# Tuning MySQL

## It's About Performance

Bill Karwin

Senior Database Architect @ SchoolMessenger

MySQL Worldwide Conference & Expo

Percona Live

MySQL Tuning

# INTRODUCTION

# What Are We Talking About?

- Configuration variables.
  - We change them to allocate resources for specific features in MySQL.
  - Or enable/disable optional behavior.
  - Make settings persistent by editing /etc/my.cnf

# Types of Tuning Changes

- Many configuration variables can be GLOBAL or SESSION.
  - Sessions copy global values at connect time.
  - Sessions can change tuning values for the scope of one connection.

# Tuning Advantages

- No changes required to schema.
- No changes required to code.
- Some tuning changes possible without restarting mysqld.

# Agenda

- InnoDB Buffer Pool
- InnoDB Redo Log
- InnoDB IO Capacity
- InnoDB Other
- Optimizer
- Logging
- Replication
- Connections and Threads
- Tables
- Query Cache
- Operating System
- Tuning Tools
- Monitoring Tools
- Tuning vs. Architecture

MySQL Tuning

# INNODB BUFFER POOL

# InnoDB Buffer Pool

- Largest single use of memory in the server
- In-memory cache of InnoDB data, indexes, undo pages, change buffer – anything that is stored in *pages* in the tablespace.

# Buffer Pool Size

- Enough to hold most frequently-used pages.
  - `innodb_buffer_pool_size = 10240M`

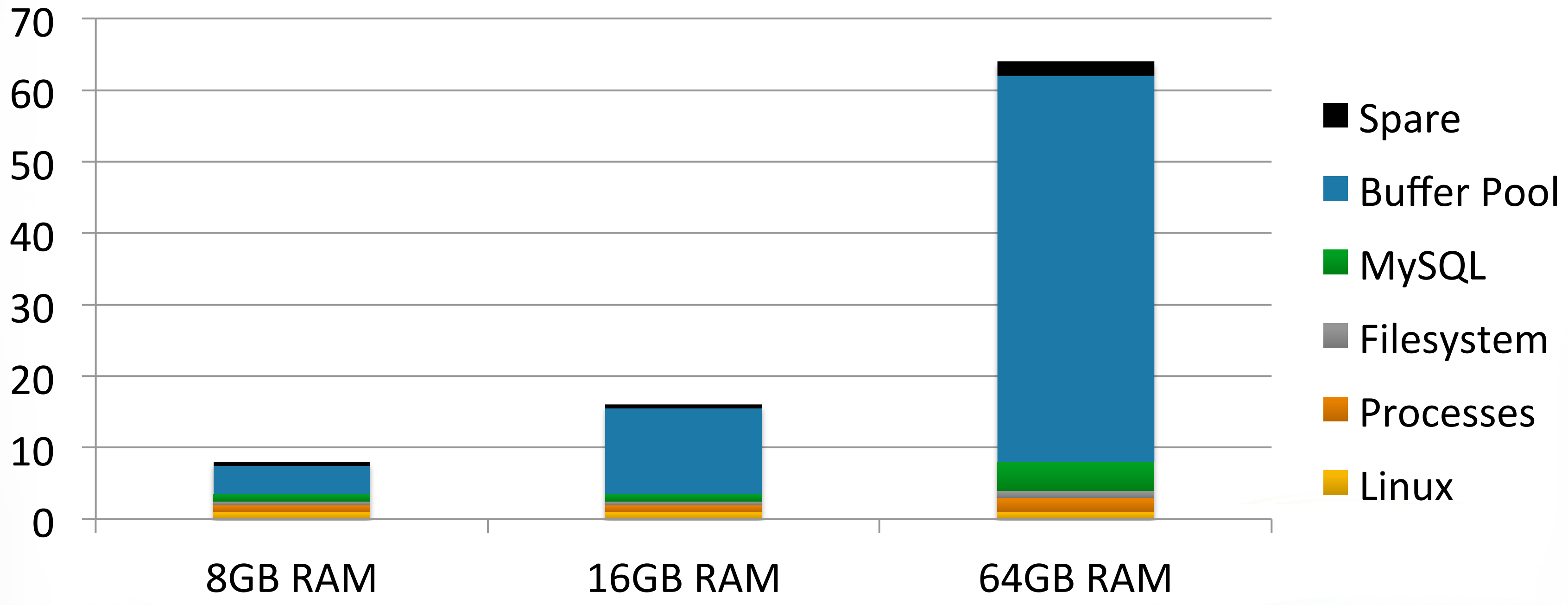
# Buffer Pool Size

- Watch the ratio of page reads to page reads that needed I/O:

```
mysql> SHOW GLOBAL STATUS LIKE
'Innodb_buffer_pool_read%s';
+----------------------------------+-------+
| Variable_name                    | Value |
+----------------------------------+-------+
| Innodb_buffer_pool_read_requests | 30887 |
| Innodb_buffer_pool_reads         | 411   |
+----------------------------------+-------+
```

*98.67% efficiency*

# Buffer Pool as a Portion of RAM



# Buffer Pool Sizing Tips

- Assume the buffer pool uses +10% RAM for internal metadata.
- You can reach a point of diminishing returns.
  - A small database doesn't need a large buffer pool.
  - Pages are not stored more than once.
- Don't oversize it and cause swapping!

# What’s In My Buffer Pool?

MySQL or Percona Server 5.6 – 5.7

```
USE information_schema;
SET @page_size = @@innodb_page_size;
SET @bp_pages = @@innodb_buffer_pool_size/@page_size;

/* MySQL or Percona Server 5.6 - 5.7 */
SELECT P.TABLE_NAME, P.PAGE_TYPE,
CASE WHEN P.INDEX_NAME IS NULL THEN NULL WHEN P.TABLE_NAME LIKE '`SYS_%' THEN P.INDEX_NAME WHEN
P.INDEX_NAME <> 'PRIMARY' THEN 'SECONDARY' ELSE 'PRIMARY' END AS INDEX_TYPE,
COUNT(DISTINCT P.PAGE_NUMBER) AS PAGES,
ROUND(100*COUNT(DISTINCT P.PAGE_NUMBER)/@bp_pages,2) AS PCT_OF_BUFFER_POOL,
CASE WHEN P.TABLE_NAME IS NULL THEN NULL WHEN P.TABLE_NAME LIKE 'SYS\_%' THEN NULL ELSE
ROUND(100*COUNT(DISTINCT P.PAGE_NUMBER)/CASE P.INDEX_NAME WHEN 'PRIMARY' THEN TS.DATA_LENGTH/
@page_size ELSE TS.INDEX_LENGTH/@page_size END, 2) END AS PCT_OF_INDEX
FROM INNODB_BUFFER_PAGE AS P
JOIN INNODB_SYS_TABLES AS T ON P.SPACE = T.SPACE
JOIN TABLES AS TS ON T.NAME = CONCAT(TS.TABLE_SCHEMA, '/', TS.TABLE_NAME)
WHERE TS.TABLE_SCHEMA <> 'mysql'
GROUP BY TABLE_NAME, PAGE_TYPE, INDEX_TYPE;
```

# What’s In My Buffer Pool?

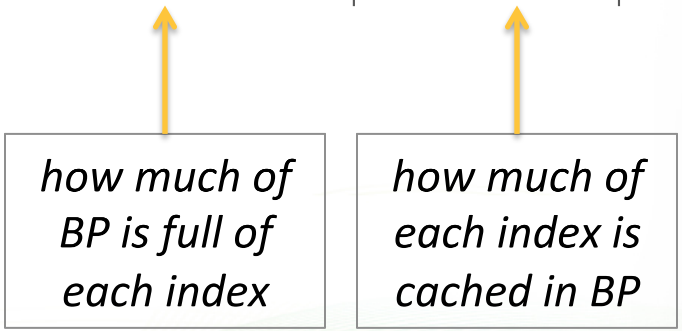
Percona Server
5.1 – 5.5

```
USE information_schema;
SET @page_size = @@innodb_page_size;
SET @bp_pages = @@innodb_buffer_pool_size/@page_size;

/* Percona Server 5.1 - 5.5 */
SELECT P.TABLE_NAME, P.PAGE_TYPE,
CASE WHEN P.INDEX_NAME IS NULL THEN NULL WHEN P.TABLE_NAME LIKE '`SYS_%' THEN P.INDEX_NAME WHEN
P.INDEX_NAME <> 'PRIMARY' THEN 'SECONDARY' ELSE 'PRIMARY' END AS INDEX_TYPE,
COUNT(DISTINCT P.PAGE_NUMBER) AS PAGES,
ROUND(100*COUNT(DISTINCT P.PAGE_NUMBER)/@bp_pages,2) AS PCT_OF_BUFFER_POOL,
CASE WHEN P.TABLE_NAME IS NULL THEN NULL WHEN P.TABLE_NAME LIKE 'SYS\_%' THEN NULL ELSE
ROUND(100*COUNT(DISTINCT P.PAGE_NUMBER)/CASE P.INDEX_NAME WHEN 'PRIMARY' THEN TS.DATA_LENGTH/
@page_size ELSE TS.INDEX_LENGTH/@page_size END, 2) END AS PCT_OF_INDEX
FROM INNODB_BUFFER_PAGE AS P
JOIN INNODB_SYS_TABLES AS T ON P.SPACE = T.SPACE
JOIN TABLES AS TS ON (T.SCHEMA, T.NAME) = (TS.TABLE_SCHEMA, TS.TABLE_NAME)
WHERE TS.TABLE_SCHEMA <> 'mysql'
GROUP BY TABLE_NAME, PAGE_TYPE, INDEX_TYPE;
```

# What's in My Buffer Pool?

```
+--------------+------------------+------------+-------+--------------------+---------------+
| TABLE_NAME   | PAGE_TYPE        | INDEX_TYPE | PAGES | PCT_OF_BUFFER_POOL | PCT_OF_INDEX |
+--------------+------------------+------------+-------+--------------------+---------------+
| NULL         | FILE_SPACE_HEADER | NULL      |     1 |               0.00 |          NULL |
| NULL         | IBUF_BITMAP      | NULL       |     1 |               0.00 |          NULL |
| NULL         | INODE            | NULL       |     1 |               0.00 |          NULL |
| `test`.`foo` | INDEX            | PRIMARY    |  2176 |               3.32 |         98.37 |
| `test`.`foo` | INDEX            | SECONDARY  |  2893 |               4.41 |         88.47 |
+--------------+------------------+------------+-------+--------------------+---------------+
```



*how much of BP is full of each index*

*how much of each index is cached in BP*

# Buffer Pool Instances

- Many threads can queue up, in contention for exclusive access to the buffer pool.
- Scalability issue more than performance issue.

