Aurimas Mikalauskas
@ Speedemy

MYSQL PERFORMANCE TUNING

I. MySQL Configuration (incl. MySQL 5.7)

AURIMAS MIKALAUSKAS

- MySQL hacker since 1999 (yep, v3.23)
- In love with performance & scalability since 2004 Danga Interactive talk on LiveJournal's Backend at LISA04
- Percona Senior Performance Consultant & Architect 2006-2015
- Performance Engineer at EstanteVirtual.com.br
- Independent data performance consultant, instructor at Speedemy.com

ON THE AGENDA FOR TODAY

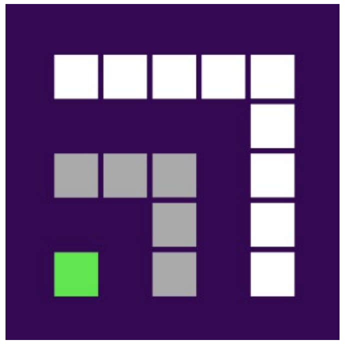
How to Choose The Right MySQL Distribution

Essentials of MySQL Configuration Tuning

17 Key MySQL Settings for Best Performance

DISTROS

MySQL? Percona Server? MariaDB? WebScaleSQL?



COMMUNITY MYSQL

- Community MySQL 5.1
  - innodb plugin not enabled by default
  - very poor InnoDB scalability (by default)
- Community MySQL 5.5
  - still missing a number of performance improvements
  - stability issues with high amounts of writes
- Community MySQL 5.6
  - not many reasons to use alternative distro
  - still not perfectly stable with high end hardware in a write-intensive environment

COMMUNITY MYSQL 5.7

- Current GA release
- Rock solid
- New features:
  - multi-source replication
  - JSON support
  - proper multi-threaded replication
  - online buffer pool resize
  - spatial data types for InnoDB
  - sys schema
  - etc. (see full list)

PERCONA SERVER

- Launched July 2008
  - as Percona patches for MySQL
  - in reality earlier than that
- Always up to date with upstream
  - i.e. Percona Server 5.1.73 = MySQL 5.1.73 + Percona code
  - also always backwards compatible
    - meaning no problem to keep switching between Percona Server and MySQL. No upgrade/downgrade scripts need to be used.
  - some exceptions to backwards compatibility in the past
    - none of these options were enabled by default
    - very clear when it’s the case
- Makes it easy to try it out and switch back at any point

PERCONA SERVER VERSION

- Using MySQL 5.1?
  - definitely switch to Percona Server 5.1
- Using MySQL 5.5? Switch to Percona Server 5.5 if…
  - adaptive hash index is a bottle-neck
  - you want faster checksums (hardware accelerated)
  - you have write-intensive workload
  - you need PAM authentication, audit log, thread pool
- Using MySQL 5.6? Switch to Percona Server 5.6 if…
  - you have very write-intensive workload and high end hardware
  - you need some of the features mentioned with 5.5
- MySQL 5.7? Use MySQL, Percona Server 5.7 coming soon.

PERCONA SERVER KEYSTONE FEATURE: EXTENDED SLOW QUERY LOG

```
# Time: 150811 12:15:06.959000
# User@Host: root[root] @ localhost [localhost]  Id: 62591948
# Schema: Core_DB  Last_errno: 0  Killed: 0
# Query_time: 2.411376  Lock_time: 0.002444  Rows_sent: 0  Rows_examined: 10811959  Rows_affected: 0
# Bytes_sent: 0  Tmp_tables: 1  Tmp_disk_tables: 0  Tmp_table_sizes: 0
# QC_Hit: No  Full_scan: Yes  Full_join: No  Tmp_table: Yes  Tmp_table_on_disk: No
# Filesort: No  Filesort_on_disk: No  Merge_passes: 0
# InnoDB_trx_id: 20A40488F
#   InnoDB_IO_r_ops: 37  InnoDB_IO_r_bytes: 606208  InnoDB_IO_r_wait: 0.000225
#   InnoDB_rec_lock_wait: 0.000000  InnoDB_queue_wait: 0.000000
#   InnoDB_pages_distinct: 1747
SELECT ...
```

microsecond TS

-VS-

```
# Time: 150811 12:15:06
# User@Host: root[root] @ localhost [localhost]  Id: 62591948
# Schema: Core_DB  Last_errno: 1051  Killed: 0
# Query_time: 2.411376  Lock_time: 0.002444  Rows_sent: 0  Rows_examined: 10811959
SELECT ...
```

