

# THE MONITIS INTERNAL MYSQL SERVER MONITORING

## INTRODUCTION

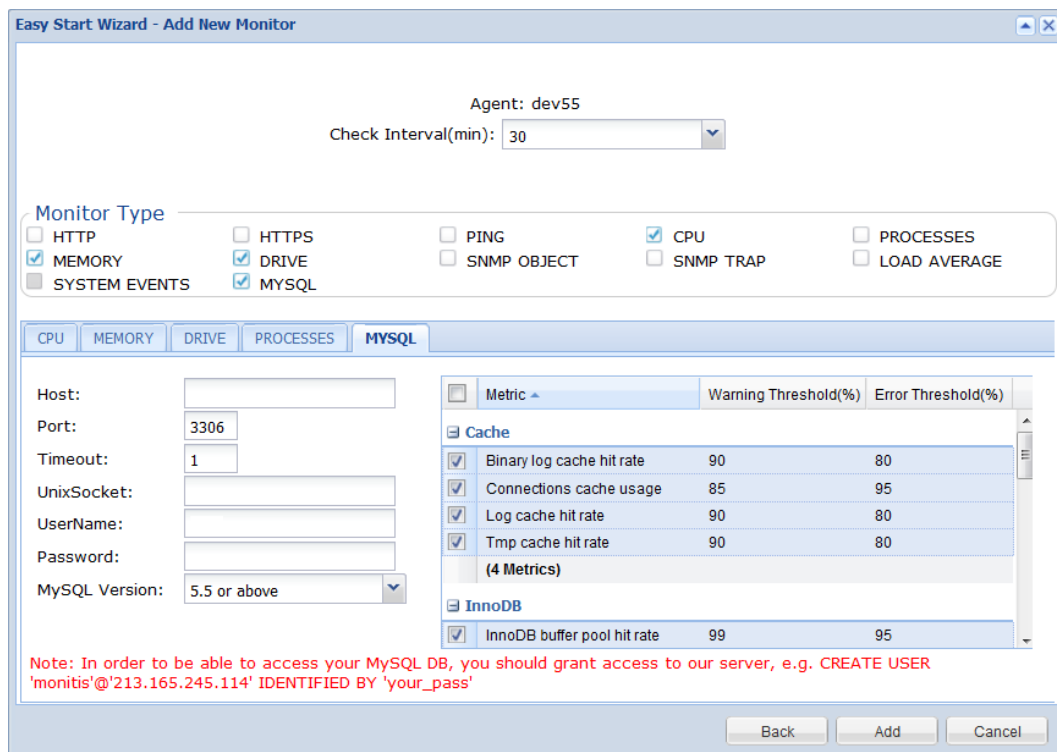
The MySQL database is a crucial part of a wide variety of products, particularly web applications (M in LAMP stands for MySQL). Naturally, it is very important to monitor the health status of MySQL. However, there is constant disagreement on which of the many MySQL status variables provide the best overview on MySQL health status and indicate that something is not right with a server.

It certainly depends on what your application does – tuning read performance is different than optimizing write operations and everything changes when you have a cluster. The average user can use small subset of variables while advanced user want to get more detailed picture of the situation. So there cannot be one set of "magic variables" to quietly optimize every situation. However, it is possible to have a more-or-less optimal set of metrics that will allow getting a “good enough” notion about the general health status of MySQL Server.

## MONITIS MYSQL MONITORING AGENT

The latest versions of MySQL (5.5.9 during writing of this article) server provide a collection for more than 335 status variables. Tracking part of these allows you to more-or-less accurately evaluate the health status of a MySQL server.

Fig 1: Screen shot from Monitis MySQL monitoring tab.



Monitis provides an internal agent for monitoring MySQL health. It currently monitors most MySQL status variables (about 245). The whole set of collected variables is divided on 8 categories that cannot be selected simultaneously. Every collection of variables is customizable so users can build their own subset of a variable for every category.

**Table 1: List of MySQL variables categories in Monitis MySQL monitoring**

Category	Ratio of monitored to total number of variables	Description
Com_xxx	100 / 147	Counts number of specific (e.g. INSERT, DELETE, SELECT, etc.) SQL statements that have been executed.
Handler	15 / 16	Counts handler operations, such as the number of times MySQL internally asks a storage engine to read the next row from an index.
Query Cache	8 / 8	The query cache behavior (e.g. number of times query was added and query result was found in the cache, count of free memory blocks in the cache, etc.).
Thread	3 / 4	Inspects the threads cache (e.g. how many threads were created to handle connections, how many threads are not sleeping, etc.).
TMP tables	3 / 3	Counts how many times MySQL has created temporary tables and files.
InnoDB	43 / 47	The InnoDB storage state (e.g. InnoDB buffer pool, log buffer behavior, number of data reads/writes, etc.)
SSL	23 / 23	Presents how the Secure Sockets Layer (SSL) is configured on a server – if applicable.
Other	51 / 90	Inspects some other status variables (e.g. the number of open tables, files, streams, joins, slow queries, etc.).
<b>Totally:</b>	<b>246 / 338</b>	

Of course, not all of the above mentioned MySQL variables monitored by the agent are equally important. Many are very important but some do not give clear information about MySQL health status. Tracking such data doesn't help much and makes MySQL monitoring more cumbersome. Monitoring a large variety of conditions and metrics of a database server is a common mistake that causes many false-positive alerts. This leads to situations where a user cannot understand what is happening with his/her application. It is therefore very important to reduce the "noise" and select the minimal-working set of monitoring metrics. Ideally, when something goes wrong, you should only get one alert, not many alerts from several different health checks. In particular, you should not monitor things that are unreliable in indicating a problem. Finally, supplying the raw data does not always allow the user to get a clear understanding of the current situation.