# Buffer Pool Instances

- Split the buffer pool into a fixed number of sub-pools, and distributes pages among them.
  - `buffer_pool_instances = <integer>`

# How Many Buffer Pool Instances?

- Typically set this to the number of CPU cores.
- Default BP instances = 1
  - MySQL 5.6 auto-defaults to 8 when BP > 1GB
- You still specify *total* RAM used in `innodb_buffer_pool_size`,
  - Automatically splits evenly between BP instances.

# InnoDB Buffer Pool Save & Restore

- After a server restart, an empty buffer pool causes low performance until it “warms up.”

# InnoDB Buffer Pool Save & Restore

- Dump & restore automatically:
  - SET innodb_buffer_pool_dump_at_shutdown = ON;
  - SET innodb_buffer_pool_load_at_startup = ON;
- Dump & restore manually (e.g. in an EVENT)
  - SET innodb_buffer_pool_dump_now = ON;
  - SET innodb_buffer_pool_load_now = ON;

# InnoDB Buffer Pool Save & Restore

```
CREATE EVENT mysql.buffer_pool_dump
ON SCHEDULE EVERY 1 HOUR
DO
   SET GLOBAL
   innodb_buffer_pool_dump_now=ON;
```

*good if you anticipate crashes!*

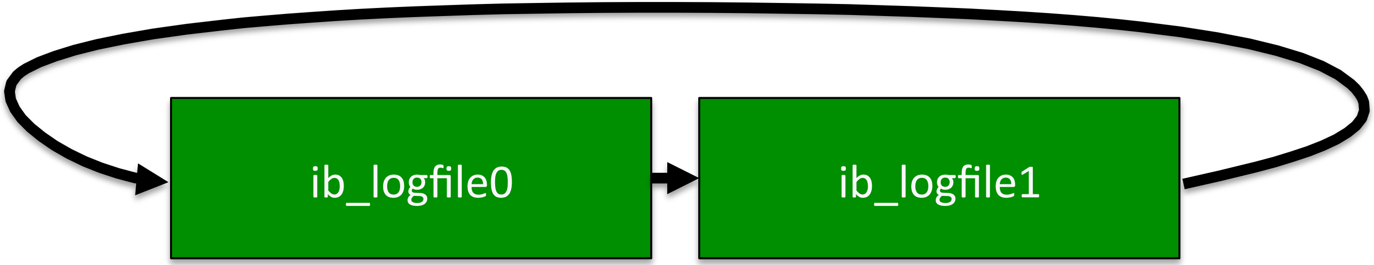
MySQL Tuning

# INNODB REDO LOG

Simon Law

# InnoDB Log File

- The log file records changes to InnoDB pages.
- The file(s) are fixed size, and are overwritten.



# InnoDB Log Size

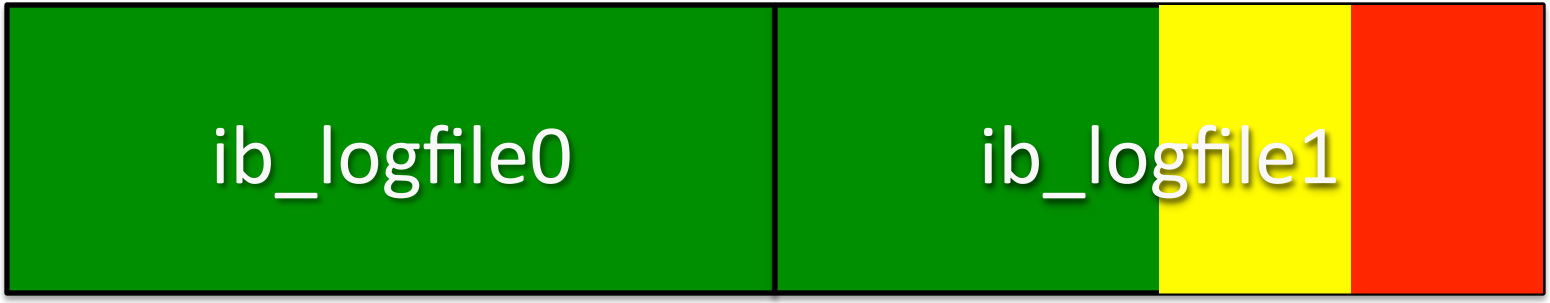
- Dirty pages in the BP must be accounted for by log entries.
- Log entries may not be overwritten until the corresponding dirty pages are flushed.
- Thus a larger log file allows more dirty pages.

# InnoDB Log Size

At 0-75% log file usage, query threads work freely, while page cleaner thread does adaptive flushing continually.

75-88% log file usage, page cleaner runs *async flush.*

88%+ log file usage, *all* query threads block for *sync flush*.



https://blogs.oracle.com/mysqlinnodb/entry/introducing_page_cleaner_thread_in

# Log File Size Indicator

- Enable InnoDB metrics for the buffer pool:

```
SET GLOBAL innodb_monitor_enable='module_buffer';
```

- Monitor for the number of sync waits:

```
SELECT name, count_reset
FROM INFORMATION_SCHEMA.INNODB_METRICS
WHERE name LIKE 'buffer_flush_sync%';
```

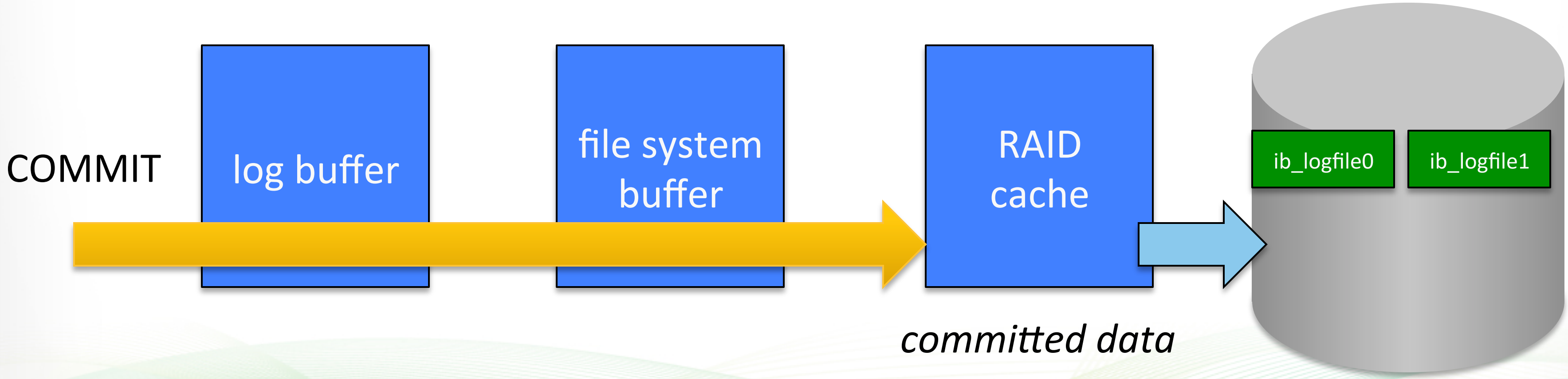
- If the counts are regularly greater than zero, increase `innodb_log_file_size`.

# Log Buffer Size

- Small (8MB) buffer for writing redo log records to the log file.
- If it's full, a COMMIT has to wait for it to flush.
  - `SHOW GLOBAL STATUS LIKE 'Innodb_log_waits';`
  - If you get frequent waits (> 1/minute), increase `innodb_log_buffer_size`.

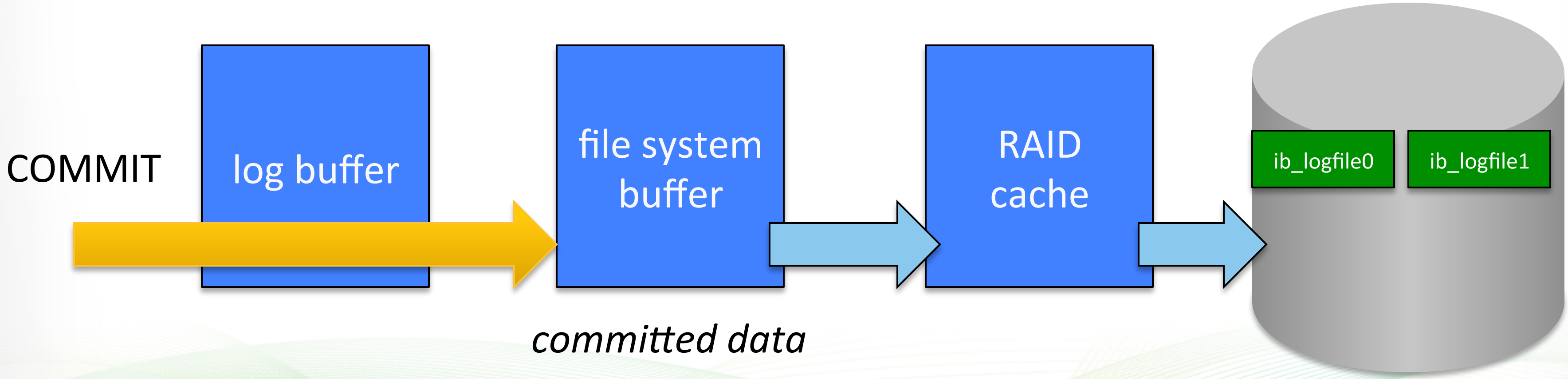
# InnoDB Log Flush at Transaction Commit

