17 Key MySQL Config File Settings (MySQL 5.7 proof)
By Aurimas Mikalauskas, Speedemy.com

17 KEY MYSQL CONFIG FILE SETTINGS (MYSQL 5.7 PROOF)

In MySQL (https://www.speedemy.com/category/mysql/) by Aurimas Mikalauskas / December 4, 2015 / 4 Comments (https://www.speedemy.com/17-key-mysql-config-file-settings-mysql-5-7-proof/#disqus_thread)

When MySQL becomes too slow (or too unstable), temptation usually is to tweak the MySQL configuration file. Indeed, it's a good place to start. But if you ever looked at the available configuration options, you know things can get messy – MySQL now has over 450 configuration variables for your consideration, that are not classified in any way, and neither of them are included in the stock my.cnf. It's hard to know where to start!

I'm hoping that this blog post will help you overcome the anxiety of tuning MySQL, whether you're setting up a new server, or tuning an already running server for better performance.

DON'T DO IT THE WAY ROOKIES DO IT

During the last 9 years I've spent at Percona working as a MySQL performance and scalability consultant, I found that customers often use the **trial and error approach** when tuning MySQL configuration: they change a few things and check if it "feels" better.

This doesn't work! Problem is that by the time you're measuring how your application "feels", situation may have changed already. Plus you have to consider the warm up time and disregard any overhead you see due to it. And most importantly, you can't base your optimization efforts on feelings – it's either faster, more scalable (and you understand why) or it's not.

After such a "tuning" session you may end up with a worse configuration than what you have started with. In fact, more often than not, that's exactly what I would find when a customer would first contact me.

WHY 17 CONFIG VARIABLES

So I decided go through *all* of the MySQL 5.1-5.7 settings and distill the ones that, as I have discovered during these 9 years, really matter for MySQL performance. Based both on my experience, benchmarks and, most importantly, real life tests with our customers systems. Here's what I found:

When it comes to MySQL Performance, there's only 17 config file settings you should be looking at. And, you can easily determine the *correct* values for most of them!

Please make sure to get yourself familiar with all of these variables, because some of them are related and can't be tuned in isolation. Also before you change anything, read the "First things first" section below where I discuss what you should be wary of when working with MySQL configuration.

FIRST THINGS FIRST

Before we get to the list, here's a few things you should keep in mind when working with MySQL configuration. These are the basic principles and most common mistakes I see being made:

1. Never trust the first response you found on Google (even if that's this blog post) and never do quick reckless changes. Understand exactly what you are changing. I will try to be verbose in this blog post, but feel free to look for more resources so you really understand what you are doing.

2. Don't get obsessed about fine-tuning the configuration – usually 10-15 variables will give you the most impact (you will find all of them here), and fine-tuning the variables is highly unlikely to have any additional benefits. It can do harm though. If you still have a performance problem even after you have applied all the recommendations (and gotten rid of everything that you shouldn't have touched in the first place), the problem is probably somewhere else – bad queries, lack of resources, etc.

3. Change one thing at a time. Especially if you already have a solid configuration. If it's a new setup, or you were running with default configuration until now, feel free to go wild and implement all of these changes at once. Otherwise, change one thing and take some time to monitor the server after the change.

4. Don't forget to update my.cnf if you made the changes online. You can change a lot of things online these days, in MySQL 5.7 you can even modify innodb buffer pool size online. But if you take that route, make sure you also update my.cnf or you will lose all the changes when MySQL is restarted.

5. Avoid duplicate entries. If you use the same variable twice, MySQL will not complain about it. It will just use the last value, so be sure you don't add the same variable twice, otherwise you may end up changing the wrong variable (and not seeing an impact). Also note that a dash "-" and an underscore "_" can be used interchangeably, so "innodb-log-file-size" and "innodb_log_file_size" are both referring to the same server setting.

6. Don't just double the size of all buffers. Some buffers are local, some global. Some are storage engine related, some are server wide. So if you have just added twice more memory to the server, check every variable you want to change before changing it and see if it makes sense to do so. Oh and if it's not mentioned in this list below, most likely you don't need to change it anyway.