EXTENDED SLOW QUERY LOG (2)

```
# Time: 150811 12:15:06.959000
# User@Host: root[root] @ localhost [localhost]  Id: 62591948
# Schema: Core_DB  Last_errno: 0  Killed: 0
# Query_time: 2.411376  Lock_time: 0.002444  Rows_sent: 0  Rows_examined: 10811959  Rows_affected: 0
# Bytes_sent: 0  Tmp_tables: 1  Tmp_disk_tables: 0  Tmp_table_sizes: 0
# QC_Hit: No  Full_scan: Yes  Full_join: No  Tmp_table: Yes  Tmp_table_on_disk: No
# Filesort: No  Filesort_on_disk: No  Merge_passes: 0
# InnoDB_trx_id: 20A40488F
#   InnoDB_IO_r_ops: 37  InnoDB_IO_r_bytes: 606208  InnoDB_IO_r_wait: 0.000225
#   InnoDB_rec_lock_wait: 0.000000  InnoDB_queue_wait: 0.000000
#   InnoDB_pages_distinct: 1747
SELECT ...
```

query plan

-VS-

```
# Time: 150811 12:15:06
# User@Host: root[root] @ localhost [localhost]  Id: 62591948
# Schema: Core_DB  Last_errno: 1051  Killed: 0
# Query_time: 2.411376  Lock_time: 0.002444  Rows_sent: 0  Rows_examined: 10811959
SELECT ...
```

EXTENDED SLOW QUERY LOG (3)

```
# Time: 150811 12:15:06.959000
# User@Host: root[root] @ localhost [localhost]  Id: 62591948
# Schema: Core_DB  Last_errno: 0  Killed: 0
# Query_time: 2.411376  Lock_time: 0.002444  Rows_sent: 0  Rows_examined: 10811959  Rows_affected: 0
# Bytes_sent: 0  Tmp_tables: 1  Tmp_disk_tables: 0  Tmp_table_sizes: 0
# QC_Hit: No  Full_scan: Yes  Full_join: No  Tmp_table: Yes  Tmp_table_on_disk: No
# Filesort: No  Filesort_on_disk: No  Merge_passes: 0
# InnoDB_trx_id: 20A40488F
#   InnoDB_IO_r_ops: 37  InnoDB_IO_r_bytes: 606208  InnoDB_IO_r_wait: 0.000225
#   InnoDB_rec_lock_wait: 0.000000  InnoDB_queue_wait: 0.000000
#   InnoDB_pages_distinct: 1747
SELECT ...
```

extra innodb info

-VS-

```
# Time: 150811 12:15:06
# User@Host: root[root] @ localhost [localhost]  Id: 62591948
# Schema: Core_DB  Last_errno: 1051  Killed: 0
# Query_time: 2.411376  Lock_time: 0.002444  Rows_sent: 0  Rows_examined: 10811959
SELECT ...
```

MARIADB

- Made by Monty Widenius
  - creator of MySQL and MyISAM
- **Advertised** as the *good* MySQL
  - as opposed to *bad* MySQL that's being "killed" by ~~Evil Corp~~ Oracle
- MariaDB 5.5 = MySQL 5.5 + XtraDB + MariaDB 5.3:
  - query optimizer improvements
  - multi-master replication
  - group commit fix
- MariaDB 10
  - real *fork* of MySQL 5.6
  - may become "backwards" incompatible

MARIADB 10 FEATURES

- Parallel replication
- Multi-source replication
- Cassandra, Spider, TokuDB storage engines
- and few others.

SWITCH TO MARIADB?

- Think twice before switching
- No great advantages over MySQL 5.7 or Percona Server
  - And yet a number of performance improvements from 5.7 will be missed in MariaDB 10
- Some features may be appealing, if that’s your case - go for it
- Backwards incompatibility is a bit worrying
  - full mysqldump->import should still be possible

WEBSCALESQL

- Collaboration by a few heavy-duty MySQL users:
  - Alibaba, Facebook, Google, Linkedin & Twitter

- MySQL 5.6 at its core
  - good stuff being back-ported from MySQL 5.7

- Has very special mission
  - is not meant to be general purpose MySQL Server
  - addresses needs of running MySQL at **SCALE**

FEW WEBSCALESQL FEATURES

- Ability to specify millisecond timeouts
- Super read-only mode
- Ability to disable deadlock detection
- Prefix index query optimization
- Performance Schema not compiled in
- InnoDB flushing performance fixes

read this article by Laurynas @ Percona to learn more.

SWITCH TO WEBSCALESQL?

- Think thrice
- Only switch if you REALLY understand what it is
- Great performance features that don’t *hurt* functionality will be ported to other variants (some already are)

THE ESSENTIALS

Of MySQL Configuration Tuning

MYSQL DEFAULTS ARE POOR

- Okay for development, not for busy production db
- Good news:
  - MySQL 5.6 defaults are better than versions < 5.6
  - MySQL 5.7 defaults are better than 5.6
  - MySQL is super easy to configure

MYSQL CONFIGURATION FILE

- One configuration file (unless it contains *include* directives)
- Location:
  - /etc/my.cnf - most systems
  - /etc/mysql/my.cnf - Debian style
  - On Windows - best use data directory (create the file)

COMMON MISTAKES

- Using Trial and Error approach
  - change something, see if it *feels* better
  - don't do it
- Asking Google for performance advice
  - answers often lack context
  - use case could be benchmarks - often non-production suitable
  - settings are hardware (or else) dependant
- Obsessing about fine-tuning the my.cnf
  - 10-15 variables is often all you need to change
  - fine-tuning won't give you significant wins

COMMON MISTAKES (2)

