## MySQL Performance Tuning Parameters

What is running slow on my server?

Using Top or Sar Command, we can find to dig the issue.

Why are my queries running slow?

proper maintenance of MySQL server, we can see in multiple options to fix these issues, in below you can find

How to fix slow running queries?

Rebuild, Reorg, etc.

## Server Level Parameters

1) InnoDB Parameters and Important Server Variables

## InnoDB Parameters and Important Variables

**Innodb\_buffer\_pool\_size**

All of the InnoDB server variables are GLOBAL in nature.

The single most important InnoDB configuration variable is the innodb\_buffer\_pool\_size.

Assuming the server is only running mysqld and most of your tables are InnoDB tables, the majority of your memory should be dedicated to the InnoDB buffer pool.

It is safe to begin at **70 – 80 percent** of your system memory allocated the InnoDB buffer, and adjust up or down as you need to.

If you have a large amount of RAM (16 Gb or more) on a dedicated MySQL server, then the buffer pool can be an even larger percentage of overall memory. When configuring a server, choose a starting value for the InnoDB buffer pool, set the other configuration values, then determine how much memory is still available.

**On Unix, the vmstat, top, and free commands show memory information.**

**In Windows, the Task Manager can show you memory usage.**

To determine if the InnoDB buffer pool is appropriately sized, run:

mysql> show variables like 'innodb\_buffer\_pool\_size';

+-------------------------+-----------+

| Variable\_name | Value |

+-------------------------+-----------+

| innodb\_buffer\_pool\_size | 134217728 |

+-------------------------+-----------+

**Innodb\_flush\_method**

mysql> show variables like 'innodb\_flush\_method';

+---------------------+-------+

| Variable\_name | Value |

+---------------------+-------+

| innodb\_flush\_method | |

+---------------------+-------+

1 row in set (0.00 sec)

This is static variable determines how the innodb storage engice interacts with the Operating system in simple words (IO Access Method) with respective I/0 operations.

**Innodb\_log\_buffer\_size**

mysql> show variables like 'innodb\_log\_buffer\_size';

+------------------------+----------+

| Variable\_name | Value |

+------------------------+----------+

| innodb\_log\_buffer\_size | 16777216 |

+------------------------+----------+

1 row in set (0.00 sec)

Buffer used for writing to the innodb logs, unless you use very large BLOB's this is static variable, this should not be over 8MB, and you can set to 2MB.

**Innodb\_log\_file\_size**

mysql> show variables like 'innodb\_log\_file\_size';

+----------------------+----------+

| Variable\_name | Value |

+----------------------+----------+

| innodb\_log\_file\_size | 50331648 |

+----------------------+----------+

This parameter shows log file size,its a static variable this determines the size of each innodb log file (id\_logfile).

**innodb\_log\_files\_in\_group**

mysql> show variables like 'innodb\_log\_files\_in\_group';

+---------------------------+-------+

| Variable\_name | Value |

+---------------------------+-------+

| innodb\_log\_files\_in\_group | 2 |

+---------------------------+-------+

This is static variable, this determines the total number of innodb log files

**Innodb\_max\_dirty\_pages\_pct**

mysql> show variables like 'innodb\_max\_dirty\_pages\_pct';

+----------------------------+-----------+

| Variable\_name | Value |

+----------------------------+-----------+

| innodb\_max\_dirty\_pages\_pct | 75.000000 |

+----------------------------+-----------+

1 row in set (0.00 sec)

This is a dynamic variable specifies the maximum percentage of pages in the innodb buffer pool that can be dirty this is, changed in the buffer pool in memory with out being saved to disk. defaults to 90% (based on Operating System)

**Innodb\_thread\_concurrency**

mysql> show variables like 'innodb\_thread\_concurrency';

+---------------------------+-------+

| Variable\_name | Value |

+---------------------------+-------+

| innodb\_thread\_concurrency | 0 |

+---------------------------+-------+

1 row in set (0.01 sec)

This is a dynamic variable determines the maximum number of system threads inside innoDB. A good number to start is twice the number of CPU's.

**Innodb\_flush\_log\_at\_trx\_commit**

mysql> show variables like 'innodb\_flush\_log\_at\_trx\_commit';

+--------------------------------+-------+

| Variable\_name | Value |

+--------------------------------+-------+

| innodb\_flush\_log\_at\_trx\_commit | 1 |

+--------------------------------+-------+

1 row in set (0.01 sec)

There are three possible options [ 0, 1, 2 ]. This dynamic system variable manages how often the innodb log buffer is written (flushed) to the log file.

