



PALOMINO DB

Configuring MySQL for Optimal Performance

Presented by:
Sheeri K. Cabral
Database Operations Manager, PalominoDB
www.palomionodb.com @sheeri

04/12/11

<http://bit.ly/iiKjoJ>

- Ask how many are using Windows
- Ask how many are using in production
- Ask how many aren't using
- Ask how many are using 5.1 vs. 5.5 vs older

For Optimal Performance

- Server tuning
- Schema optimization
- Query tuning



Server tuning is mostly variables, and that's what we're going through here today.



Are you using an old version?

Many bugs are changed in each version. Show some pages and explain how to upgrade

You can substitute manual pages with 5.1 if you're on 5.1. If you're on 5.0 or earlier **DEFINITELY UPGRADE.**

Directories

- basedir
- datadir
- tmpdir
 - /tmp



Basedir = installation directory

`$basedir/bin`

`$basedir/data` is default datadir

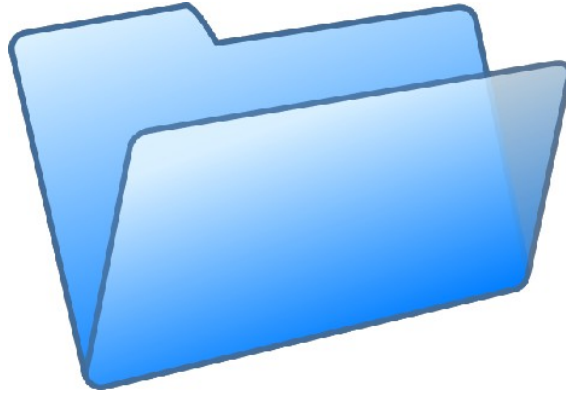
Datadir – where data is kept by default, innodb data files by default, binary logs by default.

Tmpdir – temporary files opened by mysql, like for replication or temporary intermediate files (not temporary for alter table, or those are in `$datadir`). Faster if local, be careful of memory-backed, because replication depends on it.

If `TMPDIR` is not set, MySQL uses the system default, which is usually `/tmp`, `/var/tmp`, or `/usr/tmp` on Unix, and on windows, in order `TMPDIR`, `TEMP`, and `TMP` environment variables, or finally Windows system default, which is usually `C:\windows\temp\`.

The `--tmpdir` option can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (":") on Unix and semicolon characters (";") on Windows.

What is a MySQL Database?



It's just a directory!

MySQL Files

- my.cnf / my.ini config
- Per-database
 - db.opt
 - .TRN .TRG
- .frm
- Log information



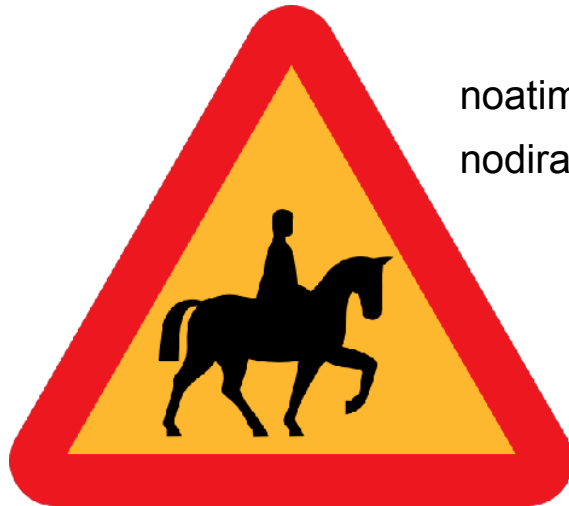
db.opt, one per database, has default charset and collation info for the database in \$datadir

.TRG one per table, .TRN one per trigger. Can have up to 6 triggers per table. .TRG is read on every DML, references any TRN that needs to be done.

.frm files have the table structure, one per table, so if you have thousands of tables this can be a problem.

The master.info and relay-log.info files are written to a LOT.

Mounting



noatime
nodiratime

7



PALOMINODB Proven Database Excellence

Otherwise access time for file and directories is saved

Ext3 supports both, so does XFS

Nobarrier?

ReiserFS (notail)

Does not affect last modified time

Storage Engine Files

- InnoDB
 - ibdata, .ibd
 - iblogfiles
- MyISAM
 - .MYI
 - .MYD



CSV has .frm and .CSM .CSV

Blackhole has .frm only

Archive has .frm and .ARV

Others may have different layouts all together, like
TokuDB, which has a directory per table, but still a
.frm file.

MySQL Variables

- `system_variable`
 - `SHOW GLOBAL|SESSION VARIABLES [LIKE...]`
 - `SELECT @@global|session.varname`
- `Status_variable`
 - `SHOW GLOBAL|SESSION STATUS [LIKE...]`



Note caps vs. no caps

Show variables and status in demo

```
mysql> SHOW GLOBAL STATUS LIKE 'Opened_tables';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Opened_tables | 45    |
+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> SHOW SESSION STATUS LIKE 'Opened_tables';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Opened_tables | 0     |
+-----+-----+
1 row in set (0.00 sec)
```



Show variables and status in demo

Then show these GLOBAL variables:

table_open_cache

Open_tables

File Descriptors

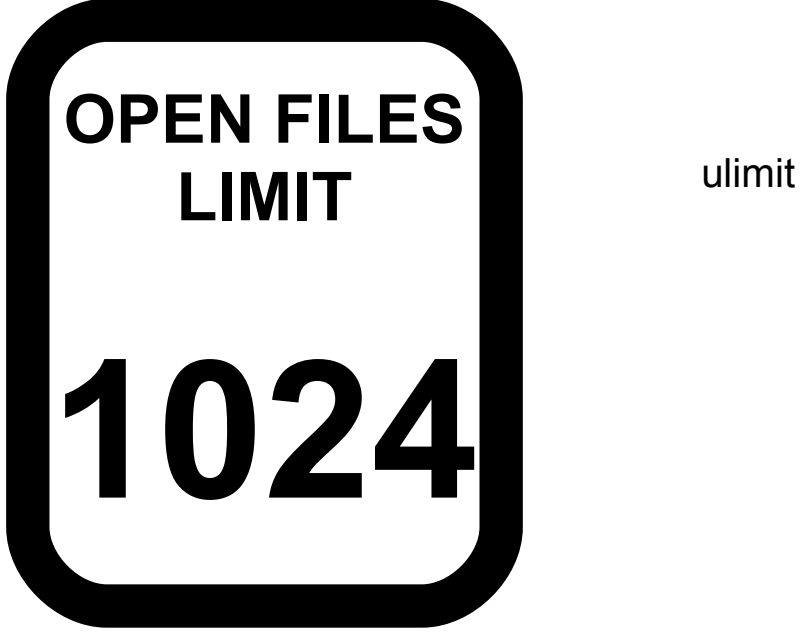
- `table_open_cache`
- `Open_tables`
- `Opened_tables`



`Open_tables` is how many are currently open

`Opened` is how many opened since opened. `FLUSH STATUS` does not clear this value.

if `Opened_tables` increases rapidly, the `table_open_cache` should be increased (if possible).