7. Use the right section – [mysqld]. MySQL configuration file can have many sessions, such as [mysql], [client], [mysqld_safe] and so on. When tuning MySQL server, make sure you are working with the [mysqld] section, or your changes will have no impact. The only exception here is if you're using the mysqld_multi script, in which case you'll be working with several sections at once.

THE 17 KEY MYSQL VARIABLES

FYI, I have prepared a special my.cnf file that includes all of these 17 variables with short explanations and links back to appropriate sections with more verbose explanations. You will be able to download it below.

And now let's get down to business. Here's the list of key MySQL variables you should be tuning to get the best performance out of your MySQL server:

1. DEFAULT_STORAGE_ENGINE – CHOOSE THE RIGHT ENGINE

If you're already on MySQL 5.6 or 5.7 and all of your tables are InnoDB, you're all set. If not, make sure you switch to InnoDB and set `default_storage_engine` to InnoDB.

Why? In a nutshell, because InnoDB is the best MySQL (and Percona Server, and MariaDB) storage engine – it supports transactions, high concurrency, has an amazing performance (when configured properly). And here's the long format version of why (http://www.speedemy.com/mysql/17-key-mysql-config-file-settings/default_storage_engine/).

If you do not agree that InnoDB is the way to go, stop right there. Do not read any further. Your eyes will bleed!

2. INNODB_BUFFER_POOL_SIZE – GET THE BEST OUT OF YOUR MEMORY

This is the most important InnoDB variable. Actually, if you're using InnoDB as your main storage engine, for you – it's THE most important variable for the entire MySQL. In fact, it's so important I have created a separate page just for innodb buffer pool size (http://www.speedemy.com/mysql/17-key-mysql-config-file-settings/innodb_buffer_pool_size/).

Essentially, `innodb_buffer_pool_size` specifies how much memory MySQL should allocate for InnoDB buffer pool, and InnoDB buffer pool is where InnoDB keeps cached data, secondary indexes, dirty data (data that's been modified, but not yet written to disk) and various internal structures such as Adaptive Hash Index.

As a rule of thumb, on a dedicated MySQL server you want to set it to 80% of total server memory. If you run a shared server, or if you wonder whether your innodb buffer pool is sized properly, see here for details (http://www.speedemy.com/mysql/17-key-mysql-config-file-settings/innodb_buffer_pool_size/).

3. INNODB_LOG_FILE_SIZE – ROOM FOR MYSQL'S REDO LOG

This sets the size of the InnoDB's redo log files which, in MySQL world, are often called simply transaction logs. And right until MySQL 5.6.8 the default value of `innodb_log_file_size=5M` was the *single biggest InnoDB performance killer*. Starting with MySQL 5.6.8, the default was raised to 48M which, for many intensive systems, is still way too low.

As a rule of thumb you should set this to accommodate ~1-2h worth of writes and if you're in a hurry, having this set to 1-2G will give you pretty good performance with pretty much any workload. However, because this is so important, I have created a special page for this variable as well – click here to learn more about MySQL's redo logging and how to fine-tune the innodb_log_file_size (http://www.speedemy.com/mysql/17-key-mysql-config-file-settings/innodb_log_file_size/).

Oh and before we go to the next variable, let me mention `innodb_log_buffer_size` real quick. "Mention" because it is often not well understood and as a consequence over-appreciated. The reality is that most of the time you will only be using a small fraction of this buffer – enough to hold the changes of your tiny transactions before they are committed and changes are written to disk.

Of course, if you have large transactions that do lots of changes, then yes, it may make sense to have a larger than default innodb log buffer size, but if you're using `autocommit` or if your transactions make no more than few kilobytes of changes, stay with the default `innodb_log_buffer_size`.

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

By default, `innodb_flush_log_at_trx_commit` is set to 1 which instructs InnoDB to flush AND *sync* after EVERY transaction commit. And if you are using `autocommit`, then every single INSERT, UPDATE or DELETE statement is a transaction commit.