- Changing many things at once
  - makes it very hard to figure out what caused which effect
  - instead, change one thing, then *measure* impact
- Not keeping my.cnf in sync with the changes you make
  - changing settings online is convenient, but…
  - don’t forget to update my.cnf
- Redundant entries in my.cnf
  - MySQL won’t mind them
  - Last value will be used
  - “-” and “_” can be used interchangeably

COMMON MISTAKES (3)

- Multiplying buffer sizes
  - don’t do it
  - some buffers are local, some server-wide
  - few variables need to be increased after hardware upgrades
- Using the wrong my.cnf section
  - [mysql], [client], [mysqld_safe] — all are incorrect choices
  - **[mysqld]** — put ALL of the server configuration here

CHANGING CONFIGURATION ONLINE

- Many things can be changed online (*dynamically*)
- Even *innodb_buffer_pool_size* starting with MySQL 5.7
- Example of changing *innodb_thread_concurrency* online:

```
mysql> show global variables like 'innodb%';
+-------------------------------------------+-------------------------+
| Variable_name                             | Value                   |
+-------------------------------------------+-------------------------+
...
| innodb_thread_concurrency                 | 0                       |
...
+-------------------------------------------+-------------------------+
140 rows in set (0.02 sec)

mysql> set global innodb_thread_concurrency = 8;
Query OK, 0 rows affected (0.01 sec)

mysql> select @@global.innodb_thread_concurrency;
+------------------------------------+
| @@global.innodb_thread_concurrency |
+------------------------------------+
|                                  8 |
+------------------------------------+
1 row in set (0.00 sec)
```

GLOBAL -VS- LOCAL SCOPE

- In many cases, you only want to change local session buffers and leave global configuration as is
  - *(sort|join|read|read_rnd)_buffer_size* are all good examples
- So:
  - For a query that needs to sort a lot of data, before you run it:

```
set sort_buffer_size = 64 * 1024 * 1024;
```

  - Some queries would find index merge intersect optimization harmful, so for such queries you can just:

```
set optimizer_switch = 'index_merge_intersection=off';
```

17

Key MySQL Settings

READY TO USE MY.CNF WITH ALL SETTINGS

I have prepared a my.cnf with short handy descriptions near each variable and appropriate links to learn more for your convenience. Download it here:

http://www.speedemy.com/17

1. DEFAULT_STORAGE_ENGINE – CHOOSE THE RIGHT ENGINE FIRST

- Already on InnoDB? Skip to next variable
- Otherwise, bear with me.

STORAGE ENGINE WHAT?

- MySQL uses pluggable storage engines since the beginning
- Different storage engines store data in different ways
  - some may even store it in a remote server (e.g. Federated)
  - others may only pretend they are storing data (e.g. Blackhole)
- Two most commonly used ones:
  - MyISAM
  - InnoDB
- Few other famous SEs:
  - Blackhole
  - Archive
  - CSV
  - TokuDB

WHAT'S WRONG WITH MYISAM

- Non transactional:
  - no **A**tomicity
  - no **C**onsistency
  - no **I**solation
  - no **D**urability
- Table level locks only
- Not scalable when it comes to write intensive workload
  - with an exception of INSERT-only workload

MEET INNODB

- Shipped with MySQL since 2001
- Fully ACID transactional storage engine, designed to handle highly concurrent workload and scale.
- Road to multi-core multi-disk systems was bumpy
  - handles high concurrency pretty well since MySQL 5.0.30
  - is hard to beat in many ways nowadays
- Default MySQL storage engine since version 5.5.5

HERE'S STORAGE ENGINES YOU ARE USING

- Run this on your server:

```
mysql> SELECT engine,
   count(*) as TABLES,
   concat(round(sum(table_rows)/1000000,2),'M') rows,
   concat(round(sum(data_length)/(1024*1024*1024),2),'G') DATA,
   concat(round(sum(index_length)/(1024*1024*1024),2),'G') idx,
   concat(round(sum(data_length+index_length)/(1024*1024*1024),2),'G') total_size,
   round(sum(index_length)/sum(data_length),2) idxfrac
  FROM information_schema.TABLES
 WHERE table_schema not in ('mysql', 'performance_schema', 'information_schema')
 GROUP BY engine
 ORDER BY sum(data_length+index_length) DESC LIMIT 10;
 +--------+--------+---------+--------+--------+------------+---------+
 | engine | TABLES | rows    | DATA   | idx    | total_size | idxfrac |
 +--------+--------+---------+--------+--------+------------+---------+
 | InnoDB |    181 | 457.58M | 92.34G | 54.58G | 146.92G    |    0.59 |
 | MyISAM |     13 | 22.91M  | 7.85G  | 2.12G  | 9.97G      |    0.27 |
 +--------+--------+---------+--------+--------+------------+---------+
 2 rows in set (0.22 sec)
```

** no need to rewrite this, it will be included in the ebook you will get at speedemy.com/17*

HERE’S MYISAM TABLES ON YOUR SERVER

- Run this:

```
mysql> SELECT
     concat(table_schema, '.', table_name) tbl,
     engine,
     concat(round(table_rows/1000000,2),'M') rows,
     concat(round(data_length/(1024*1024*1024),2),'G') DATA,
     concat(round(index_length/(1024*1024*1024),2),'G') idx,
     concat(round((data_length+index_length)/(1024*1024*1024),2),'G') total_size,
     round(index_length/data_length,2) idxfrac
  FROM information_schema.TABLES
 WHERE table_schema not in ('mysql', 'performance_schema', 'information_schema')
   AND engine = 'MyISAM'
 ORDER BY data_length+index_length DESC;
```