OPEN FILES
LIMIT

1024

ulimit

12

 PALOMINODB Proven Database Excellence

File descriptor usage is one of the reasons that `mysqld_safe` is started as root and `mysqld` is spawned from that, so it inherits the limits.

File Descriptor Usage

- `Open_files`
- `Opened_files`
- `open_files_limit`



`open_files_limit` can be set, but the server-reported value is the # of files the OS allows MySQL to open. If 0, MySQL can't set this value.

Not really anything to optimize here, just knowledge of how many files have used a certain library call (`my_open`) to open a file.

`Open_files` shouldn't be close to `open_files_limit`



DELAYED keyword is ignored on InnoDB; basically just a way to get around table-level locking on MyISAM tables. This is not durable ("D" in ACID compliant)

INSERT DELAYED will batch-write INSERTS once a certain # are accumulated. This goes faster, but has many implications.

INSERT DELAYED

- `delayed_insert_limit`
- `delayed_insert_timeout`
- `delayed_queue_size`
- `max_delayed_threads`
 - `Delayed_insert_threads`



`delayed_insert_limit`: When doing a bulk insert from `INSERT DELAYED`, process any `SELECT`s after this many records have been inserted (default 100)

`delayed_insert_timeout`: How long the handler waits for another `INSERT DELAYED` before terminating (default 300, seconds)

`delayed_queue_size`: Per-table queue size for `INSERT DELAYED`. If this fills, an `INSERT DELAYED` statement will “hang”. (default 1000)

`max_delayed_threads`: Max # of threads to handle `INSERT DELAYED`. If all threads are in use, further `INSERT DELAYED` statements ignore `DELAYED`. Thus, 0 disables `DELAYED`.

`max_insert_delayed_threads` – same as `max_delayed_threads`

`Delayed_insert_threads`: How many handler threads are in use

INSERT DELAYED

- Delayed_writes
- Delayed_errors
- Not_flushed_delayed_rows



DELAYED keyword is ignored on InnoDB; basically just a way to get around table-level locking on MyISAM tables. This is not durable (“D” in ACID compliant)

max_delayed_threads 20 by default

Delayed_insert_threads – how many threads are currently running

_writes, _errors are counters (sum)

Last is # rows waiting to be written. This would be what you'd lose in a crash!

InnoDB Flags

- `innodb_file_per_table`
- `innodb_flush_log_at_trx_commit`
- `innodb_fast_shutdown`



InnoDB is the ACID compliant transactional db

Can make it not ACID compliant!

`innodb_flush_log_at_trx_commit`

0 = log buffer written once per second and flushed

1 = log buffer written at each commit, flush to disk

2 = log buffer written at each commit, flush to disk once per second

`fast_shutdown`

0 = full purge, insert buffer merge before shutdown. can take a long time.

1 = skips at shutdown

2 = flushes logs and shuts down cold, as if a crash. No commit txn will be lost, but crash recovery will be done on startup.

InnoDB Buffer Pool

- innodb_buffer_pool_size
 - innodb data/index cache
- Innodb_page_size
- Innodb_buffer_pool_pages_total
 - Innodb_buffer_pool_pages_data
 - Innodb_buffer_pool_pages_free
 - Innodb_buffer_pool_pages_misc



misc pages are pages used for admin overhead, ie row locks & adaptive hash

Dirty Pages

- innodb_max_dirty_pages_pct
- Innodb_buffer_pool_wait_free
- Innodb_buffer_pool_pages_dirty



- innodb_max_dirty_pages_pct
 - after this % is reached, wait while dirty pages are flushed to disk
- Innodb_buffer_pool_wait_free
wait free is a counter for the # of times the buffer pool had to wait for pages to be freed by cleanup
- Innodb_buffer_pool_pages_dirty
 - Included in the pages_data
 -
 -

InnoDB Concurrency Variables

- innodb_thread_concurrency
- innodb_commit_concurrency
- innodb_concurrency_tickets
- innodb_sync_spin_loops



innodb_thread_concurrency – Max value=1000. 0 disables the checking of thread concurrency. After x OS threads are in innodb, any more are put into a waiting queue. (16)

innodb_commit_concurrency – 0 is for “unlimited” (default=0?)

innodb_concurrency_tickets – # times a thread can enter InnoDB without having to be queued if the threads exceed the value of innodb_thread_concurrency (500)

sync_spin_loops – how many times to wait for a mutex to be freed before suspending the thread (20)

InnoDB Log Basics

- innodb_log_file_size
- innodb_log_files_in_group
 - innodb_mirrored_log_groups
- innodb_log_group_home_dir



InnoDB I/O

- `innodb_io_capacity`
- `innodb_read_io_threads`
- `innodb_write_io_threads`



(InnoDB Plugin only) The maximum number of I/O operations per second that InnoDB will perform. This variable can be set at server startup, which enables higher values to be selected for systems capable of higher I/O rates. Having a higher I/O rate can help the server handle a higher rate of row changes because it may be able to increase dirty-page flushing, deleted-row removal, and application of changes to the insert buffer. The default value of `innodb_io_capacity` is 200. In general, you can increase the value as a function of the number of drives used for InnoDB I/O.

The ability to raise the I/O limit should be especially beneficial on platforms that support many IOPS. For example, systems that use multiple disks or solid-state disks for InnoDB are likely to benefit from the ability to control this parameter.

`read_io` & `write_io` threads = 4 by default.

InnoDB Logs

- innodb_log_buffer_size
 - Innodb_log_waits



innodb_log_buffer_size – “sensible values range from 1Mb – 8Mb” according to the manual. “A large log buffer allows large transactions to run without a need to write the log to disk before the transactions commit. Thus, if you have big transactions, making the log buffer larger saves disk I/O.”

If log_buffer_size too small, there will be Innodb_log_waits

Other InnoDB Variables

- `innodb_additional_mem_pool_size` (16Mb)
- `innodb_open_files` (300)
- `innodb_thread_sleep_delay` (10,000 = .01 sec)



`additional_mem_pool_size` – additional memory for things like the data dictionary. If this value is too small, warnings will be written to the error log and more memory will be allocated from the OS. Default is 1 Mb.

`innodb_open_files` – file descriptors for .ibd files. do not affect `table_open_cache`, variable is independent of other open files limits.

`innodb_thread_sleep_delay` – how long (microseconds) a thread waits before joining the InnoDB queue, 10,000 = 0.1 sec

Advanced InnoDB Variables

- `innodb_flush_method`
- `innodb_max_purge_lag`



`fsync()` by default to flush data files & logs

`O_DSYNC = O_SYNC` to open and flush logs, `fsync()` for data files.

`O_DIRECT` (some GNU/Linux versions, FreeBSD, and Solaris), `O_DIRECT` (or `directio()` on Solaris) to open the data files, `fsync()` to flush data files and logs.

This variable is relevant only for Unix. On Windows, the flush method is always `async_unbuffered` and cannot be changed.

`purge_lag` = if # of txns that UPDATE or DELETE is > this, then delay a while before proceeding. 0 means no delay ever.