Sync is an expensive operation (especially when you don't have a write-back cache) as it involves the actual synchronous physical write to the disk, so whenever possible, I would recommend to avoid using this default configuration.

Two alternative values for this variable are 0 and 2:

- 0 means flush to disk, 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).

So if you have it set to 0 or 2, sync is performed **once a second** instead. And the obvious disadvantage is that you may lose last second worth of **committed** data. Yes, you read that right – those transactions would have committed, but if server loses power, those changes never happened.

Obviously, for financial institutions such as banks it's a huge no-go. Most websites, however, can (and do) run with `innodb_flush_log_at_trx_commit=0|2` and have no issues even if the server crashes eventually. After all, just few years ago many websites were using MyISAM, which *will* lose up to 30s worth of written data in case of a crash. (not to mention the crazy slow table repair process).

Finally, what's the practical difference between 0 and 2? Well, performance wise the difference is negligible really, because a flush to OS cache is cheap (read *fast*). So it kind of makes sense to have it set to 0, in which case if MySQL (but not the whole machine) crashes, you do NOT lose any data as it will be in OS cache and synced to disk from there eventually.

BTW, if you prefer durability to performance and have innodb_flush_log_at_trx_commit set to 1, let me draw your attention to the next variable which is closely related:

5. SYNC_BINLOG – THAT'S FOR DURABLE BINLOG

Tens of pages have been written about `sync_binlog` and it's relationship with `innodb_flush_log_at_trx_commit`, but let me simplify it for you for now:

a) If your MySQL server has no slaves and you don't do backups, set `sync_binlog=0` and be happy with a good performance
b) If you do have slaves and you do backups, but you don't mind losing a few events from the binary logs in case of a master crash, you may still want to use `sync_binlog=0` for the sake of a better performance.
c) If you do have slaves and/or backups, and you do care about slaves consistency and/or point in time recovery (ability to restore your database to a specific point in time by using your latest consistent backup and binary logs) and you are also running `innodb_flush_log_at_trx_commit=1`, then you should *seriously consider* using `sync_binlog=1`.

Problem is of course that `sync_binlog=1` has a pretty significant price – now every single transaction is again synced to disk – to the binary logs. You'd think why not do both sync operations at once, and you'd be right – new versions of MySQL (both 5.6 and 5.7, MariaDB and Percona Server) already have a properly working group commit, in which case the price for this `sync_binlog=1` is not that big, but older versions of MySQL have a really significant performance penalty, so be careful with that and watch your disk writes.

So far so good? Good. Next.

6. INNODB_FLUSH_METHOD – YOUR CHANCE TO AVOID DOUBLE BUFFERING

Set `innodb_flush_method` to `O_DIRECT` to avoid double-buffering. The only case you should NOT use O_DIRECT is when it is not supported by your operating system. But if you're on Linux, use O_DIRECT to enable direct IO.

Without direct IO, double-buffering happens because all database changes are first written to OS cache and then synced to disk – so you end up with the same data in InnoDB buffer pool AND in OS cache. Yes, that means in a write-intensive environment you could be losing up to almost 50% of memory, especially if your buffer pool is capped at 50% of total memory. And if not, server could end up swapping due to high pressure on the OS cache.

In other words, do set `innodb_flush_method=O_DIRECT` .

7. INNODB_BUFFER_POOL_INSTANCES – REDUCE GLOBAL MUTEX CONTENTION

MySQL 5.5 introduced buffer pool instances as a means to reduce internal locking contention and improve MySQL scalability. In version 5.5 this was known to improve the throughput to some degree only, however MySQL 5.6 was a big step up, so while in MySQL 5.5 you may want to be more conservative and have `innodb_buffer_pool_instances=4` , on MySQL 5.6 and 5.7 you may go with 8-16 buffer pool instances.

Your mileage may vary of course, but with most workloads, that should give you best results.

Oh and obviously do not expect this to make any difference to a single query response time. The difference will only show with highly concurrent workloads i.e. those with many threads doing many things in MySQL at the same time.