- innodb_flush_log_at_trx_commit = 1
  - Every transaction COMMIT is fully synchronous



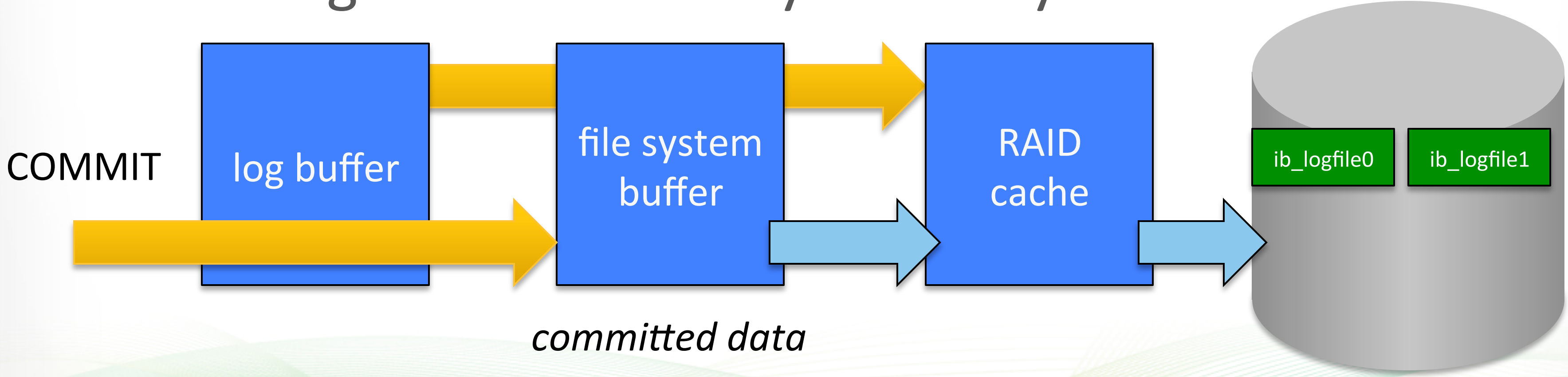
# InnoDB Log Flush at Transaction Commit

- innodb_flush_log_at_trx_commit = 2
  - Each COMMIT flushes to filesystem



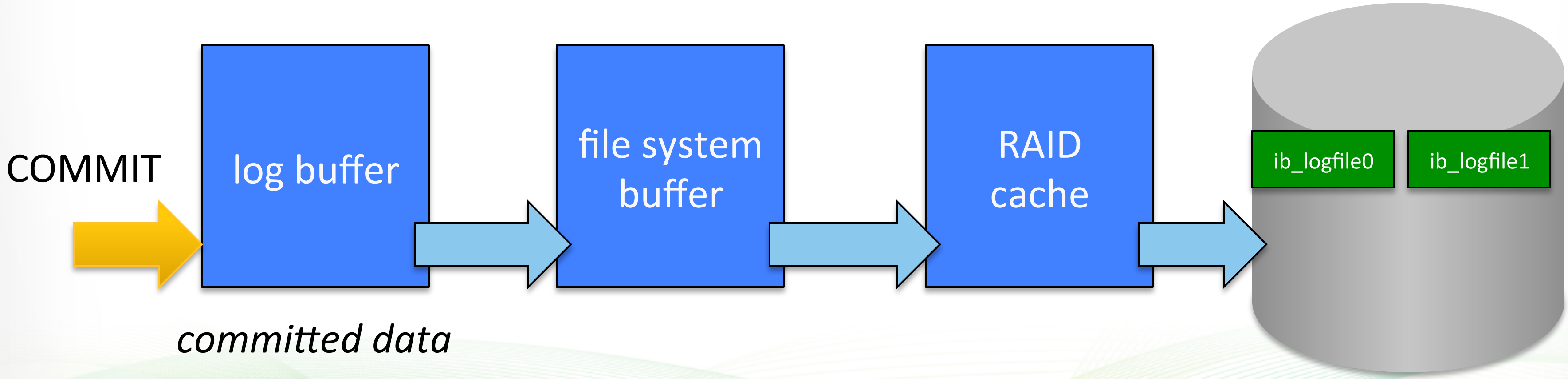
# InnoDB Log Flush at Transaction Commit

- innodb_flush_log_at_trx_commit = 2
  - Each COMMIT flushes to filesystem
  - Background thread fsyncs every 1 second



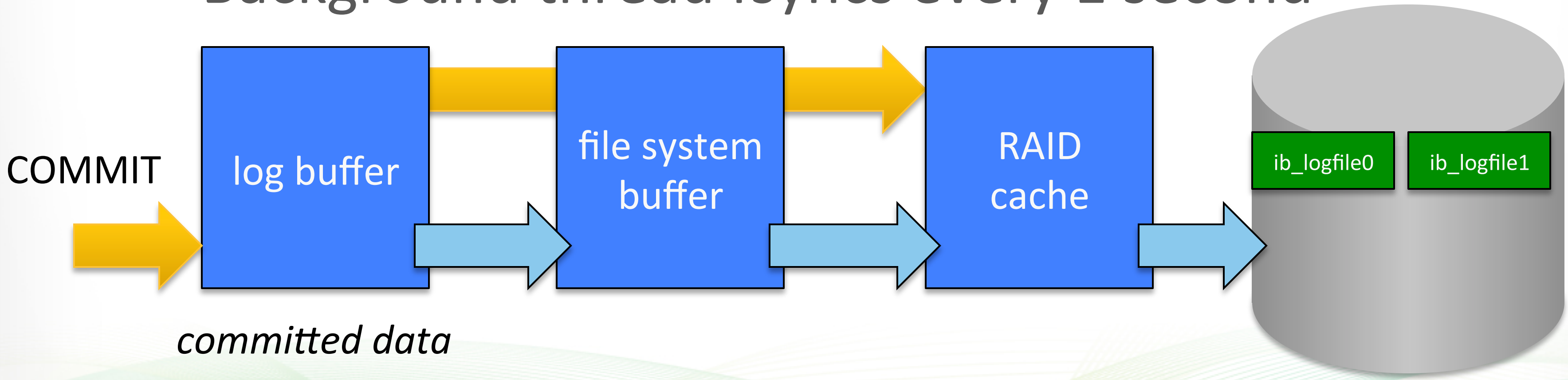
# InnoDB Log Flush at Transaction Commit

- innodb_flush_log_at_trx_commit = 0
  - COMMIT does not flush, only writes to log buffer



# InnoDB Log Flush at Transaction Commit

- innodb_flush_log_at_trx_commit = 0
  - COMMIT does not flush, only writes to log buffer
  - Background thread fsyncs every 1 second



# InnoDB Log Flush at Transaction Commit

- Tradeoff between durability and performance:
  - Sync on commit (=1) limits commits per second.
  - Flush on commit (=2) risks data loss if OS crashes.
  - No flushing (=0) risks data loss if mysqld aborts.

MySQL Tuning

# INNODB IO CAPACITY

# InnoDB IO Capacity

- Limits the IOPS InnoDB uses while:
  - Flushing dirty pages from the buffer pool to the tablespace.
  - Merging change buffer entries to secondary indexes.

# InnoDB IO Capacity Tradeoffs

- Raising IO Capacity
  - Causes flushing to become more aggressive.
  - Uses more IO load.
  - **Good when your write load is constantly high.**

# InnoDB IO Capacity Tradeoffs

- Lowering IO Capacity:
  - Causes flushing to become more gradual.
  - Spreads out the IO load.
  - Allows multiple writes to the same page to be merged into fewer flushes.
  - **Good when your write load has ups and downs.**

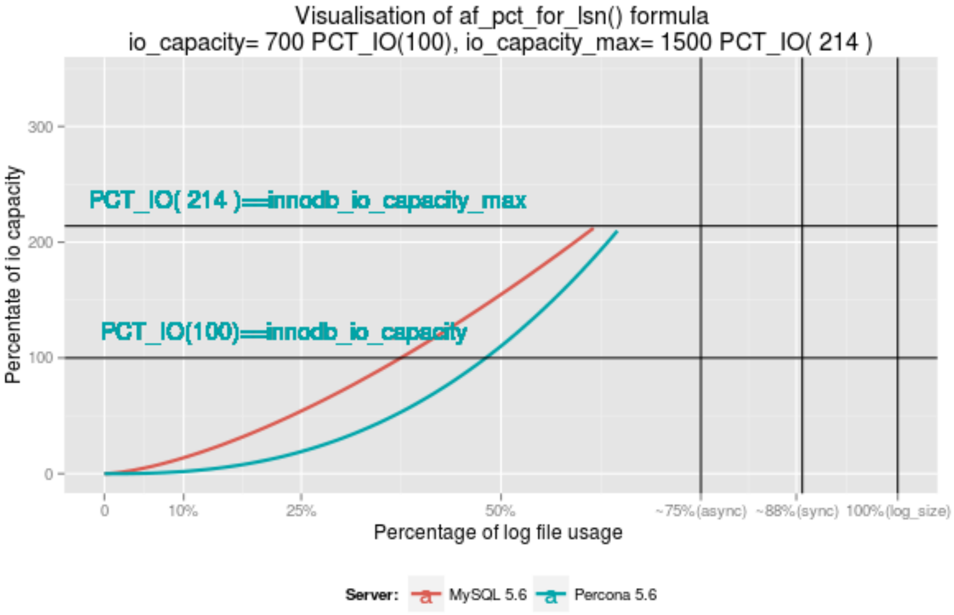
# InnoDB IO Capacity

- innodb_io_capacity = 200 /* default */
  - Limit on rate of flushing pages during idle time, or during shutdown.
  - Change buffer merges at a rate of 5-55% of innodb_io_capacity.