InnoDB Status for Performance

- `Innodb_buffer_pool_read_ahead_rnd`
- `Innodb_buffer_pool_read_ahead_seq`



`Innodb_buffer_pool_read_ahead_rnd`:

Number of random read-aheads, for when a large part of the table is scanned, in random order

`Innodb_buffer_pool_read_ahead_seq`: Number of sequential read-aheads, for sequential full table scan

High rates of these are both bad, change queries.

Bad Handler Status

- Handler_read_first
- Handler_read_rnd
 - read_buffer_size
- Handler_read_rnd_next



Handler_read_first: Usually indicates full **index** scans

Handler_read_rnd: # requests to read a data row based on a fixed position; high if lots of sorting, full table scans, ie when joins aren't using keys

read_buffer_size - Memory allocated for each scan of each table that is done. Multiple of 4096.

Handler_read_rnd_next: read the next data row; indicative of full table scans, or otherwise not using indexes that exist.

Good Handler Status

- Handler_read_key
- Handler_read_next
- Handler_read_prev



Handler_read_key: # of read requests that use a key.

High = good

Handler_read_next: incremented for each row in an index scan or range query (not necessarily good or bad, just info)

Handler_read_prev: Mostly used in ORDER BY...DESC, same as Handler_read_next but for “previous”

Non-InnoDB Disk I/O

- flush
- flush_time



- flush (OFF)
 - Sync changes to disk after every SQL statement
 - If ON, write changes to disk; let OS handle sync
- flush_time (0)
 - Close tables and sync data to disk every x seconds
 - 0 = disabled
 - Enable for systems with very few resources

Query Cache

- query_cache_type
 - have_query_cache
- query_cache_size
 - Qcache_total_blocks
 - Qcache_free_blocks
 - Qcache_free_memory
- Qcache_queries_in_cache



Type ON, have_query_cache YES
query_cache_size 128M, total = 64,428, free=15,348,
free_mem=44.1M, queries in cache=20,536
for info: <http://dev.mysql.com/doc/refman/5.1/en/query-cache-configuration.html>

query_cache_limit max size of a resultset that can be cached.
approx memory size needed for each query cached is 3 blocks.

query_cache_size 0 disables the query cache; values are multiples of 1024. Amount of memory allocated for the query cache (even if query_cache_type is 0/OFF)

query_cache_type: 0/OFF means don't use the query cache (though the buffer is still created). 1/ON means cache all queries that can be cached (SELECT SQL_NO_CACHE can be used on individual queries not to cache), 2/DEMAND means only cache those that use SELECT SQL_CACHE.

Query Cache Usage

- `query_cache_limit`
- `Qcache_not_cached`
- `Qcache_lowmem_prunes`

31



PALOMINODB Proven Database Excellence

query_cache_limit

4194304

max size of a resultset that can be cached.

`Qcache_hits`

`Qcache_inserts`

`Qcache_lowmem_prunes`

- Defragment with `FLUSH QUERY CACHE`
- `query_cache_min_res_unit` (4096) can be decreased if results are very small

`Qcache_not_cached` – due to `SQL_NO_CACHE` or results bigger than `query_cache_limit`

Query Cache Usage

- Qcache_hits
- Qcache_inserts
- Com_select



com_select is not incremented when query cache is hit

query cache hit % = $\frac{\text{Qcache_hits}}{(\text{Qcache_hits} + \text{Com_select})} * 100$

Query Cache

Domas Mituzas' query cache tuning guide:

- <http://dom.as/tech/query-cache-tuner/>



Type ON, have `_query_cache` YES
`query_cache_size` 128M, total = 64,428, free=15,348,
free_mem=44.1M, queries in cache=20,536
for info: <http://dev.mysql.com/doc/refman/5.1/en/query-cache-configuration.html>

`query_cache_limit` max size of a resultset that can be cached.
approx memory size needed for each query cached is 3 blocks.

`query_cache_size` 0 disables the query cache; values are multiples of 1024. Amount of memory allocated for the query cache (even if `query_cache_type` is 0/OFF)

`query_cache_type`: 0/OFF means don't use the query cache (though the buffer is still created). 1/ON means cache all queries that can be cached (SELECT SQL_NO_CACHE can be used on individual queries not to cache), 2/DEMAND means only cache those that use SELECT SQL_CACHE.

Timeouts

- back_log
- net_read_timeout
 - net_retry_count
- net_write_timeout



back_log=size of TCP listen queue, how many outstanding requests MySQL can have before it stops answering new requests (50)

net_read_timeout, net_write_timeout – only for TCP connections (30,10?)

net_retry_count is for interrupted read connections (60).

More Timeouts

- `connect_timeout`
- `wait_timeout`
- `interactive_timeout`



Increasing the `connect_timeout` value might help if clients frequently encounter errors of the form Lost connection to MySQL server at 'XXX', system error: errno. If this is too low, `Aborted_connects` status variable will be higher (but is not the only reason)

`wait_timeout` is how long to wait before killing sleeping non-interactive timeouts.

10 for `connect_timeout`

Wait and interactive timeout default is 28800, 8h.

Threads

- Threads_created
- Threads_cached
 - thread_cache_size



threads created, if this is rapidly increasing, increase thread_cache_size. Cache miss rate = threads_created/Connections

Threads_cached = # threads currently in thread cache

When a thread is slow to launch

- `slow_launch_time`
- `Slow_launch_threads`



`slow_launch_threads` is incremented if it takes longer than `slow_launch_time` seconds to launch a thread. `thread_cache` should be in play anyway. This isn't really a timeout. I've never seen a value >0 of `slow_launch_threads`

MyISAM Key Cache

- Can have more than one
 - CACHE INDEX IN
- key_buffer_size
- key_cache_block_size



key_buffer_size, key_cache_block_size can be set per named cache using SET GLOBAL cachename.variable=value, ie

```
SET GLOBAL session.key_buffer_size=10240;
```

MyISAM Key Cache

- Uses LRU; hot/warm sub-chains
- `key_cache_age_threshold`
- `key_cache_division_limit`



`key_cache_division_limit` and `key_cache_age_threshold` can be set per named cache using `SET GLOBAL cachename.variable=value`.

`age_threshold` = how fast something gets demoted from “hot” to “warm” sub-chain

`division_limit` – % of key cache to use in warm (vs. hot) sub-chain of the cache

MyISAM Key Cache Sizing

- Percentage used for all key caches:
 - $1 - (\text{Key_blocks_unused} * \text{key_cache_block_size}) / \text{key_buffer_size}$
- Key_blocks_used
- Key_blocks_not_flushed



- Key_blocks_used
 - max used at any time

MyISAM Key Cache Efficiency

- Cache miss %
 - $\text{Key_reads (from disk)} / \text{Key_read_requests} * 100$
- Key_write_requests
- Key_writes



This slide and the next slide were switched in the slide decks you have.

MyISAM Key Cache Efficiency

- Cache miss %
 - $\text{Key_reads (from disk)} / \text{Key_read_requests} * 100$
 - $34 / 254122 * 100 = 0.01 \%$
- Key_write_requests (10)
- Key_writes (10)



Logs

- log_output (FILE, TABLE)



General Log

- general_log
 - log
- sql_log_off (session)



SET SESSION sql_log_off=ON by a user with the SUPER privilege to not log anything to the general log. Off by default.