Fortunately the Monitis agent allows you to customize all the categories and select to monitor only important variables.

## MYSQL HEALTH EVALUATION METRICS

### OVERVIEW

The main aim of this document is to suggest more precise and more informative methods of MySQL server health monitoring based on minimally-possible count of MySQL status variables. Naturally, there are many tools (free and commercial, Web oriented and standalone) that more-or-less accurately allow you monitor and evaluate the MySQL health status. I.e.: Nagios has the [check\\_mysql\\_health](#) plug-in, Oracle provides commercial [MySQL Enterprise monitor](#), [MySQLTuner](#) is a free script written in Perl. Note that the common concept in most modern, popular MySQL monitoring tools is to present new, informative and easy to understand metrics based on raw data, rather than providing the raw status data itself.

## HEALTH EVALUATION

The MySQL server maintains system variables (340 for version 5.5.9) that indicate how it is configured and status variables (338 for version 5.5.9) that provide raw information about the state of the database. The status variables can be divided into two categories – GLOBAL, which collects the values over all connections, and SESSION, which shows the values for the current connection. Most status variables “mindlessly” accumulate the count of all occurred events like commands, queries, connections, caches and buffers filling, etc. This can give a rough assessment of overall status, but usually filtering, preprocessing and simple calculations are required to get more informative metrics that better evaluate MySQL server health status.

Please remember that most MySQL status variables are counters that may just show an average estimation of database server status, which is generally calculated by tracking processes since the last restart of the server. Very often, short-term evaluations and dynamic behaviors of the server are also required. The dynamic evaluation is required to follow the behavior of problematic processes and for real-time detection of dangerous situations.

A series of average values for permanently increasing counters can be prepared for periodical (e.g. per minute) system requests. Therefore the differential behavior of any variable ( $v_{dyn}$ ) can be calculated by the following simple formula which calculates the average differential value ( $v_n - v_{n-1}$ ) of the variable at the end ( $v_n$ ) and beginning ( $v_{n-1}$ ) of a defined time period ( $dt$ ).

$$v_{dyn} = v_n - v_{n-1} / dt$$

The average, long-term estimations are usually used to see general trends and detect the inaccuracies and mistakes in server configuration and problems cause by external events. These metrics have well defined states and can be shown as green/yellow/red light signals. On the other hand, the dynamic behavior used in movement analysis and real-time detection of alarming situations are usually shown in graphs.

In the next chapter, the selected MySQL metrics have been described and the way to obtain them from raw MySQL data has been formulated. Moreover, suggested threshold values have been defined. Of course, users may change the threshold values and set new desired ones.

The metrics defined below are the recommended base for monitoring MySQL status and are used in the Monitis monitoring agent. To make use of the monitoring agent easier, the most important information has been repeated in the tables at the end of this document. Appendix A presents the importance (value) of the defined metrics along the related area and possibility of their graphical presentation. Suggested threshold values have been collected in the second appendix. Possible user actions (recommendations) to improve the situation if metric values cross the threshold are presented in Appendix C.

## SHORT DESCRIPTIONS OF DEFINED METRICS

### KEY CACHE HIT RATE

**EFFECT:** Performance

**DESCRIPTION:**

The key cache hit rate represents the proportion of index values that are being read from the key cache in memory instead of from disk. This should normally be greater than 99%. It is worth noting that this number can be misleading (e.g. the difference between 99% and 99.9% looks small, but it really represents a tenfold increase). Therefore, evaluating, in addition, the dynamic behavior of cache misses ( $key\_reads$ ) can be useful. For example, 5 misses per second usually will not cause increased I/O workload, but 80 per second might cause performance problems.

Usually, it is necessary to set the `key_buffer_size` to at least a quarter, but no more than half, of the total amount of memory on a server. Ideally, it will be large enough to contain all indexes (the total size of all `.MYI` files on the server). If it is impossible for some reason, the best way to evaluate sufficiency of the current `key_buffer_size` is to compare the ratio of the `key_reads` (number of physical reads index blocks from disk) to the `key_read_requests` (number of requests to read a key block from the cache). This ratio should normally be less than 0.01. If this value is not close to zero, it indicates that MySQL key cache is overloaded and `key_buffer_size` variable should be reconfigured.

**FORMULA:**

Normally, the usage of the MyISAM key buffers is evaluated by calculating:

$$\text{key cache hit rate} = 1 - \frac{\text{key\_reads}}{\text{key\_read\_requests}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
Less than 99%	Less than 95%

## KEY CACHE WRITES

**EFFECT:** Performance

**DESCRIPTION:**

Key cache writes is another extremely important variable, because MySQL internally uses MyISAM storage for temporary tables (join, sort, order, etc.) and most `INFORMATION_SCHEMA` tables.

Normally the effectiveness of MyISAM key buffer writes will depend on the kind of queries the MySQL server primarily executes. But it should be configured correctly to avoid mass misses while putting keys into the cache.

**FORMULA:**

The efficiency of MyISAM key cache can be evaluated by key writes to the cache rate:

$$\text{key cache write rate} = \frac{\text{key\_writes}}{\text{key\_write\_requests}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
Less than 90%	Less than 75%

## QUERY CACHE HIT RATE

**EFFECT:** Performance

**DESCRIPTION:**

The query cache should experience a high degree of "hits", meaning that queries in the cache are being reused by other user connections. A low hit rate may mean that not enough memory is allocated to the cache, identical queries are not being issued repeatedly to the server, or that the statements in the query cache are invalidated too frequently by `INSERT`, `UPDATE` or `DELETE` statements.