For the greatest possible durability and consistency in a replication setup using InnoDB with transactions, use innodb\_flush\_log\_at\_trx\_commit=1 and sync\_binlog=1 in your master server my.cnf file.

* When it is set to 0, nothing is done on commit; rather the log buffer is written and flushed to the InnoDB redo log once a second. This gives better performance, but a server crash can erase the last second of transactions.
* **When it is set to 1, the log buffer is written to the InnoDB redo log file, and a flush to disk performed after each transaction. This is required for full ACID compliance.**
* With a setting of 2, logs are written after each transaction commit and flushed to disk once per second. only an operating system crash or a power outage can erase the last second of transactions. InnoDB's crash recovery works regardless of the value.

■ Innodb\_buffer\_pool\_pages\_data is the total number of used data pages (clean and dirty).

■ Innodb\_buffer\_pool\_pages\_dirty is the number of dirty data pages. The number of clean data pages can be calculated from these first two status variables.

■ Innodb\_buffer\_pool\_pages\_flushed is the number of data pages that have been flushed to disk.

■ Innodb\_buffer\_pool\_pages\_free is the number of unused data pages.

■ Innodb\_buffer\_pool\_pages\_misc is the number of data pages used for InnoDB overhead.

■ Innodb\_buffer\_pool\_pages\_total is the total number of pages.

Calculate the ratio of unused data pages to the total number of pages:

Innodb\_buffer\_pool\_pages\_free / Innodb\_buffer\_pool\_pages\_total

**If the ratio is high (close to 1), then the InnoDB buffer pool is probably set too high**. A less likely cause is that the innodb\_max\_dirty\_pages\_pct system variable is set too low, and dirty pages are being flushed very often, freeing up pages long before they are needed.

show variables LIKE '%innodb\_max\_dirty\_pages\_pct%';

Conversely, if the ratio is low, the size of the InnoDB buffer pool may need to be set higher. Using the information you have about the free memory on your system, increase the InnoDB buffer pool size, restart mysqld, and continue to monitor the status variables after the newly sized InnoDB buffer pool has been used for a while. Continue the adjust-monitor-adjust cycle,

and once your system is at the right level, continue to monitor levels, making sure to check performance once every month or two.

Make sure to always leave a buffer of a half-gigabyte or so of memory because mysqld performs very poorly when it is forced to use swap space. Keep in mind that under high load, mysqld will use more memory.

All these statements lead me to write this blog post to (try to) explain how InnoDB’s buffer pool works and what it contains.

**Buffer pool content**

InnoDB data is stored in **16 KB pages** (blocks), either on disk (*ibdata* files) or in memory (buffer pool). Each one of these pages may contain one or more row.

The buffer pool is basically a **cache** for these pages: once a page’s content is requested by a query, the page is cached in the buffer pool.

You may be wondering what kind of data is stored in these pages.

Short answer: **indexes** (on a size point of view).

MySQL offers a very useful **INFORMATION\_SCHEMA** database, which contains since version 5.5 a table named **innodb\_buffer\_page**.

This table holds 1 record (row) per page in the buffer pool, including interesting data such as what the page contains. Take a look at [MySQL’s official documentation of this table](http://dev.mysql.com/doc/refman/5.6/en/innodb-buffer-page-table.html) if you want details on it’s columns and data.

Now let’s have a little fun with this table.

**Number of pages in buffer pool, if you set to 200M**

**Query:**

|  |
| --- |
| mysql> select count(\*) from information\_schema.innodb\_buffer\_page;  +----------+  | count(\*) |  +----------+  | 16382 |  +----------+  1 row in set (0.03 sec) |
| select BLOCK\_ID,PAGE\_TYPE,TABLE\_NAME,INDEX\_NAME from information\_schema.innodb\_buffer\_page; |

**Output:**

|  |
| --- |
| +----------+  | count(\*) |  +----------+  | 16382 |  +----------+ |

We’ve got **163 82** pages in our buffer pool **×** 16 KB per page = **2 62 112 KB**. This value matches *innodb\_buffer\_pool\_size* parameter.

**Page types in buffer pool**

**Query:**

|  |
| --- |
| Select page\_type as Page\_Type,sum(data\_size)/1024/1024 as Size\_in\_MB  from information\_schema.innodb\_buffer\_page group by page\_type  order by Size\_in\_MB desc; |

**Result:**

|  |
| --- |
| +-------------------+--------------+  | Page\_Type         | Size\_in\_MB   |  +-------------------+--------------+  | INDEX             | 158.66378689 |  | UNKNOWN           | 0.00000000   |  | TRX\_SYSTEM        | 0.00000000   |  | SYSTEM            | 0.00000000   |  | FILE\_SPACE\_HEADER | 0.00000000   |  | IBUF\_BITMAP       | 0.00000000   |  | EXTENT\_DESCRIPTOR | 0.00000000   |  | ALLOCATED         | 0.00000000   |  | INODE             | 0.00000000   |  | BLOB              | 0.00000000   |  | UNDO\_LOG          | 0.00000000   |  | IBUF\_FREE\_LIST    | 0.00000000   |  | IBUF\_INDEX        | 0.00000000   |  +-------------------+--------------+ |

As you can see, merely **INDEX** pages are cached in the buffer pool.