Slow Query Log

- `slow_query_log`
 - `log_slow_queries`
- `slow_query_log_file`
- `Slow_queries`



What gets logged as a slow query

- `long_query_time`
- `log_queries_not_using_indexes`
- `min_examined_row_limit`



`long_query_time` in seconds, can be fractional or 0.

Error Logging

- log_error
- log_warnings (1)
 - sql_notes (ON)



sql_notes is equivalent to log_warnings, except for notes, and it's a session variable only. On by default

Binary Logging

- log_bin
 - sql_log_bin, sql_log_update
- max_binlog_size
- binlog_format
- expire_logs_days



binlog_format

- **mysqlbinlog --base64-output=DECODE-ROWS**
-
-
- **SET SESSION sql_log_bin=OFF** if you don't want to log the current session to the binary log. For example, FLUSH commands, making data changes that shouldn't replicate (ie, if sync'ing).

Binary Log Cache

- `binlog_cache_size`
- `Binlog_cache_use`
- `Binlog_cache_disk_use`



I don't think I've ever tuned this

Replication and Binary Logging

- `log_slow_slave_statements`
- `log_slave_updates`
- `sync_binlog`
- `sync_frm`



`server_id` is the unique ID of the server; a slave does not apply binary log entries with its own ID. This is how infinite loops are avoided in replication (though you can make one happen accidentally if you change the `server_id` on a slave that is not caught up).

`sync_binlog` will use `fdatasync` every `x` writes to the binary log. Slowest choice (if battery-backed write cache, not as slow), but also safest. Default is 0, which means rely on the OS to flush to disk.

`sync_frm` set to ON means that non-temporary tables have their `.frm` file sync'd to disk with `fdatasync` on table create.

Temporary Tables

- Per-thread
 - tmp_table_size
 - max_heap_table_size
- Created_tmp_tables
- Created_tmp_disk_tables



if created_tmp_disk_tables is too big, maybe increase tmp_table_size & max_heap_table_size

Memory Settings

(not storage engine dependent)

- `join_buffer_size`
- `read_rnd_buffer_size`
- `max_prepared_stmt_count`
- `preload_buffer_size`



`read_rnd_buffer_size` - Per-client buffer used in sorting when an index is present. Larger values can improve ORDER BY and GROUP BY, but it's per-client, so be careful. Best to increase this value by session, not globally, if you can.

`join_buffer_size` = 1 buffer for each FULL join of 2 tables. a 3-way join has to join buffers. This is ONLY used if there are no indexes on one table and a full table scan has to be done.

`max_prepared_stmt_count` - Limit is so a DOS can't occur, 0 disables prepared statements. Prepared statements take up memory.

`preload_buffer_size` (32768)
Buffer size allocated when pre-loading indexes

Memory Settings

(not storage engine dependent)

- `query_alloc_block_size`
- `query_prealloc_size`
- `thread_stack`



`query_alloc_block_size` = size of memory allocation for objects during parse and execute query stages. If memory is fragmented, increasing this can help

`query_prealloc_size` = The size of the persistent buffer used for statement parsing and execution. This buffer is not freed between statements. If you are running complex queries, a larger `query_prealloc_size` value might be helpful in improving performance, because it can reduce the need for the server to perform memory allocation during query execution operations.

Minimum size of persistent buffer for parse and execute query stages. Persistent, so a larger value may improve performance if there are frequent memory allocations.

`thread_stack` = Per-thread stack size

Table Definition Cache

- `table_definition_cache`
- `Open_table_definitions`
- `Opened_table_definitions`



`table_definition_cache` – larger value speeds up the time it takes to open a table; does not use file descriptors and takes up less space than `table_open_cache`. (Slide 44 has `table_open_cache`)

Usually large “Opened” value means you need to increase the cache, or actually changing table definitions.

Sorting Status Variables

- Sort_range
- Sort_rows
- Sort_scan



Sort_range = # sorts done using ranges

Sort_rows = # sorted rows

Sort_scan = # sorts done using full table scan

Sorting System Variables

- `sort_buffer_size`
 - `Sort_merge_passes`
- `max_sort_length`



`Sort_merge_passes` = # of sort merge passes. If large, consider increasing `sort_buffer_size`

`max_sort_length` = maximum for TEXT/BLOB types

Join Buffer

- join_buffer_size
- Select_full_join
- Select_scan



Select_full_join = #joins that did full table scans b/c they didn't use indexes.

Select_scan = # joins that did a full table scan on the first table (not both tables). High # is bad, not as bad as Select_full_join, but still not so good...find in slow query log using "log queries not using indexes". (I think).

Joining

- max_join_size
- sql_big_selects
- Select_range_check



max_join_size = if more than this many rows need to be examined, don't allow this statement to proceed. Basically this tries to avoid runaway queries.

sql_big_selects = SELECT statements > max_join_size are aborted if set to 0. 1 by default, which is all selects are allowed. If max_join_size is changed from the default, sql_big_selects is automatically set to 0. If max_join_size is then changed (ie, in session) then sql_big_selects is ignored.

Select_range_check = # joins w/out keys that check for key usage after each row. Worry if >0, check indexes in tables.

Join Status Variables

- `Select_full_range_join`
- `Select_range`



`Select_full_range_join` = # joins that used a range search (good, uses indexes).

`Select_range` = # joins that used a range search on the 1st table only (good, uses indexes). The 2nd table may have used exact match, no index, etc. so it's hard to tell if that means the 2nd table is bad or not.

Optimizer

- optimizer_prune_level
- optimizer_search_depth
- optimizer_switch



optimizer_prune_level (1)

0 disables, so exhaustive search. 1, plans are pruned based on # of rows retrieved by intermediate plans.

optimizer_search_depth (62)

max search depth of optimizer. If ># tables in the query, slower to find the plan but gets a better plan. If <# tables in the query, quick to find a plan but it may be suboptimal. If 0, system picks a “reasonable” value. If set to max tables in query +2, uses 5.0 algorithm.

optimizer_switch (5.1.34, not PFM)

Globally or per-session, can set index_merge={on|off},
index_merge_intersection={on|off},
index_merge_union={on|off},
index_merge_sort_union={on|off}

Optimization

- `max_seeks_for_key`
- `range_alloc_block_size`
- `sql_select_limit`
- `Last_query_cost`



`max_seeks_for_key` = Limit assumed max number of seeks when looking up rows based on an index. The MySQL optimizer assumes that no more than this number of key seeks are required when searching for matching rows in a table by scanning an index, regardless of the actual cardinality of the index. By setting this to a low value (say, 100), you can force MySQL to prefer indexes instead of table scans.

