Optimization and Problem Solving of Pandora FMS

Optimization and Problem Solving of Pandora FMS

Pandora FMS server can monitor about 2000 devices (between 5 and 80 thousands modules, depending on available hardware); but this also requires fine-tuning the configuration of the database.

This article also explains some techniques to detect and solve problems of your Pandora FMS installation.

Percona optimization

All tests and validations are performed with Percona Server for MySQL® 8 (recommended option). Due to the similarities between Percona Server for MySQL 8 and MySQL® 8 there is a great compatibility between both solutions. For the practical purposes of this topic the term MySQL is used, always bearing in mind that the tests are performed on Percona Server for MySQL.

To learn more about “Data Backup and Recovery in Pandora FMS”, go to this link.

General Advice

- Unless otherwise specified, this entire topic refers to MySQL version 8 .
- See also “Upgrading from MySQL 5.7 to MySQL 8”.

To work with tables larger than 2 GB, it is necessary to follow some guidelines:

- MySQL recommends using a 64-bit system. 32-bit systems may have serious problems from year 2038 onwards.
- The more RAM and the more CPU, the better the performance. In our experience, RAM is more important than CPU. The minimum for a system will be 4 GB. A good choice for a large system is 16 GB. Remember that more RAM can speed up key updates by keeping the most used key pages in RAM.
- It is a good idea to be able to remove the system in case of failure. For systems where the database is on another dedicated server for the database use Gigabit Ethernet, preferably with fiber optics rather than copper. Latency is as important as performance.
- Disk optimization is very important for very large databases: databases and tables will have to be split on different disks. In MySQL you can use symbolic links for this. Use different disks for the

system, the database and, if necessary, the binary replication logs.

The use of SSD disks is recommended due to their speed and improved system latency.

- If you start the client and MySQL server are in the same machine, use sockets instead of TCP/IP connexions when connecting with MySQL (this could result in an improvement of a 7.5%). Do it without specifying the host name or the localhost when connecting with MySQL. Disable the start of the binary session and the *replication* if it only launches a MySQL host server.
- Pandora FMS works on MySQL and it is recommended to use the modified version of MySQL (Percona Server for MySQL), which offers a better performance. The plugins programmed are for Percona®.

Please note that the following items greatly affect performance:

- Only use binary logs if you use a MySQL configuration with replication.
- Do not use logs tracing queries or *slow query logs*. This is only recommended in specific cases.

Disable binary replication

If you have configured a Pandora FMS HA system, the binary replication is necessary.
This recommendation is only valid if you have a single Pandora FMS server.

It is enabled by default on most GNU/Linux distros. To disable it, edit the `my.cnf` file, usually in `/etc/my.cnf` and comment the following lines:

```
# log-bin=mysql-bin
# binlog_format=mixed
```

Comment both lines, and then restart MySQL Server.

Disk IO Performance

To learn more about “Data Backup and Recovery in Pandora FMS”, go to this link.

There are three very important configuration tokens, directly related to disk IO, and should be considered because improper IO access is usually the most important bottleneck in MySQL.

innodb_log_file_size

```
innodb_log_file_size = 64M
```

This value is set by default, which can be higher (even 512M) without any risk, except for recovery

in case of any problem or higher disk occupation. The default value of MySQL is 5M, which is very low for production environments with high transaction volume. To change this value with an already running system:

1. First make a complete DUMP in the databases.
2. Delete the Innodb binary index files (usually in `/var/lib/mysql/ib*`).
3. Change file `my.cnf` with the chosen value.
4. Restart MySQL.
5. Load the SQL DUMP.

Since the process is the same as the one to activate the *`innodb_file_per_table`* token (described below), it is recommended to do the whole process simultaneously.

innodb_io_capacity

```
innodb_io_capacity = 300
```

This parameter has the value 200 by default, but you have to know the IOPS of the system disk. You can find out exactly by looking for IOPS and the exact hard disk model, where the recommended values are: 7500RPM → 100 IOPS, 15000 RPM → 190 IOPS, SSD → 1500 IOPS. More information in this link.

innodb_file_per_table

Use a table space for each table:

In Percona is possible to store each InnoDB table and its index in its own file. This feature is called “multiple tablespaces” because each table has its own table space.

The use of multiple space tables can be useful for users that want to move specific tables to separate physical disks or the ones who want to restore table back ups without interrupting the use of the rest of the InnoDB tables.

Multiple tablespaces can be enabled by adding this line to the `mysqld` section of the `my.cnf` file:

```
[mysqld]
innodb_file_per_table
```

After restarting the server, InnoDB will store each new created table in its own `name_table.ibd` file in the database directory to which the table belongs to. This is similar to what the MyISAM store motor does, but MyISAM divides the table in a `tbl_name.MYD` data file and a `tbl_name.MYI` index file.