If the query cache hit rates goes to low please checks MySQL configuration parameters:

- query\_cache\_type (enables or disables query-caching mechanism, and specifies how the cache should work if it's enabled),
- query\_cache\_limit (maximum size of a result set that MySQL can cache)
- query\_cache\_size (total amount of system memory that can be allocated to the query cache).

In most cases increasing the query\_cache\_size can resolve the situation.

**FORMULA:**

$$\text{query cache hit rate} = \frac{\text{qcache\_hits}}{\text{Qcache\_hits} + \text{com\_select}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
Less than 90%	Less than 80%

## QUERY CACHE PRUNES RATE

**EFFECT:** PERFORMANCE

**DESCRIPTION:**

Some already cached queries can be pruned from a query cache because of low memory. This effect can be evaluated by using Qcache\_lowmem\_prunes status variables.

If the rate is low you should increase the query-cache-size slightly to keep most queries inside the cache. But keep in mind that it might be impossible to keep all query results in the cache, especially if the server is under heavy load.

**FORMULA:**

$$\text{query cache prunes rate} = \frac{\text{Qcache\_lowmem\_prunes}}{\text{queries}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
Less than 95%	Less than 90%

## SLOW QUERIES

**EFFECT:** Performance

**DESCRIPTION:**

The server can detect and count of queries that have taken more than long\_query\_time seconds (slow\_queries). Generally, the high value indicates that many queries are not being optimally executed. The percentage of slow queries can be evaluated by using the following formula. Its value should be as low as possible.

Try to find and reconstruct the slow executed queries (in most cases they are not optimally created) using the following technique: switch on the logging of slow queries (log\_slow\_queries ON) and analyze the log results by using e.g.

mysqldumpslow tool. Next, run EXPLAIN on the slow query and then, based on the particular query, take the appropriate actions to fix it. Note that, ideally, there should not be any slow queries, but sometimes there are a few.

**FORMULA:**

$$\text{slow queries rate} = \frac{\text{slow\_queries}}{\text{queries}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
Less than 95%	Less than 90%

### TABLE CACHE HIT RATE

**EFFECT:** Performance

**DESCRIPTION:**

The table cache is used to cache file descriptors for currently open tables and there is a single cache shared by all clients. If the value of open\_tables is approaching the value of table\_cache, this may indicate performance problems.

Normally, when table cache hit rate < 0.85 then table\_cache size probably needs to be increased. Alternately, when table cache fill < 0.95 it means that the table\_cache size is too big and you can decrease its size.

If the hit rate is high it is usually advised to increase the table\_cache value. Notice that table\_cache is related to max\_connections and are used by temporary tables and files. Thus, increase it carefully (and gradually) not to exceed the limitation of your operating system for open file descriptors.

**FORMULA:**

$$\text{table cache hit rate} = \frac{\text{open\_tables}}{\text{opened\_tables}} * 100\%$$

$$\text{table cache fill} = \frac{\text{open\_tables}}{\text{tables\_cache}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
Less than 95%	Less than 90%

### TABLE LOCK CONTENTION

**EFFECT:** Performance

**DESCRIPTION:**

The number of times when waiting is necessary to fulfill a table lock (e.g. during transactions) should be as small as possible to avoid performance problems. The high value of the table lock connection is a complex problem that requires careful analysis. You can first try to optimize queries (since MySQL locks complete tables to separate READ/WRITE access)

so that the optimization of problematic queries can decrease the odds of table locks. Next, optimize tables (try to split the problematic tables). Finally, switch to the InnoDB engine which uses row level locking instead of table and is therefore much less susceptible to lock contention.

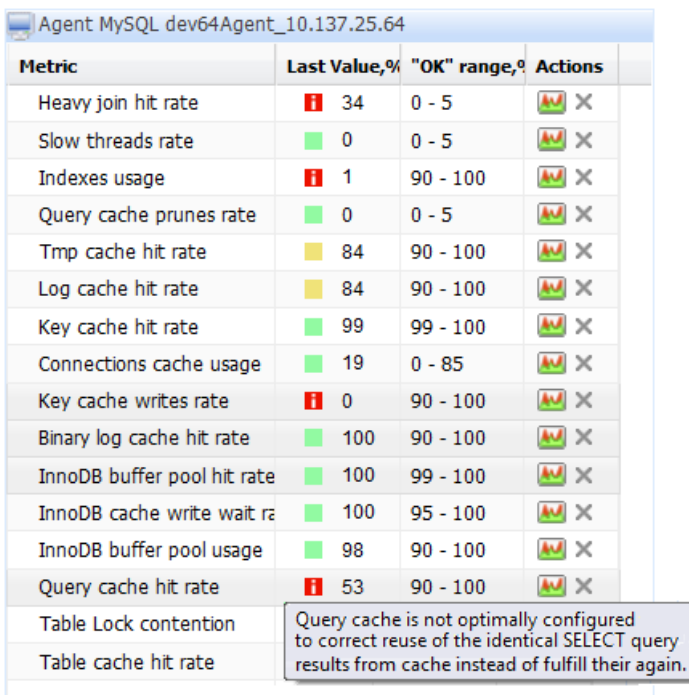
**FORMULA:**

$$\text{table lock contention} = 1 - \frac{\text{table\_lock\_waited}}{\text{table\_lock\_waited} + \text{table\_lock\_immediate}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
Less than 70%	Less than 40%

**Fig 2. Screen shot of Monitis MySQL agent snapshot.**



**INNODB BUFFER POOL HIT RACE**

**EFFECT:** Performance

**DESCRIPTION:**

InnoDB engine should access most data from RAM to reach good performance, and therefore the InnoDB buffer cache hit rate should be high as possibly (close to 100%).