# InnoDB IO Capacity

- innodb_io_capacity_max = 2000 /* default */
  - Limit on rate of flushing during busy time.

# Adaptive Flushing



http://www.percona.com/blog/2013/10/30/innodb-adaptive-flushing-in-mysql-5-6-checkpoint-age-and-io-capacity/

# InnoDB IO Capacity Caveats

- Internals of flushing dirty pages change in each major version of MySQL (5.1/5.5/5.6/5.7/…).
- Different goals? Minimizing IO vs. minimizing log checkpoint age.
- Don’t go so high that you cause IO queuing!

http://www.mysqlplus.net/2013/01/07/play-innodb_io_capacity/

# InnoDB Buffer Pool LRU Scan Depth

- When the BP is full, reading new pages must evict pages currently in the BP.
- Which pages? The least recently used (LRU).

# InnoDB Buffer Pool LRU Scan Depth

- How many pages may be evicted per second?

```
innodb_lru_scan_depth = 1024 /* default */
```

- Increase this if you have spare IO capacity.
- This setting is per BP instance
  - Unlike innodb_io_capacity*, which are for total capacity of flushing for all BP instances.

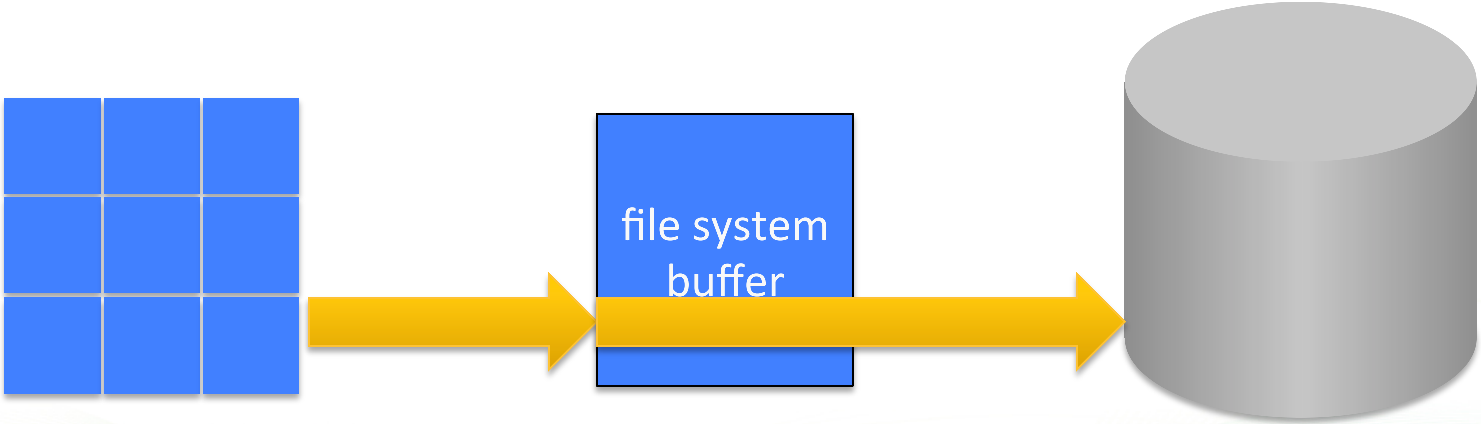
http://mysqlha.blogspot.com/2013/05/configuring-innodb-for-mysql-56.html

MySQL Tuning

# INNODB OTHER CONFIGURATION

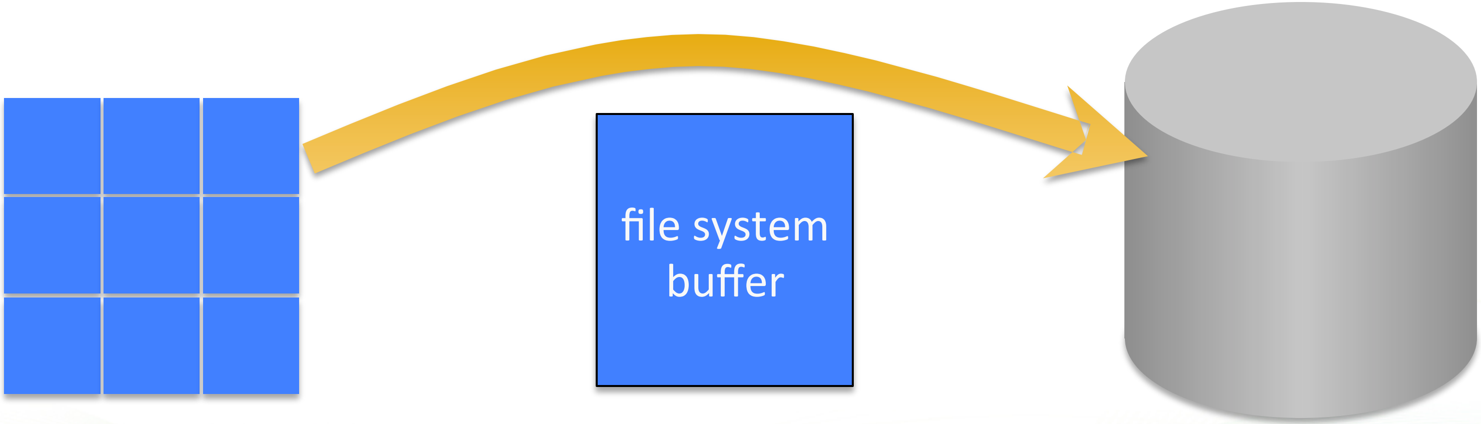
# InnoDB Flush Method

- innodb_flush_method=fdatasync /* default */
- Page flush writes to filesystem, and fsyncs



# InnoDB Flush Method

- innodb_flush_method=O_DIRECT
- Page flush bypasses the filesystem



# InnoDB Flush Method

- The best setting depends on your hardware
  - O_DIRECT is often good on caching RAID
  - O_DIRECT not good when IO has latency (e.g. SAN, DRBD, or Amazon EBS when not EBS-optimized)
- To get optimal results, benchmark *your* application workload on *your* hardware

# InnoDB Change Buffer

- If you INSERT/UPDATE/DELETE in a non-unique index, the *change buffer* helps to delay changes to the index.
  - Makes writes faster
  - The more indexes, the greater the benefit

# InnoDB Change Buffer

- The queue of changes can grow up to 25% of the size of the buffer pool.

```
innodb_change_buffer_max_size=25 /* default */
```

  - In practice, this fills up if you have a *lot* of writes for a sustained period.

# InnoDB Change Buffer

- Background thread merges buffered changes into indexes, at a rate of 5% IO capacity.*
  - Increases gradually up to 55% of IO capacity if the change buffer is more than half of `innodb_change_buffer_max_size`, as a percentage of the BP size.

* It's supposed to merge at a rate of 100% IO capacity when the system is idle, but merging itself counts as "not idle" so I'm not sure it can ever do that.

# InnoDB Change Buffer

To shrink the change buffer, try:

- Increase the merge rate by raising innodb_io_capacity, or lowering innodb_change_buffer_max_size

# InnoDB Change Buffer

- …or reduce the growth of the change buffer:
  - innodb_change_buffering = { inserts | updates | deletes | changes | purges | none }
  - Global option only, not per session or per table.
- …or change write-heavy tables to a different storage engine.

# InnoDB Adaptive Hash Index

- Cache of frequently-requested index values.
- Speeds up secondary index searches automatically – nothing to enable.
- See it working in InnoDB status:

```
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
60608.42 hash searches/s, 86753.09 non-hash searches/s
```

~41% of searches use the AHI

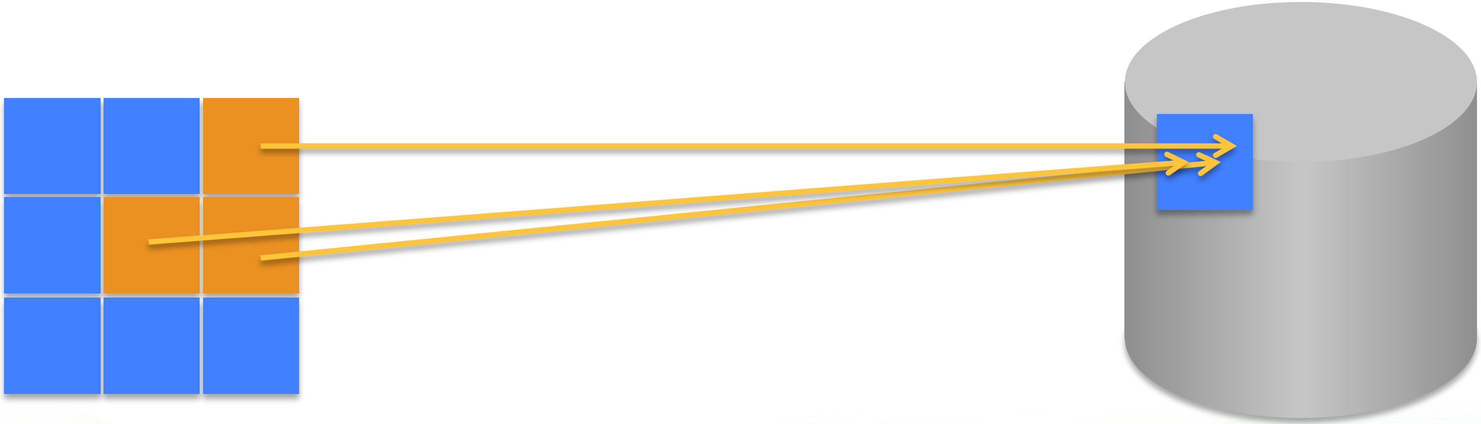
# InnoDB Adaptive Hash Index

*But –*

- The mutex for the AHI can become a bottleneck.
  - SEMAPHORES section of InnoDB status reports waits in btr0sea.c
- You can disable it:
  - skip_innodb_adaptive_hash_index
- Percona Server can split it:
  - innodb_adaptive_hash_index_partitions = <N>

