InnoDB Logs

Vol. 1

By Stephane Combaudon and Vadim Tkachenko

About this Percona eBook: InnoDB is the default transactional storage engine for MySQL and the most important and broadly useful engine overall. It was designed for processing many short-lived transactions that usually complete rather than being rolled back. This Percona eBook focuses on one basic area: InnoDB logs.

In Chapter 1, Stephane Combaudon shares is experience in MySQL performance audits, listing the top 10 MySQL settings to tune after installation - many of which include InnoDB settings. In Chapter 2 explains why choosing a good InnoDB log file size is key to InnoDB write performance. Vadim Tkachenko, in Chapter 3, details the expected recovery time from 8G innodb_log_file.

Table of Contents

About Percona

Percona was founded in August 2006 by Peter Zaitsev and Vadim Tkachenko and now employs a global network of experts with a staff of more than 100 people. Our customer list is large and diverse, including Fortune 50 firms, popular websites, and small startups. We have over 1,800 customers and, although we do not reveal all of their names, chances are we're working with every large MySQL user you've heard about. To put Percona's MySQL expertise to work for you, please contact us.

Contact Us 24 Hours A Day

Is this an emergency? Get immediate assistance from Percona Support 24/7. Click here

Skype: oncall.percona
GTalk: oncall@percona.com
AIM (AOL Instant Messenger): oncallpercona
Telephone direct-to-engineer: +1-877-862-4316 or
UK Toll Free: +44-800-088-5561
Telephone to live operator: +1-888-488-8556
Customer portal: https://customers.percona.com/

Sales North America	(888) 316-9775 or (208) 473-2904
Sales Europe	+44-208-133-0309 (UK) 0-800-051-8984 (UK) 0-800-181-0665 (GER)
Training	(855) 55TRAIN or (925) 271-5054

10 MySQL settings to tune after installation

By Stephane Combaudon

When we are hired for a MySQL performance audit, we are expected to review the MySQL configuration and to suggest improvements. Many people are surprised because in most cases, we only suggest to change a few settings even though hundreds of options are available. The goal of this post is to give you a list of some of the most critical settings.

We already made such suggestions a few years ago on the MySQL Performance Blog, but things have changed a lot in the MySQL world since then! So this chapter will pick up where we left off and get you up to date.

Before we start…

Even experienced people can make mistakes that can cause a lot of trouble. So before blindly applying the recommendations of this post, please keep in mind the following items:

- Change one setting at a time! This is the only way to estimate if a change is beneficial.
- Most settings can be changed at runtime with SET GLOBAL. It is very handy and it allows you to quickly revert the change if it creates any problem. But in the end, you want the setting to be adjusted permanently in the configuration file.
- A change in the configuration is not visible even after a MySQL restart? Did you use the correct configuration file? Did you put the setting in the right section? (all settings in this post belong to the [mysqld] section)
- The server refuses to start after a change: did you use the correct unit? For instance, innodb_buffer_pool_size should be set in bytes while max_connection is dimensionless.
- Do not allow duplicate settings in the configuration file. If you want to keep track of the changes, use version control.
- Don’t do naive math, like “my new server has 2x RAM, I’ll just make all the values 2x the previous ones”.

Basic settings

Here are 3 settings that you should always look at. If you do not, you are very likely to run into problems very quickly.

innodb_buffer_pool_size: this is the #1 setting to look at for any installation using InnoDB. The buffer pool is where data and indexes are cached: having it as large as possible will ensure you use memory and not disks for most read operations. Typical values are 5-6GB (8GB RAM), 20-25GB (32GB RAM), 100-120GB (128GB RAM).

innodb_log_file_size: this is the size of the redo logs. The redo logs are used to make sure writes

are fast and durable and also during crash recovery. Up to MySQL 5.1, it was hard to adjust, as you wanted both large redo logs for good performance and small redo logs for fast crash recovery. Fortunately crash recovery performance has improved a lot since MySQL 5.5 so you can now have good write performance and fast crash recovery. Until MySQL 5.5 the total redo log size was limited to 4GB (the default is to have 2 log files). This has been lifted in MySQL 5.6.

Starting with innodb_log_file_size = 512M (giving 1GB of redo logs) should give you plenty of room for writes. If you know your application is write-intensive and you are using MySQL 5.6, you can start with innodb_log_file_size = 4G.

max_connections: if you are often facing the 'Too many connections' error, max_connections is too low. It is very frequent that because the application does not close connections to the database correctly, you need much more than the default 151 connections. The main drawback of high values for max_connections (like 1000 or more) is that the server will become unresponsive if for any reason it has to run 1000 or more active transactions. Using a connection pool at the application level or a thread pool at the MySQL level can help here.