When the buffer is too small, MySQL cannot access some data from memory and needs to use disk, which has much longer access time. In this situation check the correctness of the InnoDB buffer configuration. MySQL provides many configuration parameters; the most important are the `innodb_buffer_pool_instances` and `innodb_buffer_pool_size`. For best efficiency, specify a correct combination of `innodb_buffer_pool_instances` and `innodb_buffer_pool_size` so that each buffer pool instance will be as much as possible (often advised at least 1 gigabyte).

**FORMULA:**

$$\text{InnoDB buffer pool hit ratio} = 1 - \frac{\text{innodb\_buffer\_pool\_reads}}{\text{innodb\_buffer\_pool\_requests}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
Less than 99%	Less than 95%

### INNODB BUFFER POOL USAGE

**EFFECT:** Performance

**DESCRIPTION:**

To obtain high performance the InnoDB buffer pool usage shouldn't be exceeding 100%. If exceeded, it probably means that InnoDB memory buffer is incorrectly configured and its configuration parameters should be checked.

**FORMULA:**

$$\text{InnoDB buffer pool usage} = 1 - \frac{\text{innodb\_buffer\_pool\_pages\_data}}{\text{innodb\_buffer\_pool\_pages\_total}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
Less than 90%	Less than 80%

### INNODB CACHE WRITE WAIT RATE

**EFFECT:** Performance

**DESCRIPTION:**

For optimal performance, InnoDB should wait as little as possible before writing pages. Generally, the InnoDB buffer pool should be a bit (say 10%) larger than your data (total size of InnoDB Table Spaces) because it does not only contain data pages but also adaptive hash indexes, insert buffer, etc.

**FORMULA:**

$$\text{InnoDB buffer pool usage} = 1 - \frac{\text{innodb\_buffer\_pool\_wait\_free}}{\text{innodb\_buffer\_pool\_write\_requests}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
Less than 95%	Less than 90%

### HEAVY JOIN HIT RATE

**EFFECT:** Performance

**DESCRIPTION:**

Join operations are often quite a heavy process, especially when indexes are not used correctly. The InnoDB heavy join hit rate, as defined below, should have a low value. If it does not, it's a good idea to check indexes in the most commonly used tables. They might be not optimized for popular queries.

**FORMULA:**

For evaluating the rate of heavy join operations the following formula can be used:

$$\text{heavy join hit rate} = \frac{\text{select\_full\_join} + \text{select\_range\_check} + \text{select\_scan}}{\text{com\_select}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
Less than 95%	Less than 90%

## INDEXES USAGE

**EFFECT:** Performance

**DESCRIPTION:**

When MySQL does a lot of table scans it might mean that the server does not use indexes efficiently. A good indicator of this problem is when the values of handler\_read\_rnd\_next and handler\_read\_rnd together (number of rows read via full table scans) - are high compared to the sum of handler variables which denote all row accesses - such as handler\_read\_key, handler\_read\_next etc. It is advised to examine tables and queries for proper use of indexes. Turn on the slow query log, identify the queries using a full table scan and tune them. It might also be necessary to reorganize and tune the database schema, indexing or queries.

**FORMULA:**

$$\text{InnoDB full table scan rate} = 1 - \frac{\text{handler\_read\_rnd\_next} + \text{handler\_read\_rnd}}{\text{handlers sum}} * 100\%$$

where:

$$\text{handler sum} = \text{handler\_read\_rnd\_next} + \text{handler\_read\_rnd} + \text{handler\_read\_first} + \text{handler\_read\_key} + \text{handler\_read\_next} + \text{handler\_read\_prev} + \text{handler\_read\_last}$$

**THRESHOLDS:**

Warning	Critical
Less than 90%	Less than 80%

## LOG CACHE HIT RATE

**EFFECT:** Performance

**DESCRIPTION:**

For optimal performance, InnoDB shouldn't have to wait before writing DML activity to the InnoDB log buffer. Therefore, a high number of `innodb_log_waits` (times per second that waiting was required for writing activities to be flushed before continuing) indicates that the log buffer is too small. A large log buffer enables large transactions to run without a need to write the log to disk before the transactions commit and saves disk I/O. The InnoDB log cache hit rate will show how close to optimal the usage of log cache is (should be close to 100%).

When InnoDB log cache values are low try to increase the `innodb_log_buffer_size`. (Normally, the sensible values range from 1MB to 8MB). Notice that optimally the `innodb_log_buffer_size` should be a multiple of `innodb_log_file_size`.

**FORMULA:**

$$\text{InnoDB log cache} = 1 - \frac{\text{innodb\_log\_writes}}{\text{innodb\_log\_write\_requests}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
Less than 90%	Less than 80%

**TMP CACHE HIT RATE**

**EFFECT:** Performance

**DESCRIPTION:**

If the space required to build a temporary table exceeds either `tmp_table_size` or `max_heap_table_size`, MySQL creates a disk-based table in the server's `tmpdir` directory. Also, tables that have TEXT or BLOB columns are automatically placed on disk. For performance reasons it is ideal to have most temporary tables created in memory, leaving exceedingly large temporary tables to be created on disk. The tmp cache hit rate value can show how well the temporary table cache is used. It should be as close to 100% as possible.