`range_alloc_block_size` = block size allocated when range optimization is done

`sql_select_limit` = Max # of rows to return from a SELECT query (not applied to SELECTs within stored procedures, or SELECT queries where result set is returned, as in CREATE TABLE...SELECT and INSERT INTO...SELECT)

`Last_query_cost` = 0 for “complex” like UNION and subquery, because it can't be calculated appropriately. Session variable, default 0

Locking Status Variables

- Table_locks_immediate
- Table_locks_waited
- Com_lock_tables
- Com_unlock_tables



Table_locks_immediate

times a table lock request was granted immediately

Table_locks_waited

times a table lock request had to wait (big # is a problem)

Com_lock_tables

Com_unlock_tables

Locking System Variables

- `sql_buffer_result`
- `low_priority_updates`
 - `sql_low_priority_updates`
- `max_write_lock_count`



`sql_buffer_result` (OFF)

If ON, forces results to be stored in temp tables, releases locks earlier than if not enabled. Session variable only. Useful for, say, if it takes a long time for a result to be sent to a client.

`low_priority_updates` (OFF) If on, writes are lower priority than reads for table-level locking tables. ie, MyISAM, not InnoDB. called `sql_low_priority_updates` before, both show up in `SHOW GLOBAL VARIABLES` currently.

`max_write_lock_count` (4,294,967,295)

After this many write locks, allow a read lock. Basically by default writes are higher priority than reads, but if you want x writes to happen, then allow a read, set this variable.

MyISAM-specific

- `concurrent_insert`
- `delay_key_write`
- `keep_files_on_create`



ALSO - `(sql_)low_priority_updates` from slide 100

`concurrent_insert`. 1=default, `concurrent_insert` if no fragmentation. 2, `concurrent_insert` even with fragmentation. 0 = no concurrent insert.

`delay_key_write` ON=default, key buffer not flushed for tables created with `DELAY_KEY_WRITE` keyword. Other values = OFF and ALL, which treats all MyISAM tables as having `DELAY_KEY_WRITE` on.

`keep_files_on_create` = overwrite existing MYI and MYD files when a `.frm` file isn't present. Set to on or always explicitly use `INDEX_DIRECTORY` and `DATA_DIRECTORY` to not overwrite MYI and MYD files. If a `.frm` file is present, `mysqld` returns an error that the table exists.

MyISAM-specific

- `myisam_data_pointer_size`
- `myisam_stats_method`
- `myisam_use_mmap`



`myisam_data_pointer_size` = The default pointer size in bytes, to be used by CREATE TABLE for MyISAM tables when no MAX_ROWS option is specified. This variable cannot be less than 2 or larger than 7. The default value is 6. A value of 4 allows tables up to 4GB; a value of 6 allows tables up to 256TB. If you get “Table is full” you may need to increase this, or set MAX_ROWS when doing CREATE TABLE.

`myisam_stats_method` = How the server treats NULL values when collecting stats about distribution of index values for MyISAM tables. Values are: **nulls_equal** all NULL index values are considered equal and form a single value group that has a size equal to the number of NULL values

nulls_unequal NULL values are considered unequal, each NULL forms a distinct value group of size 1.

nulls_ignored, NULL values are ignored.

`myisam_use_mmap` = added in 5.1.4, use memory mapping for reading/writing MyISAM tables.

Repairing MyISAM Tables

- `myisam_recover_options`
- `myisam_repair_threads`



`recover_options=`recovery mode. Values = any combination of the values of `DEFAULT`, `BACKUP`, `FORCE`, or `QUICK`, separated by commas. Specifying with no argument is the same as specifying `DEFAULT`, and specifying with an explicit value of "" disables recovery (same as not giving the option). If recovery is enabled, each time `mysqld` opens a MyISAM table, it checks whether the table is marked as crashed or wasn't closed properly. If this is the case, `mysqld` runs a check on the table. If the table was corrupted, `mysqld` attempts repair.

The following options affect how the repair works:

BACKUP If the data file was changed during recovery, save a backup of the `tbl_name.MYD` file as `tbl_name-datetime.BAK`.

FORCE Run recovery even if we would lose more than one row from the `.MYD` file.

QUICK = Don't check the rows in the table if there aren't any delete blocks.

DEFAULT = Recovery w/out backup, forcing, quick checking.

Before the server automatically repairs a table, it writes a note about the repair to the error log. If you want to be able to recover from most problems without user intervention, you should use the options `BACKUP,FORCE`. This forces a repair of a table even if some rows would be deleted, but it keeps the old data file as a backup so that you can later examine what happened.

multi-threaded repair by sorting is beta-quality

MyISAM Index Sorting

- Done with REPAIR TABLE, ALTER TABLE, LOAD DATA INFILE, CREATE INDEX
- `myisam_sort_buffer_size`
- `myisam_max_sort_file_size`



`myisam_sort_buffer_size` = buffer size for sorting indexes during REPAIR TABLE or adding a new index to an existing table w/data. max is 4Gb before 5.1.23, 16,384 petabytes 5.1.23 and later, assuming 64-bit. 64-bit windows limit is 4Gb no matter what version, larger values are truncated.

`myisam_max_sort_file_size` = max temp file when re-creating a MyISAM index during repair, ALTER TABLE or LOAD DATA INFILE. If bigger than this, index is created using `key_cache` instead, which takes longer.

Aborted connects/clients

- Aborted_clients
- Aborted_connects
 - max_connect_errors
 - connect_timeout)



Aborted_clients = client left w/out properly closing.
Usually due to not closing a db handler (in code, or by someone stopping code before the close occurred).

Aborted_connects = client could not connect – wrong username/password, wrong permissions, connect_timeout reached (can indicate DNS problems), or wrong info in connection packet.

Profiling

- `have_community_features`
- `profiling`
 - `per-session`
- `profiling_history_size`



`have_community_features` – profiling was a community contributed feature...

`profiling` = Whether or not query profiling is turned on. Per-session only. See <http://dev.mysql.com/doc/refman/5.1/en/show-profiles.html>

`profiling_history_size` = How many queries to keep profiling information for. Rotated on a FIFO basis.

SQL Behavior

- `sql_auto_is_null`
- `sql_mode`
- `sql_quote_show_create`



`sql_auto_is_null`= if on, `SELECT * FROM tbl WHERE autoincrement_col IS NULL` will return the last inserted row (similar to `SELECT LAST_INSERT_ID`)

`sql_mode` - very long to explain, my book has 6 pages on it!

`sql_quote_show_create` = `SHOW CREATE TABLE` statements are quoted so identifiers are escaped

SQL Behavior

- `sql_safe_updates`
 - also called `i-am-a-dummy`
- `updatable_views_with_limit`
- `new`
- `old`



`sql_safe_updates` = updates or deletes with no limit xor no where clause using an index are not allowed if this is set to ON.

`updatable_views_with_limit` = YES by default, meaning that an update is allowed if it has a LIMIT to an updatable view that does not contain a unique key in its definition. If set to NO, updates with LIMIT are not allowed on these types of views.

`new` = used in 4.0 to turn on 4.1 behaviors

`old` = changes default scope of index hints. In general we don't recommend index hints, so this shouldn't be a problem. (index hints with no FOR clause apply only to how indexes are used for row retrieval and not to resolution of ORDER BY or GROUP BY clauses). Added in 5.1.17 as `old_mode`, changed to `old` in 5.1.18, this is how index hints are used before 5.1.17.