**Buffer pool usage per index**

**Query:**

|  |
| --- |
| select  table\_name as Table\_Name, index\_name as Index\_Name,count(\*) as Page\_Count, sum(data\_size)/1024/1024 as Size\_in\_MB from information\_schema.innodb\_buffer\_page  group by table\_name, index\_name order by Size\_in\_MB desc; |

**Result:**

|  |
| --- |
| +--------------------------------------------+-----------------+------------+-------------+  | Table\_Name                                 | Index\_Name      | Page\_Count | Size\_in\_MB  |  +--------------------------------------------+-----------------+------------+-------------+  | `mobapps`.`core\_url\_rewrite`               | PRIMARY         |       2829 | 40.64266014 |  | `mobapps`.`core\_url\_rewrite`               | FK\_CORE\_UR |        680 |  6.67517281 |  | `mobapps`.`catalog\_product\_entity\_varchar` | PRIMARY         |        449 |  6.41064930 |  | `mobapps`.`catalog\_product\_index\_price`    | PRIMARY         |        440 |  6.29357910 |  | `mobapps`.`catalog\_product\_entity`         | PRIMARY         |        435 |  6.23898315 |  +--------------------------------------------+-----------------+------------+-------------+ |
|  |

mysql> SHOW VARIABLES LIKE 'wait\_timeout%';

+---------------+-------+

| Variable\_name | Value |

+---------------+-------+

| wait\_timeout | 28800 |

+---------------+-------+

1 row in set (0.00 sec)

By Default 28800 Sec means 8 hours, you can reduce to 2 hours or 1 hour avoid performance issues.

mysql> SHOW VARIABLES LIKE 'interactive\_timeout';

+---------------------+-------+

| Variable\_name | Value |

+---------------------+-------+

| interactive\_timeout | 28800 |

+---------------------+-------+

1 row in set (0.01 sec)

By Default 28800 Sec means 8 hours, you can reduce to 2 hours or 1 hour interactive connections this also will help to avoid performance issues.

mysql> SHOW VARIABLES LIKE 'max\_allowed\_packet';

+--------------------+---------+

| Variable\_name | Value |

+--------------------+---------+

| max\_allowed\_packet | 4194304 |

+--------------------+---------+

1 row in set (0.00 sec)

This will client will use to sent SQL Statements in the form of rows, to the MySQL server.

If you are using the **mysql** client program, its default max\_allowed\_packet variable is 16MB. To set a larger value.

But in real time the parameter will be supported upto 2 GB for some applications you can set with max\_allowed\_packet = 500M as well based your application usage and some applications you can set with (1073741824 Bytes – 1024M (1 GB)

innodb\_read\_io\_threads = 4

innodb\_write\_io\_threads = 4

mysql> show variables like 'innodb\_read\_io\_threads';

+------------------------+-------+

| Variable\_name | Value |

+------------------------+-------+

| innodb\_read\_io\_threads | 4 |

+------------------------+-------+

1 row in set (0.00 sec)

mysql> show variables like 'innodb\_write\_io\_threads';

+-------------------------+-------+

| Variable\_name | Value |

+-------------------------+-------+

| innodb\_write\_io\_threads | 4 |

+-------------------------+-------+

1 row in set (0.00 sec)

mysql>

show variables like

To enable multiple buffer pool instances, set the innodb\_buffer\_pool\_instances configuration option to a value greater than 1GB (the default) up to 64 (the maximum). This option takes effect only when you set innodb\_buffer\_pool\_size to a size of 1GB or more.

mysql> show variables like 'innodb\_buffer\_pool\_instances';

+------------------------------+-------+

| Variable\_name | Value |

+------------------------------+-------+

| innodb\_buffer\_pool\_instances | 1 |

+------------------------------+-------+

1 row in set (0.00 sec)

Dividing the buffer pool into separate instances can improve concurrency, by reducing contention as different threads read and write to cached pages. This feature is typically intended for systems with a buffer pool size in the multi-gigabyte range.