InnoDB settings

InnoDB has been the default storage engine since MySQL 5.5 and it is much more frequently used than any other storage engine. That's why it should be configured carefully.

innodb_file_per_table: this setting will tell InnoDB if it should store data and indexes in the shared tablespace (innodb_file_per_table = OFF) or in a separate .ibd file for each table (innodb_file_per_table= ON). Having a file per table allows you to reclaim space when dropping, truncating or rebuilding a table. It is also needed for some advanced features such as compression. However it does not provide any performance benefit. The main scenario when you do NOT want file per table is when you have a very high number of tables (say 10k+).

With MySQL 5.6, the default value is ON so you have nothing to do in most cases. For previous versions, you should set it to ON prior to loading data as it has an effect on newly created tables only.

innodb_flush_log_at_trx_commit: the default setting of 1 means that InnoDB is fully ACID compliant. It is the best value when your primary concern is data safety, for instance on a master. However it can have a significant overhead on systems with slow disks because of the extra fsyncs that are needed to flush each change to the redo logs. Setting it to 2 is a bit less reliable because committed transactions will be flushed to the redo logs only once a second, but that can be acceptable on some situations for a master and that is definitely a good value for a replica. 0 is even faster but you are more likely to lose some data in case of a crash: it is only a good value for a replica.

innodb_flush_method: this setting controls how data and logs are flushed to disk. Popular values are O_DIRECT when you have a hardware RAID controller with a battery-protected write-back cache and fdatasync (default value) for most other scenarios. sysbench is a good tool to help you choose between the 2 values.

innodb_log_buffer_size: this is the size of the buffer for transactions that have not been committed yet. The default value (1MB) is usually fine but as soon as you have transactions with large blob/text fields, the buffer can fill up very quickly and trigger extra I/O load. Look at the Innodb_log_waits status variable and if it is not 0, increase innodb_log_buffer_size.

Other settings

query_cache_size: the query cache is a well known bottleneck that can be seen even when concurrency is moderate. The best option is to disable it from day 1 by setting query_cache_size = 0 (now the default on MySQL 5.6) and to use other ways to speed up read queries: good indexing, adding replicas to spread the read load or using an external cache (memcache or redis for instance). If you have already built your application with the query cache enabled and if you have never noticed any problem, the query cache may be beneficial for you. So you should be cautious if you decide to disable it.

log_bin: enabling binary logging is mandatory if you want the server to act as a replication master. If so, don't forget to also set server_id to a unique value. It is also useful for a single server when you want to be able to do point-in-time recovery: restore your latest backup and apply the binary logs. Once created, binary log files are kept forever. So if you do not want to run out of disk space, you should either purge old files with PURGE BINARY LOGS or set expire_logs_days to specify after how many days the logs will be automatically purged.

Binary logging however is not free, so if you do not need for instance on a replica that is not a master, it is recommended to keep it disabled.

skip_name_resolve: when a client connects, the server will perform hostname resolution, and when DNS is slow, establishing the connection will become slow as well. It is therefore recommended to start the server with skip-name-resolve to disable all DNS lookups. The only limitation is that the GRANT statements must then use IP addresses only, so be careful when adding this setting to an existing system.

Conclusion

There are of course other settings that can make a difference depending on your workload or your hardware: low memory and fast disks, high concurrency, write-intensive workloads for instance are cases when you will need specific tuning. However the goal here is to allow you to quickly get a sane MySQL configuration without spending too much time on changing non-essential MySQL settings or on reading documentation to understand which settings do matter to you.

Measuring the amount of writes in InnoDB redo logs

By Stephane Combaudon

Choosing a good InnoDB log file size is key to InnoDB write performance. This can be done by measuring the amount of writes in the redo logs. You can find a detailed explanation in this post.

To sum up, here are the main points:

- The redo logs should be large enough to store at most an hour of logs at peak-time
- You can either use the LSN in the SHOW ENGINE INNODB STATUS OUTPUT or the Innodb_os_log_written global status variable (if you are a Percona Server user, the LSN is also given by the Innodb_lsn_current status variable)

While reviewing the recommendation I made for a customer, one of my colleagues told me I was wrong in my redo log size calculations. After each one double checked the calculations, it turned out that we experienced something not expected:

- Using Innodb_os_log_written, I found that around 7.15 GB of redo logs were written per hour
- Using the LSN, my colleague found 2.70 GB/hour (almost a 3x difference!)

Something was obviously wrong in our understanding of how to measure the amount of writes in the redo logs. Let's first have a look at what the documentation says. It states that

- Innodb_os_log_written is *the number of bytes written to the log file*
- The LSN is an *arbitrary, ever-increasing value [that] represents a point in time corresponding to operations recorded in the redo log*

What is not obvious from the documentation is that while Innodb_os_log_written is incremented when the log file is written, the LSN is incremented when the log buffer is written.