Variables That Are Not Useful

- `big_tables`
 - `sql_big_tables`
- `old_alter_table`
- `pseudo_thread_id`



`big_tables` (OFF) was a session-only variable, and saved all intermediate sets to files. Prevents most “table is full” errors for intermediate tables. This functionality is automatic in MySQL 3.23.2 and up, using memory for temp tables and switching to disk tables when the intermediate set is large enough.

`old_alter_table` – see

<http://www.pythian.com/news/2963>

It’s used to disable the optimizations that were added in 5.1 for “Faster Alter Table”

`pseudo_thread_id`

“ This variable is for internal server use. “

Not used/Not useful

- table_lock_wait_timeout
- rpl_recovery_rank
- Rpl_status



table_lock_wait_timeout and rpl_recovery_rank – not used

Rpl_status

“ The status of fail-safe replication (not yet implemented). “

These variables are not used

- time_format (%H:%i:%s)
- date_format (%Y-%m-%d)
- datetime_format (%Y-%m-%d %H:%i:%s)
- innodb_file_io_threads
 - used on Windows only



Mostly Unused Variables

- `log_tc_size`
 - not shown in `SHOW GLOBAL VARIABLES`
- `Tc_log_max_pages_used`
- `Tc_log_page_size`
- `Tc_log_page_waits`



`log-tc-size` – size of the memory-mapped implementation of the log that is used by mysql when it acts as the transaction coordinator for recovery of internal XA transactions.

`Tc_log_max_pages_used` = high water mark for # pages used in the log.

If the product of `Tc_log_max_pages_used` and `Tc_log_page_size` is always significantly less than the log size, `log-tc-size` can be reduced. Currently, this variable is unused: It is unneeded for binary log-based recovery, and the memory-mapped recovery log method is not used unless the number of storage engines capable of two-phase commit is greater than one. (InnoDB is the only applicable engine.)

`Tc_log_page_size` x 0

`Tc_log_page_waits` x 5 – not sure why there are waits when this is theoretically unused?

Percona Server (5.1)

- XtraDB instead of InnoDB
- Somewhat transparent
- Patched MySQL
- http://www.percona.com/docs/wiki/percona-server:features:indexes:variable_reference



Lots of features

I will only go over the performance ones, though, and not touch on things that are off by default.

Will also only go over 5.1 Percona features, not 5.5, otherwise it's just too much.

Like:

- information_schema tables that show things like response time for queries, more stats in general, like indexes used.
- make InnoDB more reliable after a crash
- ability to export/import physical backups of one table, like myisam.

innodb_fast_recovery



77



PALOMINODB Proven Database Excellence

No overhead to this change

This change is simple. If the new page's `oldest_modification` is,,,

[newer than any `oldest_modification` in flushlist]

add to first of the `flush_list`

[older than any `oldest_modification` in flushlist]

add to last of the `flush_list`

[else]

overwrite `oldest_modification` by the oldest `oldest_modification` in
`flush_list`

add to last of the `flush_list`

innodb_read_ahead



This variable controls the read-ahead algorithm of InnoDB. The following values are available:

'none': disables read-ahead

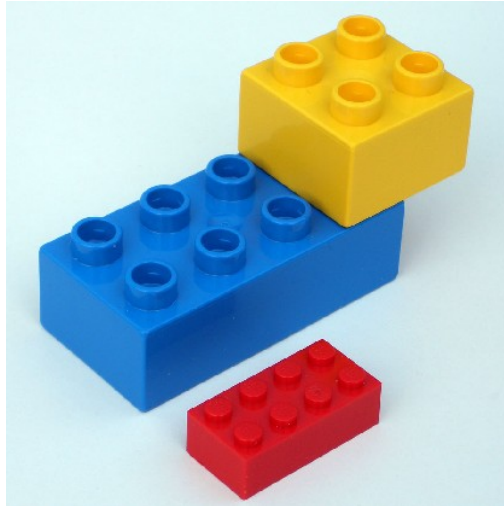
'linear' (default): if enough pages within the same extent are accessed sequentially, InnoDB will automatically fetch the remaining pages

'both': enable both 'random' and 'linear' algorithms.

You can also control the threshold from which InnoDB will perform a read ahead request with the `innodb_read_ahead_threshold` variable (a regular innodb variable)

`_threshold`: (InnoDB Plugin only) Controls the sensitivity of linear read-ahead that InnoDB uses to prefetch pages into the buffer cache. If InnoDB reads at least `innodb_read_ahead_threshold` pages sequentially from an extent (64 pages), it initiates an asynchronous read for the entire following extent. The permissible range of values is 0 to 64. The default is 56: InnoDB must read at least 56 pages sequentially from an extent to initiate an asynchronous read for the following extent.

innodb_log_block_size



This variable changes the size of transaction log records. The default size of 512 bytes is good in most situations. However, setting it to 4096 may be a good optimization with SSD cards. While settings other than 512 and 4096 are possible, as a practical matter these are really the only two that it makes sense to use.

Next slide also touches on some ssd stuff.

Checkpointing

- innodb_adaptive_checkpoint
 - none
 - reflex
 - estimate
 - keep_average



Default is none ≤ 1.05 , estimate ≥ 1.06

InnoDB constantly flushes dirty blocks from the buffer pool. Normally, the checkpoint is done passively at the current oldest page modification (this is called “fuzzy checkpointing”). When the checkpoint age nears the maximum checkpoint age (determined by the total length of all transaction log files), InnoDB tries to keep the checkpoint age away from the maximum by flushing many dirty blocks. But, if there are many updates per second and many blocks have almost the same modification age, the huge number of flushes can cause stalls.

Adaptive checkpointing forces a constant flushing activity at a rate of approximately $[\text{modified age} / \text{maximum checkpoint age}]$. This can avoid or soften the impact of stalls caused by aggressive flushing.

*** 'reflex': This behavior is similar to `innodb_max_dirty_pages_pct` flushing. The difference is that this method starts flushing blocks constantly and contiguously based on the oldest modified age. If the age exceeds 1/2 of the maximum age capacity, InnoDB starts weak contiguous flushing. If the age exceeds 3/4, InnoDB starts strong flushing. The strength can be adjusted by the MySQL variable `innodb_io_capacity`. In other words, we must tune `innodb_io_capacity` for the 'reflex' method to work the best.

*** 'estimate': If the oldest modified age exceeds 1/2 of the maximum age capacity, InnoDB starts flushing blocks every second. The number of blocks flushed is determined by $[\text{number of modified blocks}]$, $[\text{LSN progress speed}]$ and $[\text{average age of all modified blocks}]$. So, this behavior is independent of the `innodb_io_capacity` variable.

*** 'keep_average': This method attempts to keep the I/O rate constant by using a much shorter loop cycle (0.1 second) than that of the other methods (1.0 second). It is designed for use with SSD cards.

!: In some cases `innodb_adaptive_checkpoint` needs larger transaction log files (`innodb_adaptive_checkpoint` makes the limit of modified age lower). So, doubling the length of the transaction log files may be safe.

innodb_checkpoint_age_target



Adaptive checkpointing forces a constant flushing activity at a rate of approximately $[\text{modified age} / \text{maximum checkpoint age}]$. This can avoid or soften the impact of stalls caused by aggressive flushing.