**FORMULA:**

$$\text{tmp cache hit rate} = 1 - \frac{\text{create\_tmp\_disk\_tables}}{\text{create\_tmp\_tables}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
Less than 90%	Less than 80%

**BINARY LOG HIT RATE**

**EFFECT:** Performance

**DESCRIPTION:**

The binary log captures occurring DML, DDL, and security changes and stores these changes in a binary format. The binary log enables point-in-time recovery, preventing data loss during a disaster recovery situation. It is used on master replication servers as a record of the statements to be sent to slave servers. It also enables you to review all alterations made to your database. However, the binary log files should be created and recorded in memory to avoid affecting server

performance and the backups on disk should be done from time to time. The binary log cache hit rate can show how close to optimal the usage of the binlog cache is (It should be close to 100%).

With a small value of `binlog_cache_size`, it is easy to overrun this cache, hence a temporary file will be created/used to store this information until the transaction completes. To avoid this, you can try to increase `binlog_cache_size` and/or to change overall binary log format to MIXED mode. Notice that beginning with MySQL 5.5.9, `binlog_cache_size` sets the size for the transaction cache only, and the size of the statement cache is governed by the `binlog_stmt_cache_size` system variable.

**FORMULA:**

$$\text{bin log cache hit rate} = 1 - \frac{\text{binlog\_cache\_disk\_use}}{\text{binlog\_cache\_use}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
Less than 90%	Less than 80%

### THREAD CACHE HIT RATE

**EFFECT:** Performance

**DESCRIPTION:**

Each connection to the MySQL database server runs in its own thread. Thread creation takes time, so rather than killing the thread when a connection is closed, the server keeps the thread in its thread cache and reuses it for a new connection later.

The hit rate that evaluates the usage of threads cache instead of creating a new thread should be as close to 100% as possible.

It's extremely important to measure not only the average but also the dynamic behavior of thread cache hit rate. If `threads_created` keeps going up it means your 'thread\_cache\_size' is set too low and the cached threads couldn't be reused. Just keep bumping up 'thread\_cache\_size' until 'threads\_created' no longer increases. The ideal situation is to get `threads_created` as close as possible to `thread_cache_size`. So no new connections have to wait for new thread allocation.

**FORMULA:**

$$\text{thread cache hit rate} = 1 - \frac{\text{threads\_created}}{\text{connections}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
Less than 95%	Less than 90%

### SLOW THREADS RATE

**EFFECT:** Performance/Availability

**DESCRIPTION:**

The server can detect and count the threads that have taken more than `slow_launch_time` seconds to create (`slow_launch_threads`). If it occurs often it can lead to general system overload and may be caused by the existence of non-optimal queries. The percentage of slow threads can be evaluated by using the following formula

**FORMULA:**

$$\text{slow threads rate} = 1 - \frac{\text{slow\_launch\_threads}}{\text{connections}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
Less than 95%	Less than 90%

## CONNECTIONS USAGE

**EFFECT:** Availability

**DESCRIPTION:**

Once the maximum connection limit for the MySQL server has been reached, no other user connections can be established and errors occur on the client side of the application. So the connections usage shouldn't normally exceed 90 – 95%. Otherwise you may try to increase connections limit (`max_connections`). This situation can occur also when the process failed to release the database connection resources for some reason or simply because of unclosed connections. Anyway, this situation requires careful analysis to identify the concrete reason and fix it.

**FORMULA:**

$$\text{connections usage} = 1 - \frac{\text{threads\_connected}}{\text{max\_connections}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
More than 85%	More than 95%

## MYSQL CPU USAGE

**EFFECT:** Resource-usage

**DESCRIPTION:**

CPU usage by the `mysqld` process should be quite low on a properly configured and well-tuned system. Excessive CPU usage can be indicative of many problems: insufficient RAM, fragmented disks, poorly-tuned queries, etc. This metric can be evaluated by external tools, e.g. “top” Linux simple tool.

```
top -b -n 1 -p $MYSQL_PID > test.txt
```

Where `$MYSQL_PID` is `mysql` process id and `test.txt` is the name of the file where you want to save the output of the command.

**FORMULA:**

The pure (excluding other background processes on host machine) CPU usage by the MySQL process can be evaluated by the following formula:

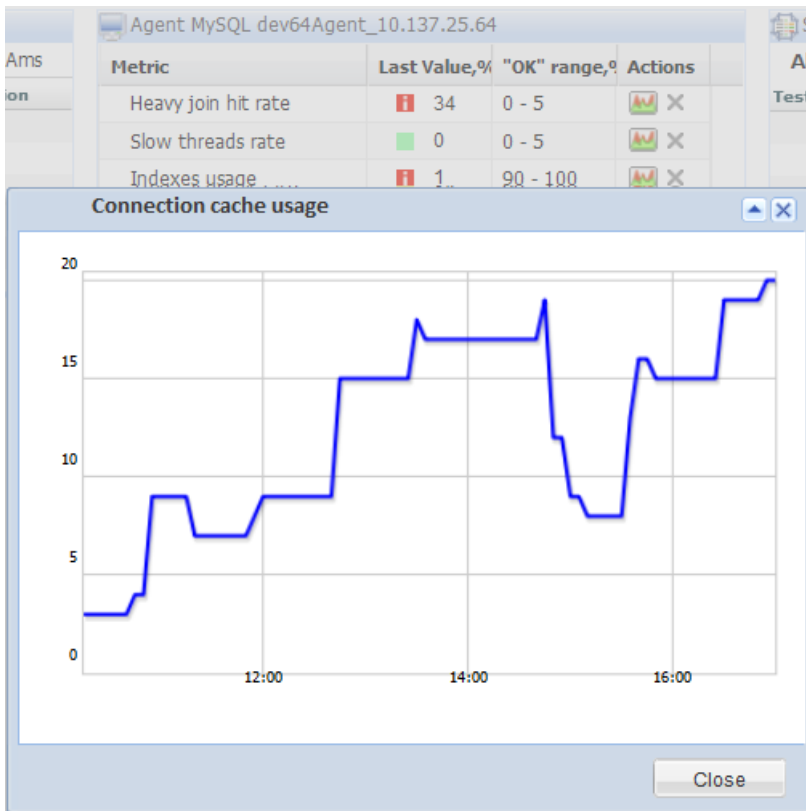
$$\text{CPU usage} = \frac{\text{process\_cpu\_usage}}{\text{process\_cpu\_usage} + \text{cpu\_idle}} * 100\%$$