This is interesting. It means that the durability setting can skew the results: if innodb_flush_log_at_trx_commit is set to 0, you can accidentally omit or add 1 second of write activity. Of course if you measure variations over 60s, this will not explain a 3x difference with the LSN. It also means that if the write workload is very non uniform, you can easily get very different numbers if you are not taking measures exactly at the same time for the 2 methods.

However, the write workload had not so much variance in my case. I also ran a test with a constant write workload (a mono-threaded script that inserts one row at a time in a table, as fast as it can) and I ended up with the same result: numbers were very different between the 2 methods. Even stranger, the innodb_os_log_written method consistently gave higher numbers than the LSN method, when we would have expected the opposite.

It was time for digging into the source code. All the credits should actually be given to Alexey Kopytov, who not only took the time to read the code again and to make tests, but who also caught something we all missed: writing to the redo logs and increasing the LSN have completely different logics.

The LSN simply shows the byte offset, so when you write 100 bytes to the log buffer, the LSN is increased by 100.
Writing to the redo logs is a much more complicated process: every write is a 512-byte write and there can be overlapping writes. Not clear? Let's look at an example when innodb_flush_log_at_trx_commit is set to 1 or 2 (again, thanks Alexey):

- Transaction 1 writes 100 bytes to the log buffer
- At commit, InnoDB writes a 512-byte block at offset xxx and increments Innodb_os_log_written by 512 bytes
- Transaction 2 writes 200 bytes to the log buffer
- At commit, InnoDB appends those 200 bytes to the same log block and overwrites the same 512-byte file block at offset xxx, then increases Innodb_os_log_written by another 512 bytes

At this point, the LSN has increased by 300 and Innodb_os_log_written by 1024 (a 3x difference!). This means that the documentation is correct: Innodb_os_log_written is the number of bytes written to the redo logs. But it does not reflect the growth of the redo logs.

So when you are trying to size the redo logs, looking at the LSN variations is a much better approximation than looking at the Innodb_os_log_written variations, which can be significantly far from the reality. However keep in mind that even the LSN is an approximate metric: if your write workload is non uniform and your sampling interval too short, you may well underestimate or overestimate the growth of your redo logs.

How long is recovery from 8G innodb_log_file

By Vadim Tkachenko

In my previous posts on the MySQL Performance Blog I highlighted that one of improvements in Percona Server is support of **innodb_log_file_size** > 4G. This test was done using Percona Server 5.5.7, but the same performance expected for InnoDB-plugin and MySQL 5.5.

The valid question how long is recovery in this case, so let's test it. I took the same tpcc-mysql 1000W workload with 52GB and 144GB **innodb_buffer_pool_size** with data located on Virident tachIOn card and killed mysqld after 30 mins of work.

The recovery time after start is:
for 52GB **innodb_buffer_pool_size**:

```
101220 21:54:31  InnoDB: Database was not shut down normally!
..
101220 22:02:40  InnoDB: Rollback of non-prepared transactions completed
```

that is 7min 51sec

for 144GB **innodb_buffer_pool_size**:

```
101220 22:45:37  InnoDB: Database was not shut down normally!

..
101220 22:55:00  InnoDB: Rollback of non-prepared transactions completed
```

that is 9min 23sec

and
for 144GB **innodb_buffer_pool_size** with data stored on RAID10:

```
101220 23:46:01  InnoDB: Database was not shut down normally!
..
101221  0:00:58  InnoDB: Rollback of non-prepared transactions completed
```

that is 14min 57sec

I think this time is acceptable as trade-off for performance.

However it should be taken into account if you use HA solution like DRBD, as it basically means this is time for fail-over period, and your system will be down during this time.

About the authors

Stéphane Combaudon joined Percona in July 2012, after working as a MySQL DBA for leading French companies such as Dailymotion and France Telecom.

In real life, he lives in Paris with his wife and their twin daughters. When not in front of a computer or not spending time with his family, he likes playing chess and hiking.

Vadim Tkachenko, co-founder and CTO, leads Percona's development group. He is an expert in LAMP performance, especially optimizing MySQL and InnoDB internals to take full advantage of modern hardware using his multi-threaded programming background. Source code patches authored by Vadim have been incorporated by Oracle Corporation and its predecessors into the mainstream MySQL and InnoDB products.

Vadim co-founded Percona in 2006 after four years in the High Performance Group within the official MySQL Support Team. He serves on Percona's Executive Team. He also co-authored High Performance MySQL, 3rd Edition with Percona's Peter Zaitsev and Baron Schwartz. Previously he founded a web development company in his native Ukraine. He now lives in California with his wife and their two children.