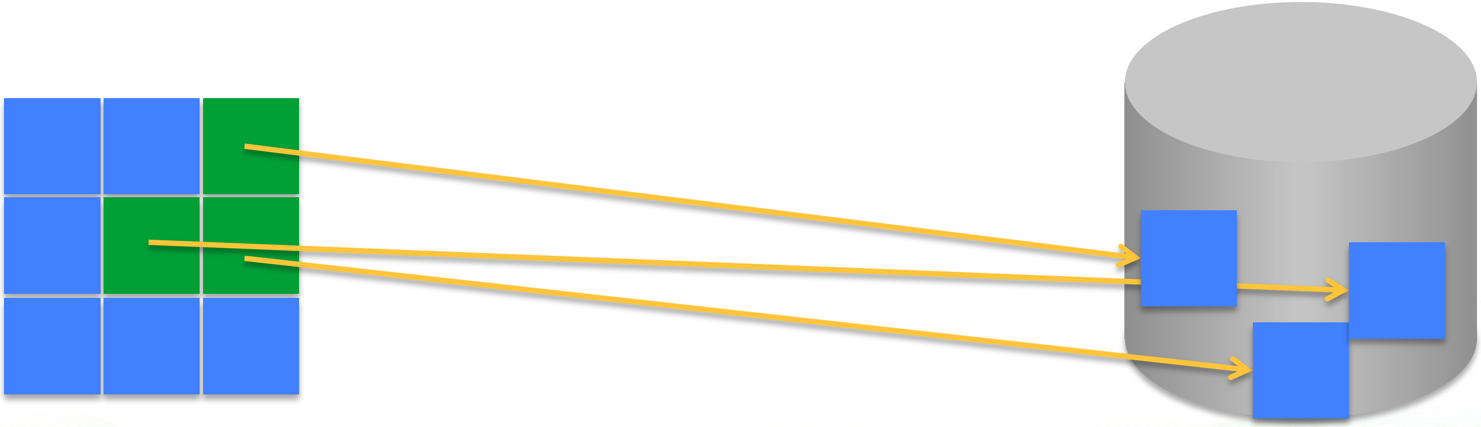
# InnoDB Doublewrite Buffer

- Every dirty page flush writes twice:
  - First, write to the doublewrite buffer on disk



# InnoDB Doublewrite Buffer

- Every dirty page flush writes twice:
  - Second, write to the respective pages on disk



# InnoDB Doublewrite Buffer

- The doublewrite buffer adds overhead.
- Alternative:
  - Put datadir on a transactional filesystem.
  - Disable doublewrite buffer:

    skip_innodb_doublewrite

http://www.percona.com/blog/2014/05/23/improve-innodb-performance-write-bound-loads/

# InnoDB Read/Write Threads

- Background threads to read and write pages.
- See them working in InnoDB Status:

```
--------
FILE I/O
--------
I/O thread 2 state: waiting for completed aio requests (read thread)
I/O thread 3 state: waiting for completed aio requests (read thread)
I/O thread 4 state: waiting for completed aio requests (read thread)
I/O thread 5 state: waiting for completed aio requests (read thread)
I/O thread 6 state: waiting for completed aio requests (write thread)
I/O thread 7 state: waiting for completed aio requests (write thread)
I/O thread 8 state: waiting for completed aio requests (write thread)
I/O thread 9 state: waiting for completed aio requests (write thread)
Pending normal aio reads: 0 [0, 0, 0, 0] , aio writes: 0 [0, 0, 0, 0] ,
```

*Watch the number of pending reads and writes.*

*If these go too high (~64), then increase number of IO threads.*

MySQL Tuning

# OPTIMIZER CONFIGURATION

# Optimizer Switches

- Enable/disable optimizer features that aren't doing what you want. Example:
  - optimizer_switch='index_merge_intersection=off';

http://www.percona.com/blog/2012/12/14/the-optimization-that-often-isnt-index-merge-intersection/

# Sort Buffer Size

- In-memory buffer per thread for sorting query results.
  - `SET sort_buffer_size = 256K /* default */`
- If the result is too large, subsets are sorted and merged on disk.
  - `SHOW GLOBAL STATUS LIKE 'Sort_merge_passes';`

```
+-------------------+---------+
| Variable_name     | Value   |
+-------------------+---------+
| Sort_merge_passes | 6060842 |
+-------------------+---------+
```

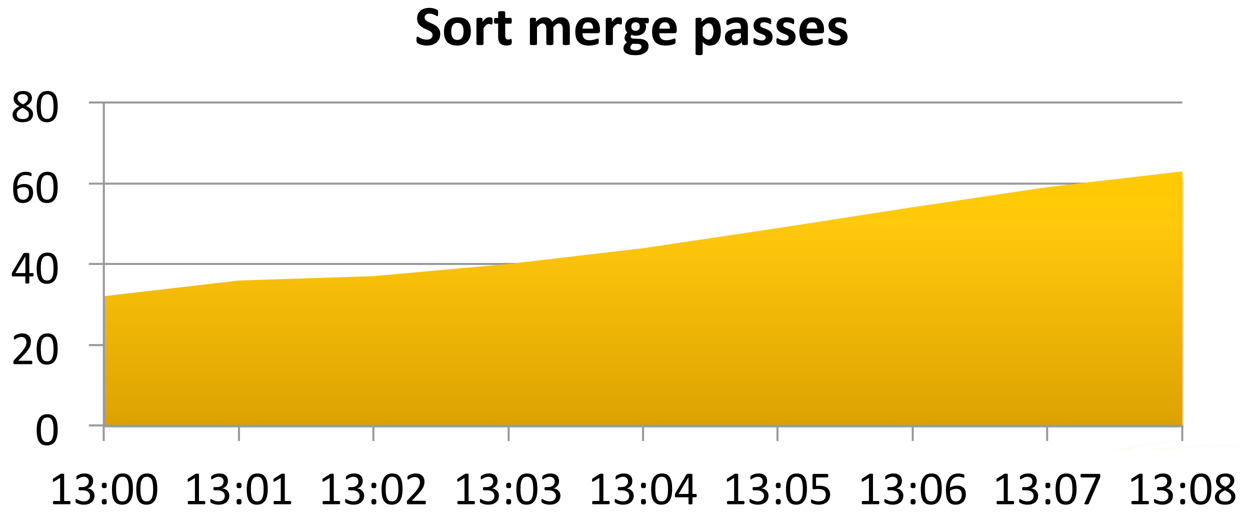
*“Measure twice, cut once.”*

# Step 1

- Choose a measurable indicator of performance
  - e.g. `sort_buffer_size` effectiveness is measured by `sort_merge_passes`
- Measure the impact of performance before changing the configuration parameter.

# Step 2

- Measure the rate of increase:



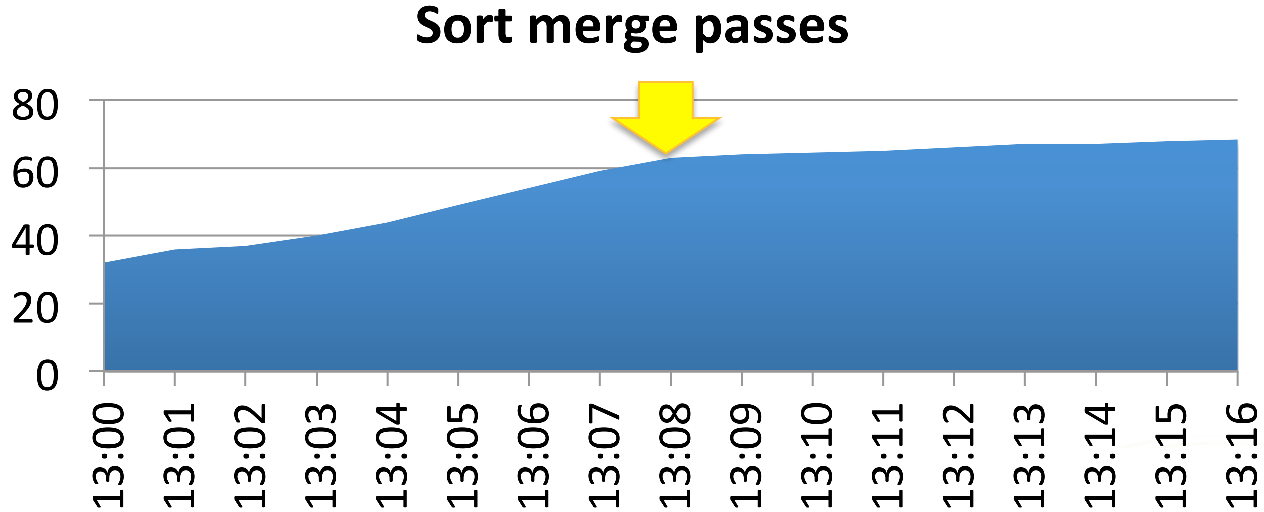
# Step 3

- Research the range of reasonable values for the corresponding configuration variable.
- Make a *modest* change.
  - For example, this variable was 256KB by default. Let’s raise it to 384KB.

```
mysql> SET GLOBAL sort_buffer_size = 384*1024;
```

# Step 4

- Re-measure the rate after the change:



# How to Decide on a Size?

- Must the rate of increase be zero? **No.**
- Using the disk for sort merge passes *occasionally* is normal.
  - Size the sort buffer so the rate of sort merge passes is low – but no need to make it zero.
- This principle applies to other tuning as well.

- Design a graph to show the bell curve of result sizes, and how the first modest increase handles more cases than the subsequent incremental increases.