Where process\_cpu\_usage is the process usage CPU time, expressed as a percentage of total CPU time.

**THRESHOLDS:**

Warning	Critical
More than 85%	More than 95%

**Fig 3. Screen shot of Monitis MySQL metric chart.**



**MYSQL RAM USAGE**

**EFFECT:** Resource-usage

**DESCRIPTION:**

Each connection from the client application is referred to as a thread by MySQL. Its required memory can be evaluated by the following:

$$\text{per-thread memory use} = \text{read\_buffer\_size} + \text{read\_rnd\_buffer\_size} + \text{sort\_buffer\_size} + \text{thread\_stack} + \text{join\_buffer\_size}$$

$$\text{total thread memory use} = \text{per-thread memory use} * \text{max\_connections}$$

There are several other “global buffers” that MySQL creates depending on which table engines you are using. The formula below assumes you have a mix of MyISAM and InnoDB tables and you are using the query cache:

Based MySQL memory usage = key\_buffer\_size + max\_head\_table\_size + innodb\_buffer\_pool\_size + innodb\_additional\_mem\_pool\_size + innodb\_log\_buffer\_size + query\_cache\_size

Memory usage by the mysqld process should be quite low on a properly configured and well-tuned system. This metric can be evaluated in manner similar to MySQL CPU usage, i.e. using top:

```
top -b -n 1 -p $MYSQL_PID > test.txt
```

Where \$MYSQL\_PID is the mysql process id and test.txt is the name of the file where you want to save the output of the command.

**FORMULA:**

The pure (excluding other background processes on host machine) memory usage by the MySQL process can be evaluated with the following formula:

$$\text{memory usage} = \frac{1}{1 + \text{free\_memory}/\text{process\_memory\_usage\_percent} * \text{total\_memory}} * 100\%$$

Where, process\_memory\_usage\_percent is the process usage memory, expressed as a percentage of total memory.

**THRESHOLDS:**

Warning	Critical
More than 85%	More than 95%

**DISK SPACE USAGE**

**EFFECT:** Resource-usage

**DESCRIPTION:**

MySQL server uses disk space to store the following information:

Database directories -- each database corresponds to a single directory.

Table format files – located in the appropriate database directory (\*.frm files).

Data and index files -- created in the appropriate database directory for each table (\*.MYD, \*.MYI files).

InnoDB engine data and index information -- are kept (for all InnoDB tables) in the tablespace (ibdata1 file),

InnoDB undo logs – (ib\_logfile0, ib\_logfile1 files) that are needed if a transaction must be rolled back.

The current allocated disk space for data and indexes can be roughly evaluated by using the following query:

```
USE MYDB;

SHOW TABLE STATUS;
```

This will output data\_length and index\_length for each table in the selected database (mydb). If you add them all together you can get the size used for your particular database. Alternatively, the following query can be used:

```
SELECT SUM(DATA_LENGTH) + SUM(INDEX_LENGTH) AS SIZE \
FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = 'MYDB';
```

If you have direct access to the host machine and root permissions then the physically occupied disk space of MySQL data can be obtained by using following command:

```
du -sh /var/lib/mysql
```

and the free disk space:

```
df -h /var/lib/mysql
```

where /var/lib/mysql is the default location of MySQL data (you should replace it if you change the default location of data during MySQL setup).

**FORMULA:**

So, the evaluation of pure Disk Usage can be done with the following formula:

$$\text{disk usage} = \frac{\text{used\_disk\_space}}{\text{used\_disk\_space} + \text{disk\_free\_space}} * 100\%$$

**THRESHOLDS:**

Warning	Critical
More than 85%	More than 95%

**APPENDIX A. SELECTED METRICS WITH SUGGESTED WEIGHT, INDICATOR OF GRAPHICAL PRESENTATION DESIRABILITY, AND AREA OF IMPACT<sup>1, 2</sup>**

Metric name	Effect	Graph <sup>4</sup>	Weight <sup>3</sup>
Key cache hit rate	Performance	Y	3
Key cache writes rate	Performance		1
Query cache hit rate	Performance	Y	3
Query cache prunes rate	Performance		2
Slow queries rate	Performance		1
Table cache hit rate	Performance	Y	3
Table Lock contention	Performance		2
InnoDB buffer pool hit rate <sup>5</sup>	Performance	Y	3
InnoDB buffer pool usage <sup>5</sup>	Performance		2