8. INNODB_THREAD_CONCURRENCY – HAVE BETTER CONTROL OVER YOUR THREADS

You may hear very often that you should just set `innodb_thread_concurrency=0` and forget it. Well, that's only true if you have a light or moderate workload. However, if you're approaching the saturation point of your CPU or IO subsystem, and especially if you have occasional spikes when the system needs to operate properly when overloaded, then I would highly recommend to tackle `innodb_thread_concurrency` .

Here's the thing – InnoDB has a way to control how many threads are executing in parallel – let's call it a concurrency control mechanism. And for the most part it is controlled by `innodb_thread_concurrency`. If you set it to zero, concurrency control is off, therefore InnoDB will be processing all requests immediately as they come in (and as many as it needs to).

This is fine if you have 32 CPU cores and 4 requests. But imagine you have 4 CPU cores and 32 requests – if you let the 32 requests run in parallel, you're asking for trouble. Because these 32 requests will only have 4 CPU cores, they will obviously be at least 8 times slower than usually (in reality, more than 8 times slower of course), but each of those requests will have it's own external and internal locks which leaves great opportunities for all the queries to pile up.

OK, not to overwhelm you right now, I'll stop there and I will only mention that there's two other variables – `innodb_thread_sleep_delay` and `innodb_concurrency_tickets` – that will help you take control over concurrency. I will write all about how this concurrency control mechanism works in a few weeks and in the meantime, I invite you to do some experimentation with `innodb_thread_concurrency`, after all you can change it online by simply running the following command:

```
SET global innodb_thread_concurrency=X;
```

For most workloads and servers, 8 is a good start and then you should only increase it gradually if you see that the server keeps hitting that limit while hardware is under-utilised. To see where the Threads stand at any given moment, run `show engine innodb status\G` and look for a similar line in the ROW OPERATIONS section:

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

9. SKIP_NAME_RESOLVE – DO SKIP THAT REVERSE IP LOOKUP

This is a funny one, but I couldn't not mention it as it's still quite common to see it not being used. Essentially, you want to add `skip_name_resolve` to avoid DNS resolution on connection.

Most of the time you will feel no impact when you change this, because most of the time DNS servers work pretty fast (and they also cache results). But when a DNS server will fail, it will be such a PITA to figure out what are those "unauthenticated connections" doing on your server and why things all of the sudden seem slow…

So. Don't wait until this happens to you. Add this variable now and get rid of any hostname based grants. The only exception here is if you're using hosts file based name resolution. Or if your DNS servers will never fail (ha ha).

10. INNODB_IO_CAPACITY, INNODB_IO_CAPACITY_MAX – CAP INNODB IO USAGE

These two variables deserve a special article just for them (and they will get one), but for now, let me brief you in:

- `innodb_io_capacity` controls how many *write* IO requests per second (IOPS) will MySQL do when flushing the dirty data,
- `innodb_io_capacity_max` controls how many *write* IOPS will MySQL do flushing the dirty data when it's *under stress*.

So, first of all, this has nothing to do with foreground reads – ones performed by SELECT queries. For reads, MySQL will do the maximum number of IOPS possible to return the result as soon as possible. As for writes, MySQL has background flushing cycles and during each cycle it checks how much data needs to be flushed and it will use no more than `innodb_io_capacity` IOPS to do the flushing. That also includes change buffer merges (change buffer is a where secondary keys dirty pages are stored until they are flushed to disk).

Second, I need to clarify what "under stress" means. This, what MySQL calls an "emergency", is a situation when MySQL is behind with flushing and it needs to flush some data in order to allow for new writes to come in. Then, MySQL will use the *innodb_io_capacity_max* amount of IOPS.

Now here's the $100 question: so what do you set the `innodb_io_capacity` and `innodb_io_capacity_max` to?

Best solution – measure random write throughput of your storage and set `innodb_io_capacity_max` to the maximum you could achieve, and `innodb_io_capacity` to 50-75% of it, especially if your system is write-intensive.

Often you can predict how many IOPS your system can do, especially if it has magnetic drives. For example, 8 15k disks in RAID10 can do about 1000 random write IOPS, so you could have `innodb_io_capacity=600` and `innodb_io_capacity_max=1000`. Many cheap enterprise SSDs can do 4,000-10,000 IOPS etc.