# Join Buffer Size

- Used by Batched Key Access in 5.6

```
SET optimizer_switch =
'mrr=on,mrr_cost_based=off';
SET optimizer_switch = 'batched_key_access=on';
SET join_buffer_size = 256K; /* default */
```

https://dev.mysql.com/doc/refman/5.6/en/bnl-bka-optimization.html

# Read Rnd Buffer Size

- Used by Multi-Range Reads in 5.6:

```
SET optimizer_switch =
'mrr=on,mrr_cost_based=off';
SET read_rnd_buffer_size = 256K; /* default */
```

https://dev.mysql.com/doc/refman/5.6/en/mrr-optimization.html

# Index Dives

- Searching for multiple disjoint values in a query
  - `WHERE foo IN (1, 2, 3, … N)`
  - `WHERE foo=1 OR foo=2 … OR foo=N`
  - MySQL 5.5 searches the index for each value individually, so very long lists result in many index “dives” – and slow queries.
- MySQL 5.6 uses index statistics instead, when the list grows longer than a configurable limit.
  - `SET eq_range_index_dive_limit = 200;`
  - Default in MySQL 5.6 is 10. In MySQL 5.7, it’s 200.

MySQL Tuning

# LOGGING CONFIGURATION

# Slow Query Log

- Write information about every “slow” query.
  - SET GLOBAL slow_query_log = ON;
- Choose FILE or TABLE output.
  - SET GLOBAL log_output = 'FILE';
  - TABLE is much slower.

# Slow Query Log Filtering

- Limit logging by time.
  - SET GLOBAL long_query_time = 10;

# Slow Query Log Filtering

- (Percona Server) Limit logging by query execution plan.
  - SET GLOBAL log_slow_filter = 'tmp_table_on_disk,filesort_on_disk';

# Slow Query Log Filtering

- (Percona Server) Limit logging by sampling, e.g. 1/100 queries or sessions.
  - SET GLOBAL log_slow_rate_limit = 100;

MySQL Tuning

# REPLICATION CONFIGURATION

# Sync Binlog

- Choose how strictly durable the binlog is – trading off performance.
- Sync writes to the binlog every *N* commits.
  - SET sync_binlog = 0; /* default */
- Many people assume this is boolean and set it to 1, which causes the highest overhead.

# Binlog Format

- binlog_format is a tradeoff between deterministic changes vs. efficiency.
  - STATEMENT writes less in the log, but requires parsing, optimization and execution on slave.
  - ROW writes more log for complex updates, but has less overhead and more reliability.

# Binlog Cache

- A buffer for uncommitted binlog writes.

```
SET GLOBAL binlog_cache_size = 32768; /* default */
```

- A transaction must save buffer to disk when the buffer is full.

# Binlog Cache

- Monitor ratio of disk use.
  - SHOW GLOBAL VARIABLES LIKE 'Binlog_cache_%use';

```
+-----------------------+---------+
| Variable_name         | Value   |
+-----------------------+---------+
| Binlog_cache_disk_use |     327 |
| Binlog_cache_use      | 6060842 |
+-----------------------+---------+
```

# Multi-Threaded Slave

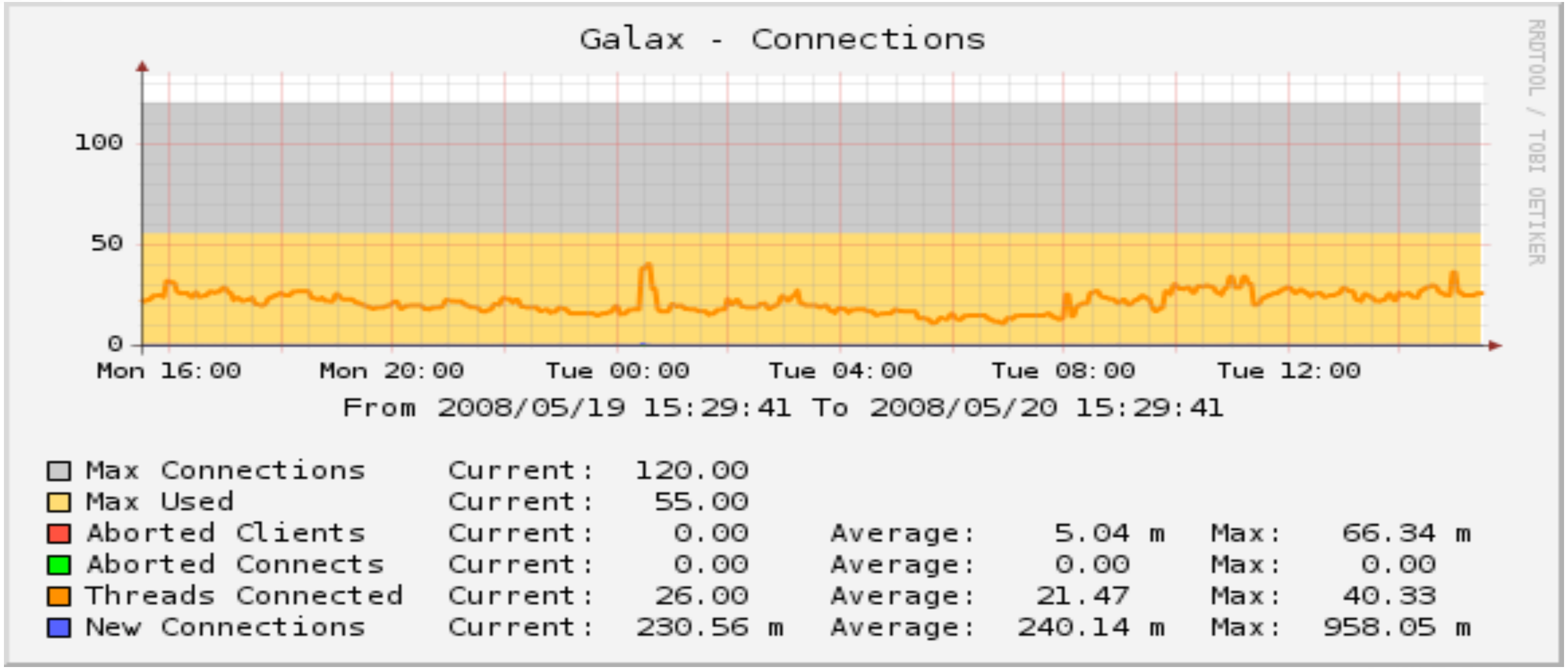
- Intended to replay binlog events as fast as the master created them.
  - SET GLOBAL slave_parallel_workers = 10;
  - Updates to a given schema still happen serially in one thread – use as many workers as schemas.*

https://blogs.oracle.com/MySQL/entry/benchmarking_mysql_replication_with_multi

MySQL Tuning

# CONNECTION AND THREAD CONFIGURATION

# Max Connections



```
From 2008/05/19 15:29:41 To 2008/05/20 15:29:41

⬜ Max Connections      Current:   120.00
🟨 Max Used             Current:    55.00
🟥 Aborted Clients      Current:     0.00     Average:     5.04 m    Max:    66.34 m
🟩 Aborted Connects     Current:     0.00     Average:     0.00      Max:     0.00
🟧 Threads Connected    Current:    26.00     Average:    21.47      Max:    40.33
🟦 New Connections      Current:   230.56 m   Average:   240.14 m    Max:   958.05 m
```

# Max Connections

- Should you set max_connections very high?
  - Have you tested what happens when your server spikes to 10,000+ threads?
  - Resource contention could make performance exponentially worse.
  - It could be better to refuse some connections, so those that are connected can finish.

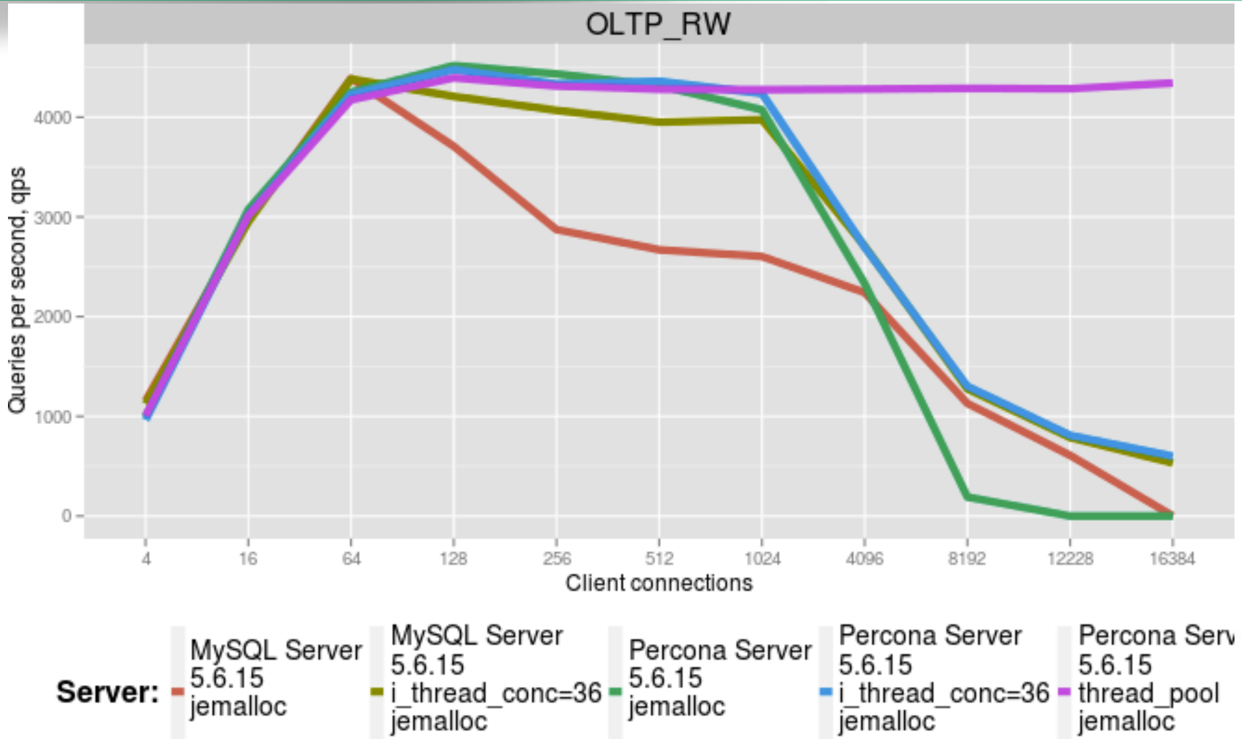
# InnoDB Thread Concurrency

- How many threads are allowed to work in the InnoDB engine in parallel?
  - SET GLOBAL innodb_thread_concurrency = 0;
    /* default, no limit */
  - If too much contention, try 2× CPU cores.