InnoDB cache write wait rate <sup>5</sup>	Performance		1
Heavy join hit rate <sup>5</sup>	Performance		2
Indexes usage <sup>5</sup>	Performance		1
Log cache hit rate <sup>5</sup>	Performance		2
Tmp cache hit rate	Performance	Y	3
Binary log cache hit rate	Performance		1
Slow threads rate	Performance		2
Thread cache hit rate	Performance/Availability	Y	3
Connections usage	Availability		3
MySQL process CPU usage <sup>6</sup>	Resource usage	Y	2
MySQL process RAM usage <sup>6</sup>	Resource usage		2
MySQL process Disk usage <sup>6</sup>	Resource usage		1

**NOTES:**

1. The suggested list of metrics does not yet cover the MySQL cluster health status monitoring. This will be defined later.
2. The metric calculation formulas are described in chapter 4.
3. Weight shows the importance of metric while evaluating MySQL health status (3 – high, 2 – middle, 1 – low),
4. Graph column indicates desirability of calculating and graphically displaying the dynamic behavior of metric.
5. InnoDB storage metric.
6. Host machine resource-usage.

**APPENDIX B. LIST OF DEFINED IN THE ARTICLE METRICS WITH VALUES OF THEIR THRESHOLDS<sup>1</sup>**

Metric	Warning	Critical
key cache hit rate	<99%	<95%
Key cache writes rate	<90%	<75%
Query cache hit rate	<90%	<80%
Query cache prunes rate	<95%	<90%
Slow queries rate	<95%	<90%
Table cache hit rate	<90%	<85%
Table Lock contention	<70%	<40%
InnoDB buffer pool hit rate <sup>2</sup>	<99%	<95%
InnoDB buffer pool usage <sup>2</sup>	<90%	<80%
InnoDB cache write wait rate <sup>2</sup>	<95%	<90%
Heavy join hit rate <sup>2</sup>	<95%	<90%
Indexes usage <sup>2</sup>	<90%	<80%
Log cache hit rate <sup>2</sup>	<90%	<80%
Tmp cache hit rate	<90%	<80%
Binary log cache hit rate	<90%	<80%
Thread cache hit rate	<95%	<90%

Slow threads rate	<95%	<90%
Connections usage	>85%	>95%
MySQL process CPU usage <sup>3</sup>	>85%	>95%
MySQL process RAM usage <sup>3</sup>	>85%	>95%
MySQL process Disk space usage <sup>3</sup>	>85%	>95%

**NOTES:**

1. The default values of WARNING and CRITICAL criteria for every metric have been approximately defined. User may have to/want to change this.
2. InnoDB storage metric.
3. Host machine resource-usage.

## APPENDIX C. LIST OF DEFINED METRICS WITH COMMON CAUSES OF ALERT AND POPULAR RECOMMENDATIONS ON HOW TO ADDRESS IT

Metric	Reason	Possible user action <sup>1</sup>
Key cache hit rate	MyISAM key buffer size is too small and cache is overloaded, so MySQL reads part of keys from disk instead of memory cache.	Increase the key_buffer_size, make sure that it is large enough to accommodate indexes for all MyISAM and temporary tables.
Key cache writes rate	MyISAM key buffer size is too small and cache is overloaded, so MySQL writes part of keys to disk instead of storing in memory cache.	As above.
Query cache hit rate	Query cache is not optimally configured to correct reuse of the identical SELECT query.	Check MySQL configuration parameters: query_cache_type, query_cache_limit, query_cache_size. Usually increasing the query_cache_size resolves a situation.
Query cache prunes rate	Query Cache engine removes queries from cache to make room for other queries because of not enough cache size.	Increase the query-cache-size slightly to keep most queries inside cache (often it's not possible to keep all query results in the cache).
Slow queries rate	Some queries exceed long_query_time seconds (default 1 sec).	Try to find and reconstruct slow executed queries.
Table cache hit rate	Table cache size is too small and cache overloaded, so MySQL opens part of tables from disk (not memory).	Carefully (not to exceed the limitation of operating system for open file descriptors) increase the table_cache value.
Table Lock contention	MySQL have to wait for a lock table because of: incorrect database design, suboptimal complex queries, using MyISAM storage instead of InnoDB, etc.	This is complex problem that require careful analysis, but try to: firstly optimize queries, next optimize (split) tables, finally, and switch to the InnoDB engine.
InnoDB buffer pool hit rate <sup>2</sup>	InnoDB buffer size is too small and MySQL have to access part of data from disk (not memory).	Check the correctness of the InnoDB buffer configuration, especially: innodb_buffer_pool_instances, innodb_buffer_pool_size.
InnoDB buffer pool usage <sup>2</sup>	InnoDB buffer is probably incorrectly configured.	Checks the configuration of MySQL memory buffer.
InnoDB cache write wait rate <sup>2</sup>	For optimal performance, InnoDB should not have to wait before writing	The InnoDB buffer pool should be a bit (~10%) larger than your data (total size of InnoDB TableSpaces).