Nothing bad will happen if you don't make it perfect. But. Beware that the default values of 200 and 400 respectively could be limiting your write throughput and consequently you may have occasional stalls for the flushing activity to catch up. If that's the case – you could be either saturating your disks, or you don't have a high enough values for these variables.

11. INNODB_STATS_ON_METADATA – TURN THEM OFF!

If you're running MySQL 5.6 or 5.7 and you didn't change the default `innodb_stats_on_metadata` value, you're all set.

On MySQL 5.5 or 5.1, however, I highly recommend to turn this OFF – that way commands like `show table status` and queries against `INFORMATION_SCHEMA` will be instantaneous instead of taking seconds to run and causing additional IO.

My former colleague Stephane Combaudon from Percona has written a very good blog post on this (https://www.percona.com/blog/2013/12/03/innodb_stats_on_metadata-slow-queries-information_schema/).

Oh and note that starting with 5.1.32, this is a dynamic variable, so you don't need to restart MySQL just to disable that.

12. INNODB_BUFFER_POOL_DUMP_AT_SHUTDOWN & INNODB_BUFFER_POOL_LOAD_AT_STARTUP

It may seem the two variables `innodb_buffer_pool_dump_at_shutdown` and `innodb_buffer_pool_load_at_startup` are not performance related, but if you are occasionally restarting your MySQL server (e.g. to apply configuration changes), they are. When both are enabled, the contents of MySQL buffer pool (more specifically, cached pages) are stored into a file upon shutdown. And when you start MySQL, it starts a background thread that loads back the contents of buffer pool and improves the warm-up speed that way up to 3-5 times.

Couple of things:

First, it doesn't actually copy the contents of the buffer pool into a file upon shutdown, it merely copies the *tablespace IDs* and *page IDs* – enough information to locate the page on disk. Then, it can load those pages really fast with large sequential reads instead of hundreds of thousands of small random reads.

Second, loading of the contents on startup happens in the background, hence MySQL doesn't wait until the buffer pool contents are loaded and starts serving requests immediately (so it's not as intrusive as it may seem).

Third (I know, I know, three isn't really a couple anymore), starting with MySQL 5.7.7, only 25% of least recently used buffer pool pages are dumped on shutdown, but you have total control over that – use `innodb_buffer_pool_dump_pct` to control how many pages (expressed in percents) would you like to be dumped (and restored). I vote 75-100.

And fourth (sorry!), while MySQL only supports this since MySQL 5.6, in Percona Server, you had this for quite a while now – so if you want to stay on version 5.1 or 5.5, check this out (https://www.percona.com/doc/percona-server/5.5/management/innodb_lru_dump_restore.html) (version 5.1 link (https://www.percona.com/doc/percona-server/5.1/management/innodb_lru_dump_restore.html)).

13. INNODB_ADAPTIVE_HASH_INDEX_PARTS – SPLIT THE AHI MUTEX

If you're running a lot of SELECT queries (and everything else is perfectly in tune), then Adaptive Hash Index is going to be your next bottle-neck. Adaptive Hash Indexes are dynamic indexes maintained internally by InnoDB that improve performance for your most commonly used query patterns. This feature can be turned off with a server restart, but by default it is ON in all versions of MySQL.

While it's quite a complex technology (more on it here (https://dev.mysql.com/doc/refman/5.5/en/innodb-adaptive-hash.html)), in most cases it gives a nice boost to many types of queries. That is, until you have too many queries hitting the database, at which point they start spending too much time waiting on the AHI locks and latches.

If you're on MySQL 5.7, you won't have any issues with that – `innodb_adaptive_hash_index_parts` is there and it's set to 8 by default, so the adaptive hash index is split into 8 partitions, therefore there's no global mutex and you're good to go.

All MySQL versions before 5.7, unfortunately, have no control over the number of AHI partitions. In other words, there's one global mutex protecting the AHI and with many select queries you're constantly hitting this wall.

