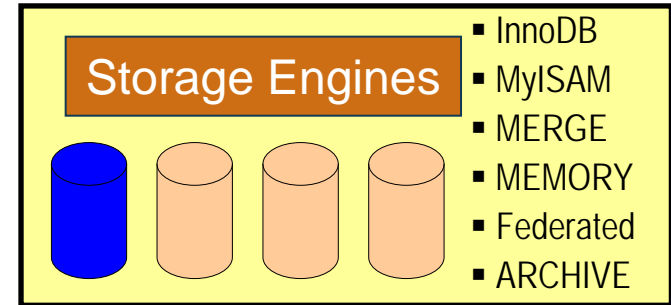
- Data can be written to disk during or after a TX
- If data written to disk DURING a TX, an UNDO log is maintained in case of ROLLBACK.

## Logs On Disk

- TX actions written to REDO log BEFORE a transaction is committed
- TX commit occurs as soon as REDO log on disk
- Background thread "writes" data from tablespace memory to disk by executing REDO logs.

# InnoDB Performance Tips

- **innodb\_buffer\_pool\_size**
  - 80% of memory on InnoDB only system
  - caches data & indexes unlike MyISAM
- **innodb\_log\_file\_size**
  - set from 25% to 100% of `innodb_buffer_pool_size`
  - check how frequently log file changes (mtime), too fast then checkpointing
  - large values increase crash recovery time
- **innodb\_flush\_log\_at\_trx\_commit**
  - 1 (slow) will flush (fsync) log at each commit. **Truly ACID**
  - 2 (fast) will only flush log buffer to OS cache on commit, sync to disk once/sec.
  - 0 (fastest) will flush (fsync) log every second or so



**mysql> SHOW INNODB STATUS;**

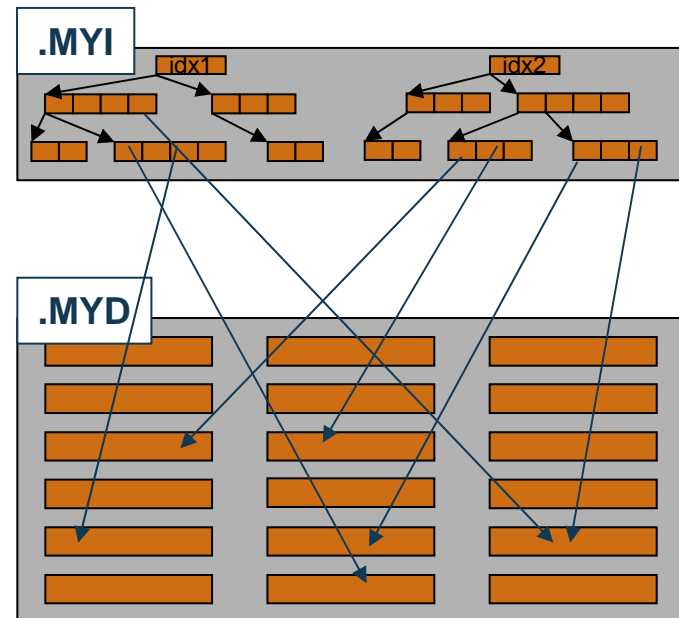
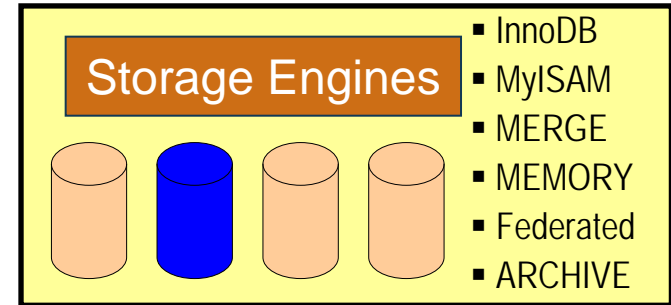
Great way to see what is going on inside InnoDB, hard to parse