This variable controls the maximum value of the checkpoint age if its value is different from 0. If the value is equal to 0, it has no effect.

It is not needed to shrink `innodb_log_file_size` to tune recovery time.

Flushing



Flushing

- innodb_adaptive_flushing
- innodb_flush_method
- innodb_flush_log_at_trx_commit_session
- innodb_flush_neighbor_pages



innodb_adaptive_flushing – default false for innodb, true for xtradb. This is an existing InnoDB variable used to attempt flushing dirty pages in a way that avoids I/O bursts at checkpoints. In XtraDB, the default value of the variable is changed from that in InnoDB.

innodb_flush_method – also in innodb,

innodb_flush_method - The following values are allowed:

'fdatasync': use fsync() to flush both the data and log files.

'O_SYNC': use O_SYNC to open and flush the log files; use fsync() to flush the data files.

'O_DIRECT': use O_DIRECT (or directio() on Solaris) to open the data files; use fsync() to flush both the data and log files.

'ALL_O_DIRECT': use O_DIRECT open and flush both the data and the log files. This value was added in Percona Server release 5.1.54-12.5.

innodb_flush_log_at_trx_commit_session – like innodb_flush_log_at_trx_commit, but a session variable. 0/1/2 are same as innodb_flush_log_at_trx_commit, so you can set it to 1 globally and 2 for certain transactions. A value of 3 means “just use the global value”.

innodb_flush_neighbor_pages – default 1

This variable specifies whether, when the dirty pages are flushed to the data file, the neighbor pages in the data file are also flushed at the same time or not. The following values are available:

0: disables the feature

1 (default): enables the feature

If you use a storage which has no “head seek delay” (e.g. SSD or enough memory for write buffering), 0 may show better performance.

innodb_lazy_drop_table



Default value is off

When `innodb_file_per_table` is set to 1, doing a `DROP TABLE` can take a long time on servers with a large buffer pool, even on an empty InnoDB table. This is because InnoDB has to scan through the buffer pool to purge pages that belong to the corresponding tablespace. Furthermore, no other queries can start while that scan is in progress.

When `innodb_lazy_drop_table` is ON, XtraDB optimizes that process by only marking the pages corresponding to the tablespace being deleted. It defers the actual work of evicting those pages until it needs to find some free pages in the buffer pool.

When `innodb_lazy_drop_table` is OFF, the usual behavior for dropping tables is in effect.



Default =0, >0 is number of purge threads. >1 is experimental, so if you need it, set it to 1.

Purge of the undo space is periodically done by the InnoDB main thread
 In most cases for an OLTP application, the transactions are small and short-running so the undo space can fit in memory in the buffer pool. The purge is then quick and efficient.

But there are several reasons that can make the undo space grow very large and go to disk:

- long-running transactions
- transactions with lots of changes
- too many updates for the purge process to keep up

In all cases performance will drop dramatically. In standard InnoDB it is difficult to find an efficient solution for this problem.

You can now have one or several threads dedicated to the purge. This feature provides several benefits:

- more control over the purge process
- more stable performance (no more performance drops)
- the InnoDB main thread does not need to take care of the purge anymore

But be aware that this feature comes at a cost: it reduces the overall performance because purging adds a non-negligible overhead. However we think it is better to have slightly worse but stable performance over time than to have better peak performance but unpredictable sharp drops.

innodb_dict_size_limit



- http://www.percona.com/docs/wiki/percona-server:features:innodb_dict_size_limit



If your data dictionary is taking up more than a gigabyte or so of memory, you may benefit from this feature. A data dictionary of this size normally occurs when you have many tens of thousands of tables. For servers on which tables are accessed little by little over a significant portion of time, memory usage will grow steadily over time, as if there is a memory leak. For servers that access every table fairly soon after being started, memory usage will increase quickly and then stabilize.

On standard InnoDB, the size of the data dictionary depends on the number and size of tables opened in the server. Once a table is opened, it is never removed from the data dictionary unless you drop the table or you restart the server. In some cases, the data dictionary grows extremely large. If this consumes enough memory, the server will begin to use virtual memory. Use of virtual memory can cause swapping, and swapping can cause severe performance degradation. By providing a way to set an upper limit to the amount of memory the data dictionary can occupy, this feature provides users a way to create a more predictable and controllable situation.

Link at bottom is how to figure out if you need this, and how to size

InnoDB Insert Buffer

- innodb_ibuf_accel_rate
- innodb_ibuf_active_contract
- innodb_ibuf_max_size



Increasing the value of `innodb_ibuf_accel_rate` increases insert buffer activity.

Default Value 100 Range 100 - 999999999

Each time the background insert buffer thread is called, its activity is altered by the value of both `innodb_io_capacity` and `innodb_ibuf_accel_rate` this way :

$$[\text{real activity}] = [\text{default activity}] * (\text{innodb_io_capacity}/100) * (\text{innodb_ibuf_accel_rate}/100)$$

`active_contract`: This variable specifies whether the insert buffer can be processed before it reaches its maximum size. The following values are allowed:

0: the insert buffer is not processed until it is full. This is the standard InnoDB behavior.

1: the insert buffer can be processed even it is not full. (1 by default)

`ibuf_max_size` - maximum size of the insert buffer. By default the insert buffer is half the size of the buffer pool so if you have a very large buffer pool, the insert buffer will be very large too and you may want to restrict its size with this variable.

Setting this variable to 0 is equivalent to disabling the insert buffer. But then all changes to secondary indexes will be performed synchronously = performance degradation. Likewise a too small value can hurt performance. If you have very fast storage (ie storage with RAM-level speed, not just a RAID with fast disks), a value of a few MB may be the best choice for maximum performance.

Buffer Pool



Buffer pool is data and indexes in memory. When you stop MySQL, buffer pool is lost.

There are 2 ways to keep the buffer pool information across a restart: doing an export/import of the buffer pool or the more fiddly storing the buffer pool in shared memory.

Buffer Pool Export/Import

- innodb_auto_lru_dump

Before shutdown:

```
mysql> select * from information_schema.XTRADB_ADMIN_COMMAND /*!  
XTRA_LRU_DUMP*/;
```

```
+-----+  
| result_message |  
+-----+  
| XTRA_LRU_DUMP was succeeded. |  
+-----+  
1 row in set (0.02 sec)
```

After startup:

```
mysql> select * from information_schema.XTRADB_ADMIN_COMMAND /*!  
XTRA_LRU_RESTORE*/;
```

```
+-----+  
| result_message |  
+-----+  
| XTRA_LRU_RESTORE was succeeded. |  
+-----+  
1 row in set (0.62 sec)
```

89



PALOMINODB Proven Database Excellence

Buffer pool uses least recently used algorithm , hence “LRU” export/import can still be done manually even if innodb_auto_lru_dump is off (off by default)

Manual:

Before shutdown:

```
mysql> select * from information_schema.XTRADB_ADMIN_COMMAND /*!  
XTRA_LRU_DUMP*/;
```

```
| XTRA_LRU_DUMP was succeeded. |
```

After startup:

```
mysql> select * from information_schema.XTRADB_ADMIN_COMMAND /*!  
XTRA_LRU_RESTORE*/;
```

```
| XTRA_LRU_RESTORE was succeeded. |
```

This special feature of Percona Server enables the buffer pool to be restored to its pre-shutdown state in a matter of minutes.