# Thread Pool

- (Percona Server) Serve more connections with a limited number of MySQL user threads.
  - `thread_handling=pool-of-threads`
  - `thread_pool_size=36`
  - `thread_concurrency=0`
- Test with the thread pool if you typically see `Threads_running` at 64 or more.

# Thread Pool



http://www.percona.com/blog/2014/01/23/percona-server-improve-scalability-percona-thread-pool/

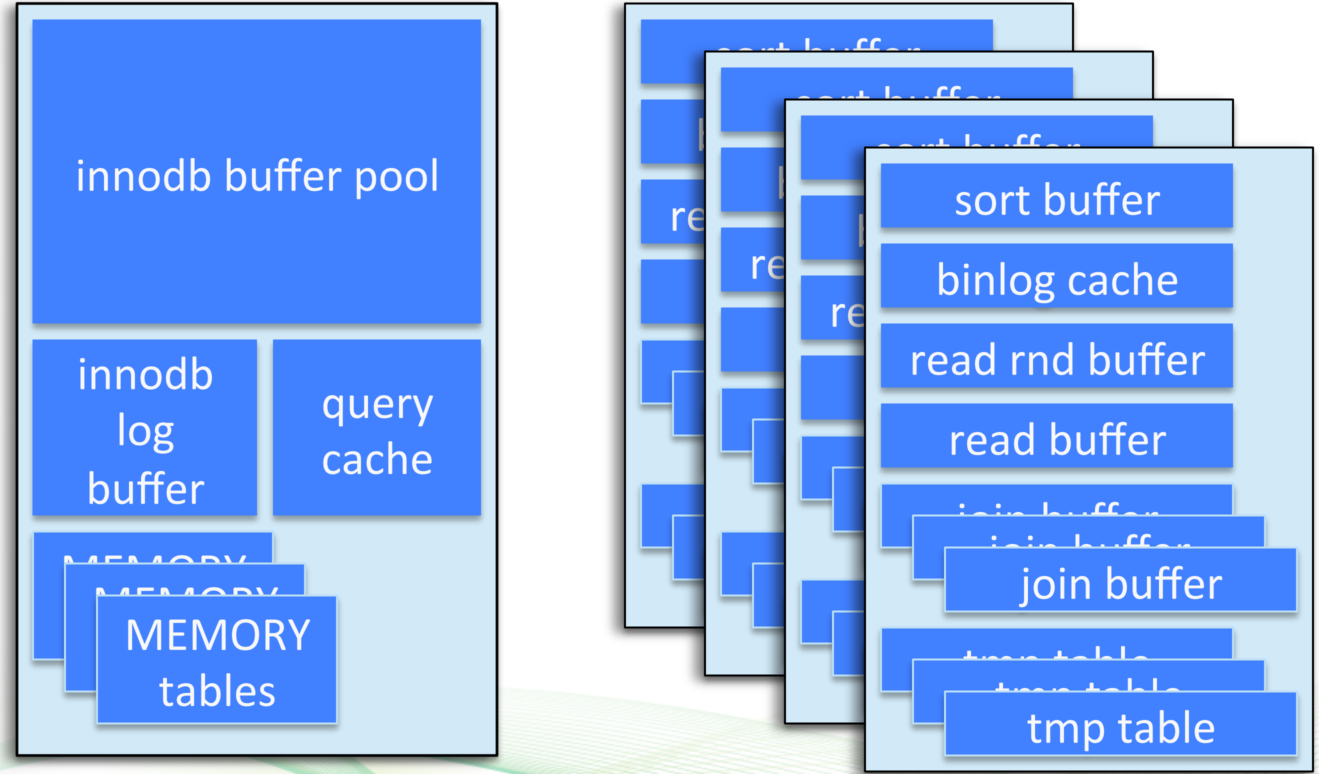
# Don’t Overallocate

- Some buffers are allocated globally:
  - innodb_buffer_pool_size
  - innodb_log_buffer_size
  - max_heap_table_size*
  - query_cache_size

- Some are allocated per SQL thread:
  - sort_buffer_size
  - binlog_cache_size
  - join_buffer_size*
  - read_buffer_size
  - read_rnd_buffer_size
  - thread_stack
  - tmp_table_size*

* may be allocated multiple times

# Global vs. Per-Thread Resources



innodb buffer pool

innodb log buffer

query cache

MEMORY tables

sort buffer

binlog cache

read rnd buffer

read buffer

join buffer

tmp table

× Max_connections

or more practically, Max_used_connections

MySQL Tuning

# TABLES CONFIGURATION

# Table Open Cache

- Cache for table metadata. Limits the number of tables in use for all threads.
  - SET GLOBAL table_open_cache = 2000; /* default */
  - Set to Threads_running × tables per query.

# Table Open Cache

- Watch the number of opened tables per second.
  - SHOW GLOBAL STATUS LIKE 'Opened_tables';
  - If the rate of increase is too sharp, increase the table_open_cache.

# Table Open Cache Instances

- MySQL 5.6 feature to split the table cache.
  - table_open_cache_instances = 1; /*default*/
  - May help reduce contention when you have `Threads_running` more than ~64.
  - Set to 8 or 16, the number of CPU cores.

# Table Definition Cache

- Cache for table metadata (contents of .frm).
  - Small overhead, so safe to increase this.
  - SET GLOBAL table_definition_cache = 400 + (@@table_open_cache/2); /* default */
  - InnoDB also uses this as a soft limit for the data dictionary cache.

# InnoDB Open Tables

- Limits the number of open InnoDB tablespaces.
  - innodb_open_tables = -1 /* default, autosized */
  - Self-adjusts up to table_open_cache.

# InnoDB File Per Table

- Store each InnoDB data in a separate .ibd file (tablespace). Default in 5.6+.
  - Required for some table options, transportable tablespaces, recovering disk space.
  - No significant performance impact, unless you have tens of thousands of tables.

MySQL Tuning

# QUERY CACHE CONFIGURATION

# Query Cache Effectiveness: Number

- Monitor ratio of QC hits vs. misses:

```
SHOW GLOBAL STATUS LIKE 'QCache%';
+-------------------------+----------+
| Variable_name           | Value    |
+-------------------------+----------+
| Qcache_hits             |  8675309 |
| Qcache_inserts          |   606084 |
```

*15:1 hits – good!*

# Query Cache Effectiveness: Magnitude

- `pt-query-digest` outputs response time histogram per query.

```
# QC Hit         63% yes,  36% no
. . .
# Query_time distribution
#   1us
#  10us  ################################################################
# 100us  ################################
#   1ms
#  10ms
# 100ms  ###
#    1s  #################################
#  10s+  #####################
```

*query cache hits*

*query cache misses – 4-5 orders of magnitude slower (for this query)*

# Query Cache Scalability

- Every thread that reads or writes the query cache must acquire a mutex.
  - SELECT must check if the query result is cached
  - INSERT/UPDATE/DELETE must evict query results
- This can become a bottleneck when you have many threads running – worse than not having the query cache!

# Query Cache Tuning

- Better to disable it unless you can confirm it's giving you a lot of benefit.
  - query_cache_type = 0
  - query_cache_size = 0

*you should set* ***both*** *in my.cnf!*

# Query Cache Defaults

- MySQL 5.5:
  - query_cache_type = 1 /* still enables mutex */
  - query_cache_size = 0 /* allocates no memory */
- MySQL 5.6.8+:
  - query_cache_type = 0 /* off */
  - query_cache_size = 1M /* still allocates memory */

MySQL Tuning

# TEMPORARY SPACE CONFIGURATION

# Temporary Table Size

- A temporary table created by a query may fit in memory up to tmp_table_size
  - SET tmp_table_size = 16M; /* default */
  - Larger temp tables are written to disk.
- Exception: temp tables with BLOB/TEXT columns always write to disk.

# Temporary Table Size

- Any in-memory table is limited by:
  - SET max_heap_table_size = 16M; /* default */
  - So it’s futile to set tmp_table_size greater.

# Temporary Table Size

- Monitor ratio of temp tables to temp tables on disk:

```
SHOW GLOBAL STATUS LIKE 'Created%tables';
+-------------------------+---------+
| Variable_name           | Value   |
+-------------------------+---------+
| Created_tmp_disk_tables | 123     |
| Created_tmp_tables      | 6060842 |
+-------------------------+---------+
```

# Temporary Table Size

- Like other buffers, increase tmp_table_size to handle more cases, until the rate of disk tables drops.

# Temporary Tables per Statement

- (Percona Server) Verbose slow-query log includes fields for Tmp_tables, Tmp_disk_tables, Tmp_table_sizes.

- Performance_Schema.events_statements_% includes columns created_tmp_tables and created_tmp_disk_tables.
  - But not tmp table sizes (http://bugs.mysql.com/74484).