CONVERTING TO INNODB

- It’s not enough to change *default-storage-engine*
- Tables need to be converted. One. By. One.
- Simply converting largest tables is not enough:
    - if at least one table in a join is MyISAM, the entire query is using table level locks. So having even a small MyISAM table in a large join can be very bad for concurrency
- Convert only when the server is configured for InnoDB properly

CONVERTING EACH TABLE TO INNODB

- This will give you a list of commands to run:

```
SET @DB_NAME = 'your_database';

SELECT CONCAT('ALTER TABLE `', TABLE_NAME, '` ENGINE=InnoDB;') AS sql_statements
  FROM INFORMATION_SCHEMA.TABLES AS tb
 WHERE TABLE_SCHEMA = @DB_NAME
   AND ENGINE = 'MyISAM'
   AND TABLE_TYPE = 'BASE TABLE'
 ORDER BY TABLE_NAME DESC;
```

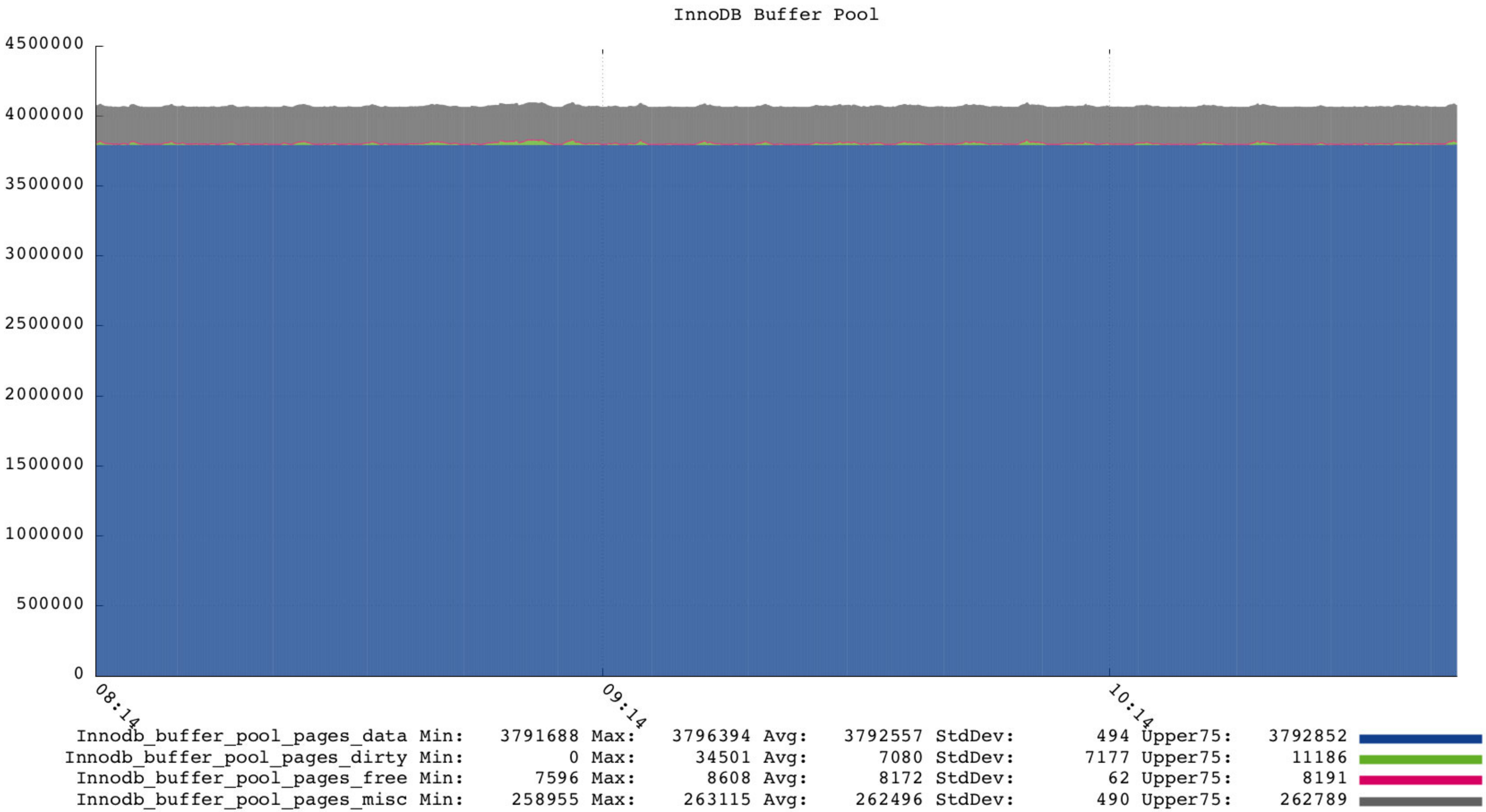
- To use InnoDB even when MyISAM is specified as a storage engine, in Percona Server you can set this in my.cnf:

```
enforce_storage_engine = InnoDB
```