So if you're running 5.1 or 5.6, and you have thousands of select queries per second, the easiest solution would be to switch to same version of Percona Server and enable AHI partitions. The variable in Percona Server is called `innodb_adaptive_hash_index_partitions`.

14. QUERY_CACHE_TYPE – ON? OFF? ON DEMAND?

A lot of people think Query cache is great and you should definitely use it. Well, sometimes it's true – sometimes it is useful. But it's only useful when you have a relatively light workload and especially if the workload is pretty much read-only with very few or virtually no writes.

If that's your workload, set `query_cache_type=ON` and `query_cache_size=256M` and you're done. Note, however, you should never set the query cache size any higher, or you will run into serious server stalls due to query cache invalidation. I've seen this too many times and until someone figures out a way to split a global query cache mutex, this will not go away.

Now if you have an intensive workload, then I would recommended this query cache size tuner (http://dom.as/2009/07/08/query-cache-tuning/) written by one of MySQL community leaders – Domas Mituzas.

More seriously though, you should not only set the `query_cache_size=0` but also it's very important that you set `query_cache_type=OFF` and restart the server when you do that – this way MySQL will stop using query cache mutex for all queries – even those that wouldn't use the query cache anyway.

So if you're still using the query cache and you shouldn't, make these changes now or you will suffer!

15. INNODB_CHECKSUM_ALGORITHM – THE SECRET HARDWARE ACCELERATION TRICK

Most mainstream CPUs nowadays support native crc32 instructions and MySQL can finally make use of that to improve the speed of calculating the InnoDB checksums significantly. Here's the two variables you should enable to make use of that:

1. `innodb_checksum_algorithm=crc32` (available since MySQL 5.6)
2. ~~`innodb_log_checksum_algorithm=crc32` (available since MySQL 5.7)~~

Starting with MySQL 5.7.7, `innodb_checksum_algorithm=crc32` is set by default, and now, starting with MySQL 5.7.9 GA, `innodb_log_checksum_algorithm` is no longer available, instead `innodb_log_checksums` was introduced which is ON by default, so you don't need to do anything here.

In MySQL 5.6, however, you only have `innodb_checksum_algorithm` option, but do set it to crc32 to enable hardware acceleration for page checksums.

Checksums are calculated every single time a page (or log entry) is read or written, so this is definitely YUUGE! (right, Donald?)

Oh and BTW, this is totally safe to change on a server that already has tables created with checksum type “innodb”. In fact, you can change it dynamically, online, while the server is running (There, I said it three times, now it’s completely redundant).

16. TABLE_OPEN_CACHE_INSTANCES – IT’S THERE FOR A REASON

Starting with MySQL 5.6.6, table cache can be split into multiple partitions and if you’re running MySQL 5.6 or newer, definitely do that.

Table cache is where the list of currently opened tables is stored and the mutex is locked whenever a table is opened or closed (and, in fact, in a number of other cases) – even if that’s an implicit temporary table. And using multiple partitions definitely reduces potential contention here. I’ve seen this LOCK_open mutex issue in the wild too many times.

Starting with MySQL 5.7.8, `table_open_cache_instances=16` is the default configuration, and I’d say it is definitely a good starting point both on MySQL 5.6 and 5.7.

17. INNODB_READ_IO_THREADS & INNODB_WRITE_IO_THREADS – LAST AND, YES, LEAST

I’ve placed this last because it’s really not as important as it may seem.

First of all, your MySQL is likely already using Asynchronous IO (AIO) which on Linux is supported since MySQL 5.5. Second, both `innodb_read_io_threads` and `innodb_write_io_threads` are only used for the background activity, such as checkpointing (flushing dirty pages to disk), change buffer merge operations and, sometimes, read-ahead activity.

So, while it’s not a key variable to tune, aligning it to the number of bearing disks is still a good idea. So for example on RAID10 with 8 disks, you may want to have `innodb_read_io_threads=8` and `innodb_write_io_threads=4` . If you have an SSD, well then just have it around 16-32, but do not expect much performance difference, unless your server is extremely write-heavy and disk IO *is* the bottle-neck.