- File IO
- Buffer Pool
- Log activity
- Row activity

In MySQL 5.0, some variables exported to **SHOW STATUS**

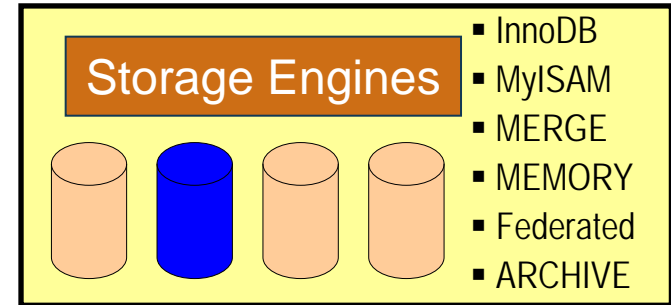
# MyISAM Storage Engine

- no ACID transactions, will be corrupted on power failure
- small disk and memory footprint, often 25%-50% less than other RDBMS
- packed indexes, works without indexes, FULLTEXT, RTREE
- table locks, concurrent inserts
- read-only packed version
- only index is cached by MySQL, data by OS
- Fast load, index build speeds
- Tables can be copied at the OS level
- Supports MERGE partitioning



# MyISAM Performance Tips

- **key\_buffer\_size (8M)**
  - For MyISAM only server 25-33% of memory is typical
- **myisam\_sort\_buffer\_size (8M)**
  - set high for index builds, as large as largest index is in table optimally
- **myisam\_recover=FORCE,BACKUP**
  - so you can sleep at night
- **delay\_key\_write=ALL**
  - delay key writes for extra write performance
  - higher chance of corruption on crash



**mysql> show status like 'key%'**

- **Key\_read\_requests, Key\_reads**
  - Key\_reads/Key\_read\_requests – key cache miss ratio
  - watch for Key\_reads/sec, match against your io system
- **Key\_write\_requests, Key\_writes**
  - miss ratio is typically larger
- **Key\_blocks\_unused**
  - maybe too much memory allocated

# Schema Design

- Normalization
  - good for OLTP, writes
  - data redundancies eliminated
  - join performance penalty
  - smaller total data set
  - E/R diagram clean translation
- Denormalization
  - good for OLAP, reporting
  - data redundancies across tables for better indexing
  - maybe eliminate joins
- Data types
  - use tinyint, smallint, mediumint, save space!
  - join columns same data type
  - varchar(64) instead of char(64)
  - declare NOT NULL if true
  - varchar(64) instead of varchar(255)
  - INT not DECIMAL(9)
- Indexing
  - multi-column
  - ordered BTREE
  - index prefixes
  - covering

# Indexing

- Index helps to speed up retrieval but expensive to maintain
  - smaller indexes, index prefixes like **index(name(8))** good
- 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, needs better than 1/3
- Define **UNIQUE** indexes as **UNIQUE**
- Index, being prefix of other index is rarely good idea
  - remove **key(a)** if you have **key(a,b)**
- **BTREE** index returns in sorted **ORDER**
  - select \* from t where b=5 order by c ... **key(b,c)** optimal
- Covering indexes even faster, no row data fetched
  - select c from t where b=5 ... **key(b,c)** optimal
- **OPTIMIZE TABLE** ... to compact & sort indexes

# Adjusting SQL 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 USE INDEX(PRIMARY)**
  - index hints, rarely use in MySQL, common in other databases
  - force index, strongly prefers index use over table scan
- **ANALYZE TABLE ...**
  - not usually needed, but run if optimizer needs help

# Server Operation: Slow 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;**

# 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
- HyperThreading – gives improvement in most cases

# Database/OS Memory Buffers

- Data and indexes cached in Database or OS buffers
- Provided automatically, usually present
  - MySQL server and OS Server settings.
- Fully transparent
- Very important to take into account
  - Access to data in memory usually 1000s of 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

# 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
- SAN, NAS, NFS, often external RAID, test carefully!
- Use larger RAID chunk (256K-1MB)
- Compute your IO needs – drive can do (150-250 IO/sec)
- Place Innodb logs on dedicated RAID1 if a lot of devices
  - otherwise sharing works well
  - OS could use the same drive

# 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

# Resources

- MySQL Online Manual – great source of information
  - <http://dev.mysql.com/doc/mysql/en/index.html>
- SysBench - Benchmark and Stress Test tool
  - <http://sourceforge.net/projects/sysbench>
- MySQL Benchmarks mailing list
  - [benchmarks@lists.mysql.com](mailto:benchmarks@lists.mysql.com)

## Q&A

### **More Performance Tuning assistance:**

[www.mysql.com/consulting](http://www.mysql.com/consulting)

[www.mysql.com/training](http://www.mysql.com/training)

Ryusuke Kajiyama

Sun Microsystems / MySQL APAC Business

[rkajiyama@mysql.com](mailto:rkajiyama@mysql.com)