2. INNODB_BUFFER_POOL_SIZE – GET THE BEST OUT OF YOUR MEMORY

- Most important variable for InnoDB
- InnoDB Buffer Pool is:
  - a cache for read data (these are stored in pages of 16kb)
  - also cache for indexes, modified (dirty) data
  - and place for many internal InnoDB structures

QUITE A TYPICAL BUFFER POOL DISTRIBUTION



```
 Innodb_buffer_pool_pages_data Min:   3791688 Max:   3796394 Avg:   3792557 StdDev:       494 Upper75:   3792852 🟦
Innodb_buffer_pool_pages_dirty Min:         0 Max:     34501 Avg:      7080 StdDev:      7177 Upper75:     11186 🟩
 Innodb_buffer_pool_pages_free Min:      7596 Max:      8608 Avg:      8172 StdDev:        62 Upper75:      8191 🟥
 Innodb_buffer_pool_pages_misc Min:    258955 Max:    263115 Avg:    262496 StdDev:       490 Upper75:    262789 ⬛
```

INNODB_BUFFER_POOL_SIZE = ?

- On a dedicated server, set to 80% of total memory
  - leaves room for other internal structures outside the buffer pool
  - query execution (session buffers)
  - OS cache (binary logs, relay logs, innodb transaction log, etc.)
  - OS memory structures
  - even on servers with 256-512GB of RAM
- Avoid swapping at all costs!
  - swapping is not the same as reading data from disk
  - it’s much much worse

INNODB_BUFFER_POOL_SIZE = ?

- On a shared server:
  - check total size of db, maybe it all fits in, say, 50% of RAM?
  - don’t strive to make it perfect
  - again, avoid swapping at all costs (use *vmstat 1* to check)
- Number of random reads from disk into BP can be checked by:

```
$ mysqladmin ext -ri1 | grep Innodb_buffer_pool_reads
 | Innodb_buffer_pool_reads                 | 1832098003     |
 | Innodb_buffer_pool_reads                 | 595            |
 | Innodb_buffer_pool_reads                 | 915            |
 | Innodb_buffer_pool_reads                 | 734            |
 | Innodb_buffer_pool_reads                 | 622            |
 | Innodb_buffer_pool_reads                 | 710            |
 | Innodb_buffer_pool_reads                 | 664            |
...
```

CHANGING INNODB_BUFFER_POOL_SIZE

- On MySQL 5.7, can be done online:

```
mysql> set global innodb_buffer_pool_size = size_in_bytes;
```

- MySQL log will go like:

```
[Note] InnoDB: Resizing buffer pool from 134217728 to 21474836480. (unit=134217728)
[Note] InnoDB: disabled adaptive hash index.
[Note] InnoDB: buffer pool 0 : 159 chunks (1302369 blocks) were added.
[Note] InnoDB: buffer pool 0 : hash tables were resized.
[Note] InnoDB: Resized hash tables at lock_sys, adaptive hash index, dictionary.
[Note] InnoDB: Completed to resize buffer pool from 134217728 to 21474836480.
[Note] InnoDB: Re-enabled adaptive hash index.
```

- Don’t forget to update my.cnf after this

- On any previous version of MySQL:
  - set an appropriate *innodb_buffer_pool_size* in *my.cnf*
  - restart MySQL server

3. INNODB_LOG_FILE_SIZE – ROOM FOR MYSQL'S REDO LOG

- Size for redo (a.k.a. transaction) logs
- 5MB up until MySQL 5.6.8 — way too small for just about any workload
  - Current default of 48MB still too small for most
- What's innodb log file?
  - next slide, please

WHAT IS INNODB REDO LOG?

- Transactional storage engines (including InnoDB) need both undo and redo functions, similar to image editor
  - well not exactly; let me explain
- Undo happens when rollback occurs
  - Implicitly or explicitly
- Redo happens after a database crash
  - Also when preparing a backup and other similar cases

UNDO

- When record is changed (but before COMMIT)
  - changes are not written to data files right away
  - first they are written to transaction log
  - and data is modified in memory
  - original *unmodified* copy is moved to *rollback segment*
- if Rollback occurs:
  - Undo needs to happen, to restore data to previous state
  - InnoDB removes the copy from the rollback segment (and copies it back if need be), removes the dirty page
  - and marks in transaction log that change was rolled back
- That's Undo.

REDO

- After COMMIT, changes are ready to be written to data files
  - but they aren't actually written (yet). This would be inefficient.
- Instead, changes are written to Redo log (and modified pages are only stored in memory)
- If MySQL crashes (before changes are written to data files):
  - Redo needs to happen
  - else, these changes would be lost forever
- After restart, InnoDB finds the last checkpoint position
  - and re-applies the same changes it had in memory before restart
- It is a gross oversimplification, but yeah, that's Redo.