Multiple buffer pool instances are configured using the innodb\_buffer\_pool\_instances configuration option, and you might also adjust the innodb\_buffer\_pool\_size value.

The total size you specify is divided among all the buffer pools. For best efficiency, specify a combination of innodb\_buffer\_pool\_instances and innodb\_buffer\_pool\_size so that each buffer pool instance is at least 1GB.

select thd\_id,conn\_id,user,db,current\_statement,program\_name from sys.x$session;

select thd\_id,conn\_id,user,db,current\_statement,program\_name from sys.x$processlist;

-- MyISAM Key Cache

**MyISAM storage engine options**

The MyISAM engine is one of the oldest storage engines, **introduced in version 3.**2. Even so,

it is still in use by many production servers.

**MyISAM Configuration Options**

Concurrent\_insert - Allows for inserts into MyISAM tables without blocking reads.

Delayed\_key\_write - Delays flushing to disk for index changes to batch the changes for

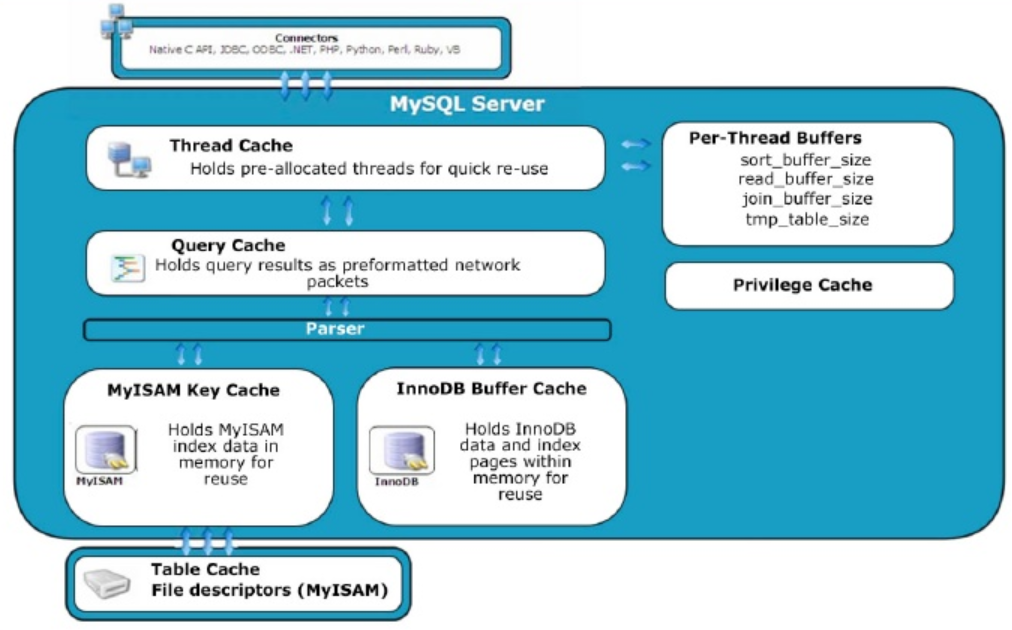
greater performance.

Key\_buffer\_size - This configures the size of the myisam index buffer

SHOW VARIABLES LIKE 'CONCURRENT\_INSERT';

SHOW VARIABLES LIKE 'DELAYED\_KEY\_WRITE';

SHOW VARIABLES LIKE 'KEY\_BUFFER\_SIZE';



These three server variables are GLOBAL in scope.

The concurrent\_inserts server variable defaults to a setting of 1 [ **AUTO** ]. With this setting, if there are any deleted rows in the data file that space is filled. Only if there are no deleted rows are inserts allowed to be added while reads occur at the same time.

With a setting of 2, MyISAM operates slightly differently. If no selects are occurring the empty space in the data file is filled.

If there are any reads, then the inserts are written to the end of the data file at the same time.

mysql> show variables like '%concurrent\_insert';

+-------------------+-------+

| Variable\_name | Value |

+-------------------+-------+

| concurrent\_insert | AUTO |

+-------------------+-------+

1 row in set (0.00 sec)

If the delay\_key\_write system variable is enabled the server will not immediately flush to disk changes to the MyISAM index buffer. Delaying the flushing operation allows for greater performance.

The tradeoff is that if you enable this setting, a server crash could cause corrupted indexes on disk. The data will probably be fine, it’s just the indexes that are affected. However, this index corruption may not be detectable automatically, so you need to make sure to rebuild the indexes using myisamcheck during server startup.

The default value is ON, which enables delayed writing for all MyISAM tables created with the **DELAYED\_KEY\_WRITE** option of CREATE TABLE. Using the ALL option enables delayed writes for all MyISAM tables, and the OFF option disables delayed key writing.