	pages into the InnoDB buffer pool.	
Heavy join hit rate <sup>2</sup>	Select queries cannot use tables indexes (index might not exist, or by not optimize for the query).	Look into tables indexes.
Indexes usage <sup>2</sup>	MySQL are doing a lot of table scans which suggests that your tables are not properly indexed or that your queries are not written to take advantage of the indexes you have.	Tables and queries should be examined for proper use of indexes: identify and tune the queries using full table scan, reorganization and tune a database schema, indexing or queries.
Log cache hit rate <sup>2</sup>	InnoDB log buffer size is too small and MySQL too often puts down the buffer content to the disk file.	Try to increase the <code>InnoDB_log_buffer_size</code> , usually should be between 1 and 8 MB, optimally the <code>InnoDB_log_buffer_size</code> should be multiple of <code>innodb_log_file_size</code> .
Tmp cache hit rate	This occurs if an internal temporary table is created initially as an in-memory table but becomes too large and MySQL have to convert it to an on-disk table.	Try to increase one of these (or both) values: <code>tmp_table_size</code> , <code>max_heap_table_size</code> values.
Binary log cache hit rate	This occurs when part of transactions is too large for the binary log cache and MySQL have to store such statements into a temporary file.	Try to increase <code>binlog_cache_size</code> and/or to change overall binary log format to MIXED mode. For MySQL 5.5.9 and newer look also at <code>binlog_stmt_cache_size</code> system variable.
Thread cache hit rate	The threads cache size is too small and cached connections cannot be reused.	The ideal situation is to get <code>Threads_created</code> as close as possible to <code>Thread_cache_size</code> , so any new connections doesn't wait for new thread allocation.
Slow threads rate	It occurs if something is delaying the new threads creation for connection.	Check existence of suboptimal queries.
Connections usage	It occurs if connections count is close or has reached current setting of maximum connections limit.	Look for of unclosed connections and later increase connections limit ( <code>max_connections</code> ).
MySQL process CPU usage <sup>3</sup>	There are many different reasons that can affect for overloading CPU usage by MySQL server.	Check connection to MySQL (especially from remote hosts). Look for slow queries. Check what currently running queries are doing (SHOW PROCESSLIST) Also check other things like: buffer sizes, table cache, query cache and <code>innodb_buffer_pool_size</code> .
MySQL process RAM usage <sup>3</sup>	There are many different reasons that can affect for overloading RAM usage by MySQL server.	Check the number of allowed connections ( <code>max_connections</code> ). Ensure that total size of "global buffers" ( <code>key_buffer_size</code> , <code>max_heap_table_size</code> , <code>innodb_buffer_pool_size</code> , <code>innodb_additional_mem_pool_size</code> , <code>innodb_log_buffer_size</code> , <code>query_cache_size</code> ) is not too big.
MySQL process Disk space usage <sup>3</sup>	Normally, MySQL uses the Disk space to store the tables' data and indexes. Besides, the disk space is used to store binary log files. In addition, MySQL require temporary disk space for temporary tables.	Optimize the database design to normalize DB and reduce unnecessary data and indexes. Remove outdated binary logs (PURGE BINARY LOGS command). Regularly use OPTIMIZE TABLE command to reclaim the unused space and to defragment the data files.

**NOTES:**

1. Establishing of caches may take a while to reach a state that is representative of normal operations. So it is usually necessary to wait some time until the system is stabilized to get correct results.

2. InnoDB storage metric.
3. Host machine resource-usage.

## APPENDIX D. THE LIST OF MYSQL STATUS VARIABLES NECESSARY TO OBTAIN METRICS DESCRIBED IN THE ARTICLE

To calculate the above described metrics helpful to monitoring MySQL status and performance, it is necessary to track the following 40 MySQL status variables. Please see [MySQL 5.5.9 status variables](#) for their detailed definition.

1. binlog\_cache\_disk\_use
2. binlog\_cache\_use
3. com\_select
4. connections
5. create\_tmp\_disk\_tables
6. create\_tmp\_tables
7. handler\_read\_key
8. handler\_read\_last
9. handler\_read\_next
10. handler\_read\_prev
11. handler\_read\_rnd
12. handler\_read\_rnd\_next
13. innodb\_buffer\_pool\_pages\_data
14. innodb\_buffer\_pool\_pages\_total
15. innodb\_buffer\_pool\_read\_requests
16. innodb\_buffer\_pool\_reads
17. innodb\_buffer\_pool\_wait\_free
18. innodb\_buffer\_pool\_write\_requests
19. innodb\_log\_write\_requests
20. innodb\_log\_writes
21. key\_read\_request
22. key\_reads
23. key\_write\_requests
24. key\_writes
25. max\_connections
26. open\_tables
27. opened\_tables
28. Qcache\_hits
29. Qcache\_lowmem\_prues
30. queries
31. select\_full\_join
32. select\_range\_check
33. select\_scan
34. slow\_lunch\_threads
35. slow\_queries
36. tables\_locks\_immediate
37. tables\_locks\_waited
38. thread\_connected
39. thread\_created
40. uptime

## **ABOUT MONITIS**

Monitis believes that the Cloud is the biggest thing to happen in IT management since IT management. Having seen this vision early, Monitis is now the global leader in developing this market. It is the first affordable network and systems monitoring solution based 100% in the Cloud. Besides Monitis' enthusiastic and loyal user base of 70,000 customers from small businesses to Fortune 500 companies to government agencies and educational institutions, Monitis has won rave reviews from the technology analyst community, such as "Most Innovative Start-Up" from The 451 Group, a listing in OnDemand 100, a ranking by Morgan Stanley, KPMG, and AlwaysOn, of 100 top private companies globally.

Monitis was founded in 2005 by a team of seasoned IT developers fed-up and tired of the limits of software-based tools, while inspired by the promise of the Cloud. Headquartered in San Jose, CA, Monitis's team of IT professionals has extensive experience running enterprise-grade IT businesses, as well as starting and selling several IT start-ups. Monitis employs a global workforce and enjoys a robust average month-on-month revenue growth of over 10%.

For more information, contact:

Monitis Inc.  
Sales & Marketing Department  
[sales@monitis.com](mailto:sales@monitis.com)  
<http://www.monitis.com>  
US & Canada Toll Free: +1-800-657-7949