# Tmpfs Partition

- Trick MySQL into writing “on disk” temp tables into memory (even with BLOB/TEXT):
  - In /etc/my.cnf:

    ```
    tmpdir=/var/lib/mysqltmp
    ```

  - In /etc/fstab:

    ```
    tmpfs /var/lib/mysqltmp tmpfs \
    noatime,size=1G,mode=700,uid=mysql,gid=mysql    0   0
    ```

MySQL Tuning

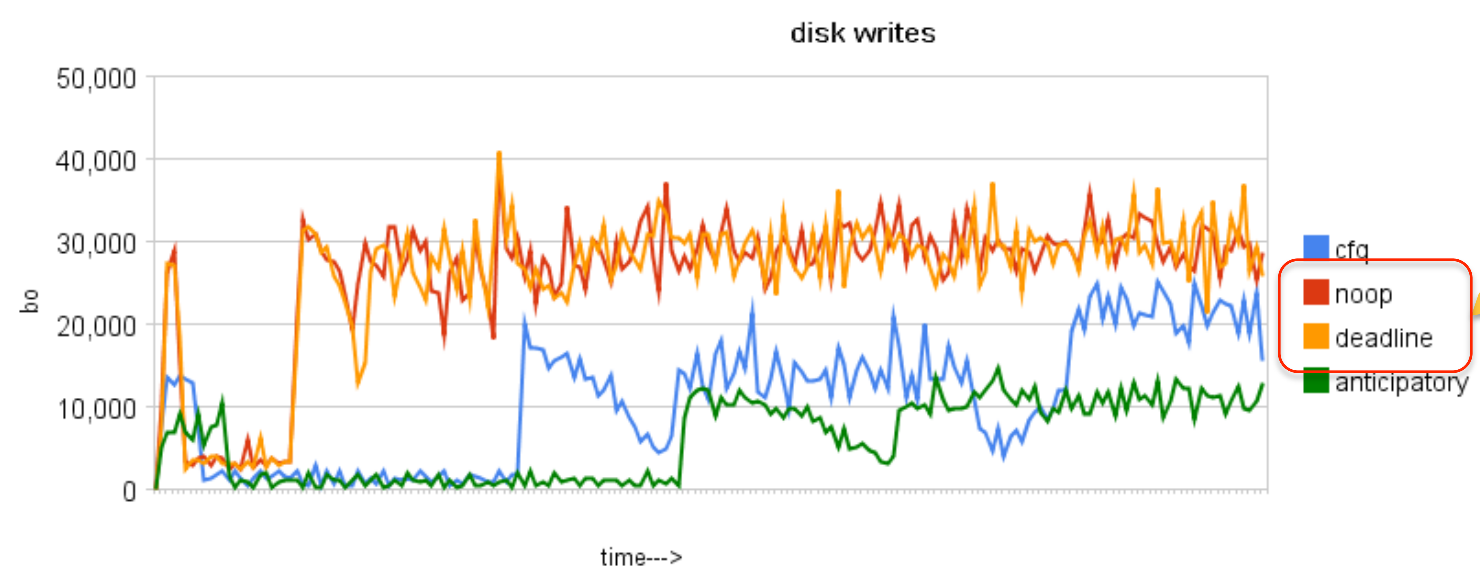
# OPERATING SYSTEM CONFIGURATION

# Filesystem

- XFS consistently performs better in Percona benchmarks.
  - Especially for multi-threaded IO while using innodb_flush_method=O_DIRECT

http://www.percona.com/blog/2011/12/16/setting-up-xfs-the-simple-edition/

# IO Scheduler



*either one of these is better*

http://www.percona.com/blog/2009/01/30/linux-schedulers-in-tpcc-like-benchmark/

# Mount Options

- Reduce unneeded writes to update access times (but less important to tune if you use XFS)
  - noatime
  - nodiratime
- If you use ext4, and battery-backed RAID cache, disable barriers
  - nobarrier

# Swappiness

- Reduce the kernel's tendency to swap proactively. Edit /etc/sysctl.conf:
  - `vm.swappiness = 1`
  - Some old advice was to set swappiness to 0, but in Linux kernel 2.6.32+, the value 0 can lead to OOM conditions that kill mysqld.
    http://www.percona.com/blog/2014/04/28/oom-relation-vm-swappiness0-new-kernel/

# NUMA

- Some random stalls have been fixed by:
  - `numactl --interleave=all`
  - `sysctl -q -w vm.drop_caches=3`
  - forcing the NUMA node allocation decisions to be made on startup
- These changes were developed at Twitter
  - http://blog.jcole.us/2012/04/16/a-brief-update-on-numa-and-mysql/
- These patches are available in Percona Server 5.5 – 5.6
  - http://www.percona.com/doc/percona-server/5.6/performance/innodb_numa_support.html

MySQL Tuning

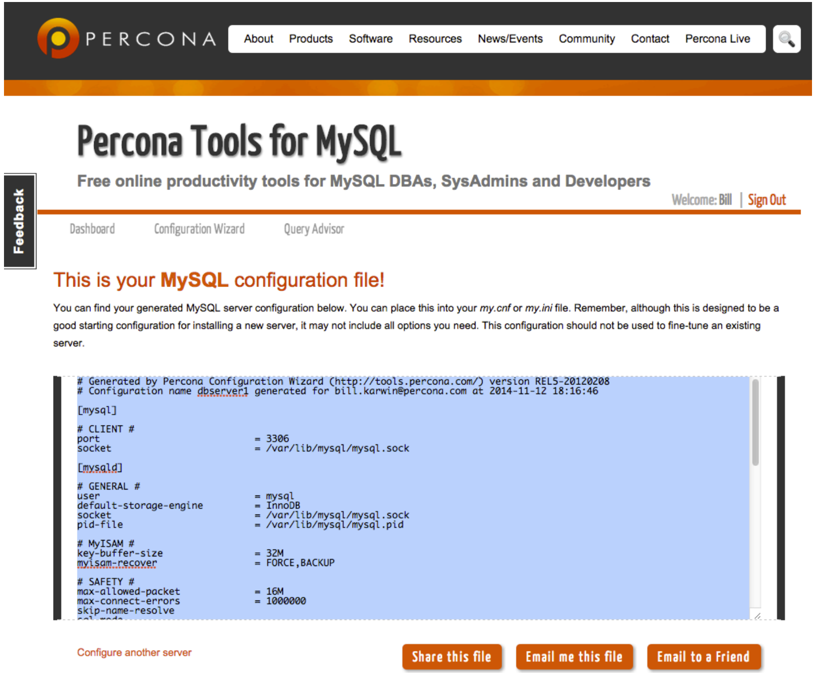
# TUNING TOOLS

# Mysqltuner

- Recommends rough tuning changes based on activity rates in SHOW GLOBAL STATUS.
  - Somewhat outdated – claims to support MySQL 5.4 and 6.0, which never existed.
  - Averages over uptime fail to account for spikes.

http://mysqltuner.com/

# Percona Configuration Wizard



## Percona Tools for MySQL

**Free online productivity tools for MySQL DBAs, SysAdmins and Developers**

### This is your **MySQL** configuration file!

You can find your generated MySQL server configuration below. You can place this into your *my.cnf* or *my.ini* file. Remember, although this is designed to be a good starting configuration for installing a new server, it may not include all options you need. This configuration should not be used to fine-tune an existing server.

```
# Generated by Percona Configuration Wizard (http://tools.percona.com/) version REL5-20120208
# Configuration name dbserver1 generated for bill.karwin@percona.com at 2014-11-12 18:16:46

[mysql]

# CLIENT #
port                           = 3306
socket                         = /var/lib/mysql/mysql.sock

[mysqld]

# GENERAL #
user                           = mysql
default-storage-engine         = InnoDB
socket                         = /var/lib/mysql/mysql.sock
pid-file                       = /var/lib/mysql/mysql.pid

# MyISAM #
key-buffer-size                = 32M
myisam-recover                 = FORCE,BACKUP

# SAFETY #
max-allowed-packet             = 16M
max-connect-errors             = 1000000
skip-name-resolve
```

*Usually good as a starting point, but not a substitute for monitoring and incremental tuning.*

https://tools.percona.com/

MySQL Tuning

# MONITORING TOOLS

# pt-mext

- Console tool to measure SHOW GLOBAL STATUS and display incremental changes.
- Nothing to install, part of the Percona Toolkit scripts.

http://www.percona.com/doc/percona-toolkit/pt-mext.html

# Innotop

- Real-time monitoring tool for queries, transactions, memory, IO, etc.

http://www.percona.com/blog/2013/10/14/innotop-real-time-advanced-investigation-tool-mysql/

# Percona Monitoring Plugins

- Templates for Cacti and Zabbix to graph GLOBAL STATUS values.

http://www.percona.com/software/percona-monitoring-plugins

# Percona Cloud Tools

- New Beta SaaS offering from Percona.
- Graph many performance metrics like PMP.
- Query analysis and trending.

https://cloud.percona.com/

MySQL Tuning

# TUNING VERSUS ARCHITECTURE

# Tuning Advantages

- No changes required to schema.
- No changes required to code.
- Some tuning changes possible without restarting mysqld.

# Tuning Disadvantages

- Most changes global or per session – but cannot be set per user, database, or query.
- Can improve performance – but only so far.

# What's the Next Step?

- Optimizing by greater order of magnitude:
  - Caching
  - Denormalizing
  - Indexing
  - Sharding or partitioning
  - Query design

# Caching

- Retain often-used query results.
- Store in memory, for faster retrieval.
- Performance strategy: avoid repeat queries.

# Denormalizing

- Store data redundantly, for a partially-completed query.
- Avoid expensive expressions, group summaries, or joins.
- Performance strategy: leverage early work.

# Indexing

- Store a pre-sorted copy of some column(s).
- Searching and sorting is now orders of magnitude faster.
- Performance strategy: data structure.

# Partitioning

- One logical table maps to multiple physical tables of similar type.
- Queries are designed to limit to one partition.
- Performance strategy: divide and conquer.

# Sharding

- Split a table into subsets of rows.
- Host each subset on a separate instance.
- Performance strategy: horizontal scaling with instances.

# Query Design

- “There’s more than one way to do it.”
- Experiment with alternative query designs for the same result.
- Performance strategy: take advantage of SQL implementation idiosyncrasies.

MySQL Tuning

# CONCLUSIONS

# Recap

- Measure relevant performance indicators before and after tuning changes.
- Re-test when you upgrade.
- Monitor continually.
- Architecture changes can improve performance beyond tuning.

# Thank you!

# Questions?