And we’re done.

I wanted to make all these configuration variables as accessible as possible, so I made a special my.cnf with my recommended configuration (and useful comments that will help you fine-tune it for your tastes). If you'd like to get that file and also receive a nice eBook with these 17 variables discussed in depth (once it's done), leave your email address in the form below.

FINAL THOUGHTS

It's quite amazing how much you can improve things with small changes over the vanilla MySQL configuration. But you wanna know the real truth? The truth is we haven't even started!

While improving MySQL configuration can actually impact its performance greatly, you must have queries that are already tuned almost to perfection for that to be the case. And most of the time I find that that's exactly where the problem lies – in the queries.

Stay tuned and I will show you exactly the process we were using at Percona to identify which queries need to be fixed – find the ones that are occasionally (or constantly) slow and also the ones that impact the server utilisation the most. The easiest way for us to stay in touch is over the email, so if you'd like to hear from me on that and also if you'd like to receive the special my.cnf file I mentioned (with my recommended configuration, and useful comments that will help you fine-tune it for your tastes), please fill in the form below:

DOWNLOAD THE MY.CNF FILE

Your Name

Email Address

DOWNLOAD

thought you might enjoy this! Check it out when you have a chance: https://www.speedemy.

key-mysql-config-file-settings-mysql-5-7-proof/)

ABOUT THE AUTHOR

AURIMAS MIKALAUSKAS

Facebook (https://www.facebook.com/aurimas.mikalauskas)

Twitter (https://twitter.com/aurimas_m)

Google+ (https://plus.google.com/+AurimasMikalauskasPlus/posts)

Linux hacker since the age of 0xE, deeply passionate about high performance and scalability. Former architect of scalable systems at Percona - largest independent MySQL and MongoDB consultancy company (for 9 years). Audiobook addict, piano & guitar virtuoso, scuba diver, wind surfer and a wannabe glider pilot. Learn more about the author (http://www.speedemy.com/about/).

4 Comments SPEEDEMY 1 Login

Recommend 1 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Heril Muratovic • 5 months ago • edited

Is innodb_flush_method suitable for XFS file system? I'm using Centos 7 and I'm taking in consideration to use innodb_flush_method=O_DIRECT

• Reply • Share ›

Pedro Pilven • 2 years ago

Do you have reco related locks for systems reading and writting a lot ?

• Reply • Share ›

Pedro Pilven • 2 years ago

Thank you so much your post has been realy helpfull for me, my services runs 2 or 3 time faster since i apply your recommendations !
Thanks again, i'm looking forward to read more posts from you !

• Reply • Share ›

Sebastian Garcia • 2 years ago • edited

Thank you so much!

• Reply • Share ›

› FREE RESOURCES

MySQL Configuration Tuning Handbook
Covers MySQL 5.7
SPEEDEMY
MYSQL PERFORMANCE TUNING SERIES

MySQL Configuration Tuning Handbook (https://www.speedemy.com/books/mysql-configuration-tuning-ebook/)

RECENT POSTS

Best places to look for MySQL help (https://www.speedemy.com/best-places-look-mysql-help/)

How to create mysql login-path (https://www.speedemy.com/create-mysql-login-path/)

Inspecting MySQL micro-stalls with pt-stalk (https://www.speedemy.com/inspecting-mysql-micro-stalls-with-pt-stalk/)

Advanced MySQL Slow Query Logging Part 3: fine-tuning the logging process (https://www.speedemy.com/advanced-mysql-slow-query-logging-3/)

11 new features coming in MySQL 8.0 that will make your eyebrows raise (https://www.speedemy.com/new-in-mysql-8-0-dr/)

CATEGORIES

Apache (https://www.speedemy.com/category/apache/)

Architecture (https://www.speedemy.com/category/architecture/)

Elasticsearch (https://www.speedemy.com/category/elasticsearch/)

Full Text Search (https://www.speedemy.com/category/full-text-search/)

MySQL (https://www.speedemy.com/category/mysql/)

Web Servers (https://www.speedemy.com/category/web-servers/)

Search