REDO LOG SIZE

- Size matters:
  - Small log files make writes slower and crash recovery faster
  - Large log files make writes faster and crash recovery slower
- Log file is like a buffer, so:
  - small log files = small buffer, so flushing needs to happen often
  - big log files = big buffer, hence flushing is more streamlined
- But:
  - with large log files, in case of crash, more work needs to be done to restore the database to the consistent state (i.e. the Redo operation)

SIZING REDO LOG

- Rule of Thumb:
  - Check that total size of your Redo logs fits in 1-2h worth of writes during your busy period

```
mysql> pager grep seq
mysql> show engine innodb status\G select sleep(60); show engine innodb status\G
Log sequence number 1777308180429
...
Log sequence number 1777354541591

mysql> nopager
mysql> select (1777354541591-1777308180429)*60/1024/1024;
+---------------------------------------------+
| (1777354541591-1777308180429)*60/1024/1024 |
+---------------------------------------------+
|                               2652.80696869 |
+---------------------------------------------+
1 row in set (0.00 sec)
```

- Based on this 60s sample, InnoDB *could be* writing around 2.6GB per hour, so *innodb_log_file_size=2560M* sounds like a good start to get 5GB worth of redo logs total

CHANGING THE REDO LOG SIZE

- MySQL ≥ 5.6 - change my.cnf and restart the server.
- MySQL < 5.6, my.cnf change is **not enough:**
  1. change innodb_log_file_size in my.cnf
  2. stop MySQL server
  3. ensure MySQL had a clean shutdown (mysql log is your friend)
  4. remove old log files, *usually* by running the following command: `rm -f /var/lib/mysql/ib_logfile*`
  5. start MySQL server – it should take a bit longer to start because it is going to be creating new transaction log files
- MySQL < 5.6.2 only supports 4GB total!

4. INNODB_FLUSH_LOG_AT_TRX_COMMIT – DURABLE OR NOT? THAT IS THE QUESTION!

- *innodb_flush_log_at_trx_commit=1* by default:
  - FLUSH and SYNC after EVERY transaction commit - full durability
  - INSERT/UPDATE/DELETE is a *transaction* if *autocommit=1* (default behaviour)
- SYNC is often expensive - it's synchronisation to disks
  - Exception is if you have non-volatile cache
    - e.g. battery-backup unit (BBU) protected write-back cache
    - or Super-capacitor with a flash memory chip
- Alternatives values for *innodb_flush_log_at_trx_commit*:
  - 0 means FLUSH the buffers to OS, but DO NOT SYNC (no actual IO is performed on commit)
  - 2 means DON'T FLUSH and DON'T SYNC (again no actual IO is performed on commit)

INNODB_FLUSH_LOG_AT_TRX_COMMIT = ?

- When 0 or 2 is chosen, SYNC is performed once per second:
  - Means you may loose up to 1s worth of committed data
  - Once per second is not guaranteed*
- *innodb_flush_log_at_trx_commit=1* is full durability
  - Required for bank transactions and similar financial operations
  - Many websites use 0 or 2 instead
    - After all, MyISAM would loose up to 30s worth of data in case of crash and it’s been default for many many years
- So then, 0 or 2?
  - Small difference, because neither causes SYNC to disk
  - 0 is good in that no data is lost if MySQL crashes (but the machine stays ON)

* I’v seen SYNC delays due to mutex contention issues

5. SYNC_BINLOG – THAT'S FOR DURABLE BINLOG

- *sync_binlog=1* makes binary logs durable
- Therefore:
  - if you have no slaves & no backups - use *sync_binlog=0* (default)
  - if you do have replication and/or backups, but you don't mind loosing few events (on slave or otherwise) in case of server power loss in order to gain better performance, *sync_binlog=0*
  - if consistency is really important and you also use *innodb_flush_log_at_trx_commit=1* anyway, do use *sync_binlog=1* and make sure you run MySQL 5.6+, Percona Server 5.6+ or MariaDB 5.5+ as these versions have a binlog group commit fix (meaning that sync calls are grouped together)

6. INNODB_FLUSH_METHOD – AVOID DOUBLE BUFFERING

- Set *innodb_flush_method=O_DIRECT*
  - Only supported in Linux
  - Overcomes OS cache for reads and for writes
- If not used, double buffering occurs
  - Bad because memory is wasted on storing the same data in memory twice (InnoDB buffer pool and OS cache)
- Very few exceptions to this
  - If you're not sure, you're not an exception.

7. INNODB_BUFFER_POOL_INSTANCES – REDUCE MUTEX CONTENTION

- Introduced in MySQL 5.5
- Reduces global buffer pool mutex contention
  - splits buffer pool into multiple buffer pools
- On MySQL 5.5 (including variants) be more conservative:
  - *innodb_buffer_pool_instances=4* should be good enough
- On MySQL 5.6+ (including variants):
  - *innodb_buffer_pool_instances=8* or even *16* is a safe bet

8. INNODB_THREAD_CONCURRENCY – CONTROL YOUR THREADS

- *innodb_thread_concurrency=0* is default and often used in benchmarks, but
  - with high workloads, setting a limit *may* work much better
- How does it work? I wish I could show you, but let me at least describe it.
  - next slide, please