The feature works as follows. The buffer pool is a list of pages, usually 16kb in size, which are identified by an 8-byte number kept in LRU. The mechanism is to save the list of 8-byte page numbers just before shutdown, and after restart, to read the pages from disk and insert them back into the LRU at the correct position. The pages are sorted by ID to avoid random I/O, which is slower than sequential I/O on most disks. The LRU list is saved to the file `ib_lru_dump` in the directory specified by the `datadir` configuration setting, so you can back it up and restore it with the rest of your data easily.

Note that this feature does not store the contents of the buffer pool (i.e. it does not write 1GB of data to disk if you have a 1GB buffer pool). It stores only the identifiers of the pages in the buffer pool, which is a very small amount of data even for large buffer pools.

This feature can be used both manually and automatically. It is safe to enable automatically, and we have not found any performance regressions in it.

Shared Memory Buffer Pool

90



PALOMINODB Proven Database Excellence

This is a way to save the buffer pool in shared memory so that when MySQL starts up again it can use it.

In order to reuse the buffer pool stored in shared memory, InnoDB must shut down cleanly before the server is restarted, otherwise the shared memory segment should be removed.

Restrictions required in order to use the buffer pool in shared memory are as follows:

- The InnoDB executable cannot be changed between restarts.

- The value of `innodb_page_size` can't be changed between restarts.

- The value of `innodb_buffer_pool_size` can't be changed between restarts. If it is changed, an error will be reported, e.g.:

InnoDB: Error: `srv_buf_pool_size` is different (shm=85899345920 current=75161927680).

For most errors resulting from the use of the feature, the solution is to manually remove/reinitialize the shared memory segment. There is also no provision for removing the shared memory segment automatically when it is no longer needed to store the buffer pool. This must also be done by manually removing the shared memory segment, using `ipcs` to find the memory segment and `ipcrm` to delete it.

Segment Size Too Large

One InnoDB message you may see is:

Warning: Failed to allocate 88165400576 bytes. (new) errno 22

This may be because the size of the segment you are trying to allocate is exceeding SHMAX. You can see the current value of SHMAX by doing:

```
cat /proc/sys/kernel/shmmax
```

If that is the problem, increase the value of SHMAX so that is large enough, e.g., by doing:

```
echo 137438953472 > /proc/sys/kernel/shmmax
```

Shared Memory Buffer Pool

- `innodb_buffer_pool_shm_key`
- `innodb_buffer_pool_shm_checksum`



If `innodb_buffer_pool_shm_key` is non-zero when InnoDB restarts, InnoDB tries to reconnect and reuse the existing buffer pool in shared memory. InnoDB will not start with an error. If an error occurs on startup, the shared memory segment will also need to be removed.

Zero by default. If the value is non-zero, it specifies the key of the shared memory segment in which to store the buffer pool.

The range of `innodb_buffer_pool_shm_key` is system dependent, from zero to usually the maximum value of an UNSIGNED INTEGER. It is an input parameter to the `shmget` system function. For details, see your system IPC manual.

Checksum validation of the shared memory buffer pool is performed at startup and shutdown when `innodb_buffer_pool_shm_checksum` is enabled. It is enabled by default. Startup and shutdown are slower when checksum validation is enabled, but enabling it adds additional protection against the shared memory region becoming corrupted.

Very Fine Tuning



Write-heavy workloads



Reduce Statistics Overhead

- `innodb_stats_auto_update`
- `innodb_stats_update_need_lock`
- `innodb_use_sys_stats_table`



`innodb_stats_auto_update`

(default 1) - InnoDB updates the each index statistics automatically (many updates were done, some information_schema is accessed, table monitor, etc...). Setting this option 0 can stop these automatic recalculation of the statistics except for “first open” and “ANALYZE TABLE command”.

`innodb_stats_update_need_lock`

(default 1) - If you meet contention of `&dict_operation_lock`, setting 0 reduces the contention. But 0 disables to update “Data_free:” of “show table status”.

`innodb_use_sys_stats_table`

(default OFF) - If this option is enabled, XtraDB uses the `SYS_STATS` system table to store statistics of table indexes. Also, when InnoDB opens a table for the first time, it loads the statistics from `SYS_STATS` instead of sampling index pages. If you use a high `stats_sample_pages` value, the first open of a table is expensive. In such a case, this option will help. Note: This option may cause less frequent updating of statistics. So, you should intentionally use the `ANALYZE TABLE` command more often.

(This variable was introduced in release 5.1.50-11.4.)

innodb_extra_rsegments



Some write-intensive workloads on boxes with many CPUs have scalability problems. The contention is caused by the rollback segment, which is single: all transactions are serialized when needing to access the segment. With this feature you can now create and use multiple segments (up to 256).

NOTE: This feature is incompatible with InnoDB. As long as a single rollback segment is used, there is no problem; the database can still be used by both XtraDB and InnoDB. However, creating multiple rollback segments will cause an internal format change to the system tablespace. Once multiple segments have been created, the database will no longer be compatible with InnoDB.

When you modify this variable, you must restart the MySQL server for the change to take effect. Please note that you must perform a slow shutdown (ie with `innodb_fast_shutdown = 0`). If you just perform a fast shutdown, the MySQL server will then restart without error but the additional segments will not be created.

Can check the `INNODB_RSEG` table if this is enabled.

innodb_fast_checksum



Write-heavy workloads

Off by default

InnoDB writes a checksum at the end of each data page in order to detect data files corruption. However computing this checksum requires CPU cycles and in some circumstances this extra overhead can become significant.

XtraDB can use a more CPU-efficient algorithm, based on 4-byte words, which can be beneficial for some workloads (for instance write-heavy workloads on servers that can perform lots of IO).

The original algorithm is checked after the new one, so you can have data pages with old checksums and data pages with new checksums. However in this case, you may experience slow reads from pages having old checksums. If you want to have the entire benefit of this change, you will to recreate all your InnoDB tables, for instance by dumping and reloading all InnoDB tables.

innodb_doublewrite_file



You can set a dedicated doublewrite file, but this is very advanced. Usually this is in the centralized ibdata1 file.

In usual workloads the performance impact is 5% or so. As a consequence, you should always enable the doublewrite buffer because the strong guarantee against data corruption is worth the small performance drop.

But if you experience a heavy workload, especially if your data does not fit in the buffer pool, the writes in the doublewrite buffer will compete against the random reads to access the disk. In this case, you can see a sharp performance drop compared to the same workload without the doublewrite buffer—a 30% performance degradation is not uncommon.

Another case when you can see a big performance impact is when the doublewrite buffer is full. Then new writes must wait until entries in the doublewrite buffer are freed.

That's it!

Questions? Comments? Feedback?

@sheeri
sheeri@palominodb.com

MySQL Administrator's Bible
OurSQL Podcast (www.oursqlcast.com)
<http://planet.mysql.com>