Rebuilding indexes for MyISAM tables takes time. **With very large tables it is going to take a significant amount. With the delayed\_key\_write setting, you are trading off performance for speed of recoverability and is something you should think about very carefully**.

ALTER TABLE DBName.TableName DELAY\_KEY\_WRITE = 1;

It is possible, using the **DELAY\_KEY\_WRITE** system variable, to force **DELAY\_KEY\_WRITE** behavior on for ALL MyISAM tables. This is done by setting the **DELAY\_KEY\_WRITE** system variable within the system configuration file to **ALL** instead of the default ON.

[mysqld]

Delay\_key\_write=ALL

As with most performance enhancement, there is usually a sacrifice to be made for speed. It is important to understand that data integrity, particularly the integrity of your table indexes are more corruptible when running with a the **DELAY\_KEY\_WRITE** feature enabled.

For this reason, it is important to have alternative means of warding off corruption with other solutions like the **–myisam-recover-options** feature to create a backup of indexes.

Another common practice with high-performance database server is using a secondary power source or battery backup, so services have time to shut down properly during a power loss scenario.

**The key\_buffer variable sets up a buffer for the MyISAM table indexes**. **MyISAM does not have a buffer for the data, so the only caching for data is done by the operating system buffers**.

When trying to determine how much memory to allocate to the key\_buffer, you should consider the server usage patterns. Are you using MyISAM for most tables? Do a lot of your queries use indexes?

Allocate some amount of memory — say, between 10 and 20 percent of the memory available to mysqld — and after some actual usage, see if you need to increase or decrease the key\_buffer.

**You can monitor key buffer usage by checking the cache hit ratio**

The cache hit ratio is the number of times a value in the buffer was

read as a percentage of the total number of times a value in the buffer was looked for. To determine

the cache hit ratio, run:

mysql> SHOW GLOBAL STATUS LIKE '%key\_read%';

+-------------------+-------+

| Variable\_name | Value |

+-------------------+-------+

| Key\_read\_requests | 25629 |

| Key\_reads | 66071 |

+-------------------+-------+

2 rows in set (0.00 sec)

And use the key\_reads and key\_read\_requests status variables in the following formula (a higher hit ratio is better):

hit\_ratio [Key\_Cache\_Miss\_Rate%] =(key\_reads/key\_read\_requests) \*100 To increase the hit ratio, use the key buffer option to set the key\_buffer\_size variable larger.

Need to increase the key\_buffer\_size, if we get below 1

**For overall MyISAM Index buffer usage**, you will need to determine the value of the status variable key\_blocks\_unused and the values of the system variables key\_cache\_block\_size and key\_buffer\_size by running:

SHOW GLOBAL VARIABLES LIKE 'key%';

SHOW GLOBAL STATUS LIKE 'key\_blocks\_u%';

Key\_blocks\_unused cache cluster unused (blocks) number, Key\_blocks\_used said once used the maximum blocks number  
Key\_blocks\_used / (Key\_blocks\_unused + Key\_blocks\_used) \* 100% ≈ 18%. [ideal value ≈ 80%]

From the value calculated for index buffer usage, look for at least 80 percent usage or you probably have too much memory allocated to the key buffer.

**Scripts:-**

**Calculate Key Cache Usage Rate**

select truncate((1-((@Key\_blocks\_unused\*@@key\_cache\_block\_size)/@@key\_buffer\_size))\*100, 2)

as "(1-((Key\_blocks\_unused × key\_cache\_block\_size) ÷ key\_buffer\_size)) × 100"\G

select @Key\_blocks\_unused as Key\_blocks\_unused,

@@key\_cache\_block\_size as key\_cache\_block\_size,

@@key\_buffer\_size as key\_buffer\_size\G

**Calculate the Ratio**

**Version 5.5 [ Now is Disabled as MySQL.5.7 ]**

select VARIABLE\_VALUE into @Key\_reads FROM **information\_schema.global\_status**

WHERE VARIABLE\_NAME = "Key\_reads"; [ Table is available but feature is disabled ]

select @Key\_reads as"Key\_reads (From Disk)",

@Key\_read\_requests as "Key\_read\_requests (Total)",

truncate(@Key\_reads/@Key\_read\_requests\*100, 2) as "Key\_Cache\_Miss\_Rate%"\G

**Version 5.5 [ Now is Disabled as MySQL.5.7 ]**

select VARIABLE\_VALUE into @Key\_read\_requests FROM **information\_schema.global\_status**

WHERE VARIABLE\_NAME = "Key\_read\_requests";