INNODB CONCURRENCY MECHANISM

- *innodb_thread_concurrency* controls how many threads can be executing in parallel
  - if 0, all requests will be served immediately
    - that's fine if you have 32 CPU cores and 4 requests
    - not so if you have 32 CPU intensive requests and 4 CPU cores
    - can become a mess when all are executing at the same time and new requests keep coming in
- number > 0 caps the number of threads that are executing at the same time
  - But it's not a simple FIFO queue
  - Each request is given a certain number of tickets
    - 500 by default on MySQL 5.5 and earlier
    - 5000 by default on MySQL 5.6 and newer

INNODB CONCURRENCY MECHANISM (2)

- A thread waits in the queue for a slot to become available
  - once it starts executing, it starts using the tickets
  - one ticket is used for every row read, insert, update, etc.
  - when all tickets are used, a thread is sent to the back of the queue and the cycle repeats until operation is complete
- The advantage is that long running queries don't prevent quick queries from ever getting into the queue, even if that does prolong their execution.
- And of course that there's no fighting for resources

INNODB CONCURRENCY CONTROLS

- *innodb_concurrency_tickets* determines the number of tickets given
    - increase for long queries to run longer before letting others in
- *innodb_thread_sleep_delay* sets amount of sleep time before joining the innodb queue (in microseconds)
- *innodb_thread_concurrency* sets how many slots are available for execution
    - use if you have spikes or hardware is often saturated
    - set to 8 and go up until you see good hardware utilization
    - it is a dynamic variable:
        - set global innodb_thread_concurrency=8;

MONITORING THE QUEUE

- You can see the number of requests queued up and executing in the *“show engine innodb status\G”* output.
- Look for something like:

```
22 queries inside InnoDB, 104 queries in queue
```

9. SKIP_NAME_RESOLVE – DO SKIP THAT REVERSE IP LOOKUP

- Add skip_name_resolve to avoid DNS resolution on connect
- No impact when all is working fine
- When DNS server fails, takes a long time to figure things out
  - slow connections due to DNS failure don't help to solve this faster
- Exception: local hosts file based names

HOW ABOUT A DEEP BREATH

and a quick stretch ?

BETTER?

OK, LET'S CONTINUE

10. INNODB_IO_CAPACITY(_MAX)? – CAP INNODB IO USAGE

- *innodb_io_capacity* controls how many write IO requests per second (IOPS) will MySQL issue when flushing the dirty data
- *innodb_io_capacity_max* controls how many write IOPS will MySQL issue flushing the dirty data when it's under stress
- IO activity related to background writes only
- Under stress means MySQL is behind with flushing activity and needs to shift gears or things may go bad
- Set *innodb_io_capacity* to 50-75% of write capacity
  - and *innodb_io_capacity_max* to 100% or close
  - write capacity = number of random write iops server can handle (more on it here)

11. INNODB_STATS_ON_METADATA – TURN THEM OFF!

- OFF by default on MySQL 5.6 and 5.7
- Safe to switch OFF on MySQL 5.5 and 5.1 too, so in my.cnf:
  - innodb_stats_on_metadata = 0
  - can be changed online too
- Makes “show table status” and some queries against INFORMATION_SCHEMA faster.
- InnoDB stats are still going to be updated, don’t worry!

12. INNODB_BUFFER_POOL_(DUMP_AT_SHUTDOWN|LOAD_AT_STARTUP)

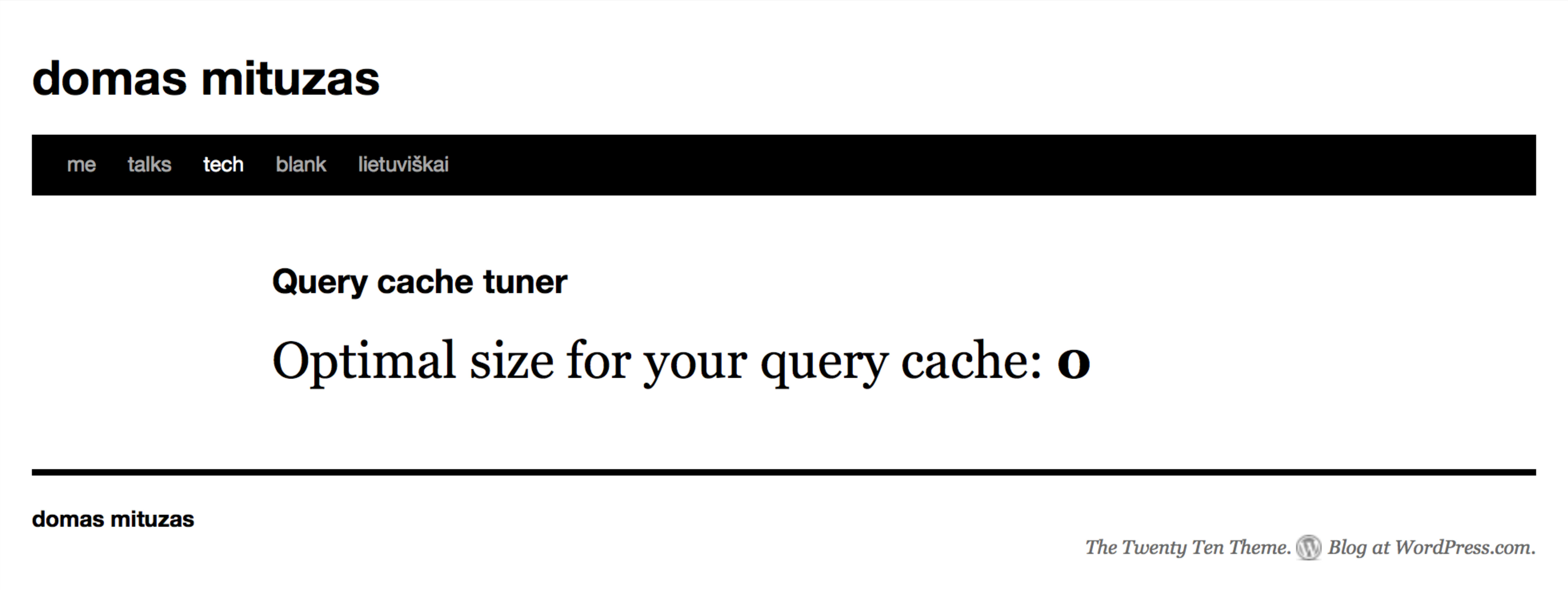
- Makes warm-up much faster
  - works even with SSDs
  - by loading contents of buffer pool on server startup
- Only page reference numbers are stored in a file
- Asynchronous activity, no direct performance impact
- *innodb_buffer_pool_dump_pct* in MySQL 5.7 to control how much of the buffer pool to dump (25 by default, I recommend 75-100)
- Supported in MySQL since 5.6, in Percona Server - since 5.1

13. INNODB_ADAPTIVE_HASH_INDEX_PARTS – SPLIT THE AHI MUTEX

- Adaptive Hash Index is ON by default
- Dynamic Hash index maintained by InnoDB to improve certain query patterns. Usually very helpful
  - except when requests for mutex start backfiring
- Starting with MySQL 5.7 mutex can be split (and is by default split into 8 partitions), i.e.
  - *innodb_adaptive_hash_index_parts=8*
- Using MySQL 5.6 and earlier?
  - Switch to an appropriate Percona Server or MariaDB version and use *innodb_adaptive_hash_index_partitions*

14. QUERY_CACHE_TYPE – ON? OFF? ON DEMAND?

- Before it becomes a bottle-neck, query cache is great with:
  - small databases with few updates
  - low concurrency workload
  - read-only databases
- Even if it's helpful, don't exceed *query_cache_size*=*256M*
  - wait time on invalidation increases significantly
  - innocent queries get blocked
- With high concurrency workload, often a bottle-neck
- Domas Mituzas suggests using this query cache tuner (next slide, please)



domas mituzas

Query cache tuner

Optimal size for your query cache: **0**

domas mituzas

The Twenty Ten Theme. Blog at WordPress.com.

QUERY CACHE TUNING

- More seriously though, mutex is still locked even if *query_cache_size=0*
- Use the following configuration:
  - *query_cache_size=0*
  - *query_cache_type=OFF*
- Requires a MySQL restart to disable mutex
- Works with MySQL 5.5 or newer
  - If using MySQL 5.1, switch to Percona Server 5.1 to get the same effect

15. INNODB_CHECKSUM_ALGORITHM – THE SECRET HARDWARE ACCELERATION TRICK

- Old checksum algorithm is expensive (CPU overhead)
- crc32 can use native CPU instructions. That's a YUUGE win.
- Use *innodb_checksum_algorithm=crc32* with MySQL 5.6
  - used by default in MySQL 5.7
- Safe to change, no need to reload data
- Can be changed online in fact

16. TABLE_OPEN_CACHE_INSTANCES – IT'S THERE FOR A REASON

- Introduced in MySQL 5.6.6 to split open cache instances
- Avoid server lock-up when opening many tables or when opening a table is slow
- Starting with MySQL 5.7.8, *table_open_cache_instances=16*
    - Set this manually in MySQL 5.6

17. INNODB_(READ|WRITE)_IO_THREADS – LAST AND, YES, LEAST

- MySQL supports Asynchronous IO on Linux since MySQL 5.5, so this is not as important as it may seem.
- These threads are used for background activities only
- Set it to match number of bearing read/write disks
  - e.g. *innodb_read_io_threads=8* and *innodb_write_io_threads=4* on RAID10 with 8 disks.
  - on SSDs, set at 32/16 (or similar) respectively

NOT THAT I DIDN'T MENTION IT, BUT

I have prepared a my.cnf with short handy descriptions near each variable and appropriate links to learn more for your convenience. Download it here:

http://www.speedemy.com/17

FINAL THOUGHTS

- Configuration file is only part of the deal
- More often than not, real the devil is in queries
- To learn about query optimization, register for a free webinar on query optimization:

http://www.speedemy.com/webinars

END

Speedemy

Speedemy_com

Speedemy.com

P.S. I have prepared a video where I have discussed MySQL configuration in great detail. It also contains an additional section about reading MySQL Status Counters, in case you’re interested. Register here to get the free video once post production is finished.