For InnoDB, data and index are kept together in the `.ibd` file. The `tbl_name.frm` file should be created as usual.

innodb_file_per_table affects only table creation. If you start the server with this option, then the

new tables will be created using `.ibd` files, but you could still have access to the existing tables in the shared table space. If you remove the option, then the new tables will be created in the shared space, but it will be still possible to have access to the tables created in multiple table spaces

Avoiding Disk Flush in Every Transaction

MySQL establishes `autocommit =1` for each connection by default. This is not bad for MyISAM, since what one person writes in the disk is not guaranteed, but for InnoDB it means that any insert / update / delete in an InnoDB table will be registered on the disk (flush).

So, would it be bad if it always writes on the disk? Not at all. It ensures that when there is any compromising event, the data will be there when the database is restored after an incident. The problem is that the DB performance is limited by the physical speed of the disk.

Given that the disk has to write the data in a disk before the writing has been confirmed, this will take some time. Even when considering a searching average time of 9ms for the disk writing, it is limited to approximately 67 commits/ sec1, which is very slow. And while the disk is busy trying for the sector to be written, it cannot read.

InnoDB can avoid some of this limitations by associating some writing together, but, even with this, this restriction still exists. You can prevent if from writing at the end of each transaction, ensuring that it uses an “automatic” writing system, which writes approximately every second. In case of failure, the data from the last second could be lost, but his is something more bearable considering that it achieves greater efficiency. To do it, use the following configuration token `innodb_flush_log_at_trx_commit = 0`. It has this value in the configuration by default.

Bigger KeyBuffer size

Depending on the system total RAM, it is a very important global parameter that speeds up DELETES and INSERT.

```
key_buffer_size = 4M
```

This is the default value in the configuration.

Other important buffers

There are several buffers that are empty by default in some distributions. Modifying these parameters can improve performance significantly compared to the default one. It is important to

make sure that these tokens exist in the MySQL configuration file.

```
key_buffer_size=4M
read_buffer_size=128K
read_rnd_buffer_size=128K
sort_buffer_size=128K
join_buffer_size=4M
```

For MySQL version 8, and later versions, the MySQL development team has withdrawn support for *query cache*, used in previous versions of Pandora FMS; for more information please visit the following web link: https://dev.mysql.com/blog-archive/mysql-8-0-retiring-support-for-the-query-cache/ .

Improving InnoDB Concurrency

There is a parameter that can affect the performance of the MySQL server with Pandora FMS. This parameter is `innodb_thread_concurrency`. This parameter is used to specify how many *`concurrent threads`* MySQL can execute.

This is an advanced parameter and should only be modified manually if performance tuning is required on high concurrency systems.

A wrong setting of this parameter may cause it to run slower than default, so it is especially important to pay attention to several aspects:

- MySQL version: In different MySQL versions this parameter behaves very differently.
- Number of actual (physical) processors: In this regard, you can access the official MySQL documentation.

The recommended value is the number of (physical) CPUs multiplied by 2 plus the number of disks where InnoDB is located.

The value of `innodb_thread_concurrency`has been changed in several versions of MySQL, currently the default value is `0`. A value of `0` means “open as many threads as possible”. Therefore, if in doubt, it can be used:

```
innodb_thread_concurrency = 0
```

MySQL Fragmentation

Like the filesystems, databases also will fragment themselves, slowing the whole system down. In a high performance system like Pandora FMS it is vital that the database state does not affect the system performance. In overloaded systems, the database could block and force the monitoring system to fall down.

Setting up the MySQL server could make Pandora FMS faster, so if you have performance problems, the reason might be a problem in MySQL Setup or problems related with the database.

Check my.cnf file

First verify `my.cnf` file and its basic configuration for MySQL. This configuration file is written in INI format and its location can be determined with the following command:

<bash> mysqld –help –verbose | more </bash>

`my.cnf` setup should be similar to this one (4 GB RAM Server and using an average server hardware). Make sure that you have all these parameters correctly inside section `[mysqld]`:

```
[mysqld]
 datadir=/var/lib/mysql
 socket=/var/lib/mysql/mysql.sock
 user=mysql
 character-set-server=utf8mb4
 skip-character-set-client-handshake

 max_allowed_packet = 64M
 innodb_buffer_pool_size = 800M
 innodb_lock_wait_timeout = 90
 innodb_file_per_table
 innodb_flush_log_at_trx_commit = 0
 innodb_flush_method = O_DIRECT
 innodb_log_file_size = 64M
 innodb_log_buffer_size = 16M
 innodb_io_capacity = 100
 thread_cache_size = 8
 thread_stack    = 256K
 max_connections = 100

 key_buffer_size=4M
 read_buffer_size=128K
 read_rnd_buffer_size=128K
 sort_buffer_size=128K
 join_buffer_size=4M
```

```
sql_mode=""

[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

If you are using MySQL 8 and do not have an HA environment, disable the binary logs with the following command in section `[mysqld]`:

```
skip-log-bin
```

If there is any change in `my.cnf` file, restart MySQL service.

- Verify the service status with `systemctl status mysqld.service`.
- Take a look at the end of the `/var/log/mysqld.log` for any error.
- For more information check the following link in MySQL website.

Restoring databases

To learn more about “Server Management and Administration”, go to this link.

When making certain modifications to `my.cnf` (for example, adding the `innodb_file_per_table` parameter), it is likely that the database will not work when restarting the service. If you get the following error you should restore the previous configuration (you should use root user credentials or *root user*) and perform a database backup:

```
InnoDB: Error: log file ./ib_logfile0 is of different size 0 5242880 bytes
InnoDB: than specified in the .cnf file 0 67108864 bytes!
```

1. Perform a backup of the database:

```
mysqldump -uroot -p pandora > /home/pandora/pandora.sql
```

2. Stop the MySQL server and move the data to a backup folder:

```
systemctl stop mysql
mv /var/lib/mysql /var/lib/mysql.bak
```

3. Create a new folder for the MySQL data (the relevant permits will be assigned in step five):

```
mkdir /var/lib/mysql
```

4. Initialize the MySQL server by specifying the destination folder in the parameter `--datadir`.

This process will generate a temporary password which must be noted (it will be displayed by standard output or stored in `/var/lib/mysqld.log`):

```
mysqld --initialize --datadir /var/lib/mysql
```

5. Assign the appropriate permissions to the new folder:

```
chown -R mysql:mysql /var/lib/mysql
chcon -R system_u:object_r:mysqld_db_t:s0 /var/lib/mysql
```

6. Start the MySQL service and log in with the MySQL client, using the password from step four:

```
systemctl start mysql
mysql -uroot -p
```

Many times MySQL/Percona systems do not load correctly the configuration parameters of the `my.cnf` file, usually because these values have been written outside the `[mysqld]` section.

7. Change the user password root to the one you want (here we use `Pandor4!`):

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'Pandor4!';
```

8. Exit the MySQL client and check that you can log in again using the new password.

9. Access the MySQL client again and create the database:

```
CREATE DATABASE pandora;
USE pandora;
```

10. Load the backup from the database:

```
SOURCE /home/pandora/pandora.sql
```

11. Create the access users for Pandora FMS using the same credentials as in the previous installation (here `Pandor4!` is used) and then give them permissions over the database:

```
CREATE USER 'pandora'@'localhost' IDENTIFIED WITH mysql_native_password BY
'Pandor4!';
CREATE USER 'pandora'@'127.0.0.1' IDENTIFIED WITH mysql_native_password BY
'Pandor4!';
GRANT ALL PRIVILEGES ON pandora.* TO 'pandora'@'localhost';
GRANT ALL PRIVILEGES ON pandora.* TO 'pandora'@'127.0.0.1';
```

12. After having configured the `my.cnf` file and restarted the MySQL service, you should check

that these changes have been correctly applied. To do this, you can check the variables one by one:

```
SHOW VARIABLES LIKE 'innodb_log_file_size';
```

```
SHOW VARIABLES LIKE 'innodb_io_capacity';
```

```
SHOW VARIABLES LIKE 'innodb_file_per_table';
```

Or with a general inquiry such as:

```
SHOW VARIABLES LIKE "innodb%";
```

Once you have verified that Pandora FMS (both Web Console and server) can connect correctly to the database and everything works correctly, you can delete the `mysql.bak` directory:

```
rm -rf /var/lib/mysql.bak
```

Check if isolated datafile for each table is ACTIVE

```
ls -lah /var/lib/mysql/pandora/*.ibd | wc -l
```

There should be more than 100 files there (depending on the version of pandora), each `.ibd` is the data file of each table, when `innodb_file_per_table` parameter is enabled in file `my.cnf`. If you do not have any of these files, `.idb` means it uses a single file to store all information. The previous one means table fragmentation is also present on all tables and performance will worsen each week.

If you have your database running in a single database, first you will need to recreate the database after correctly configuring file `my.cnf` and restarting MySQL.

Optimizing Specific tables

Other less “drastic” solution to solve the fragmentation issue is the use of MySQL OPTIMIZE tool to optimize certain tables of Pandora FMS. To benefit from it, execute directly from MySQL the following:

```
OPTIMIZE TABLE tagente_datos;
OPTIMIZE TABLE tagente;
OPTIMIZE TABLE tagente_datos_string;
OPTIMIZE TABLE tagent_access;
OPTIMIZE TABLE tagente_modulo;
```

```
 OPTIMIZE TABLE tagente_estado;
```

This prevents performance degradation over time and can be done while *hot*, i.e. while the system is running.

In very big environments, the OPTIMIZE option could be “blocked”. In this case, the best option is to rebuild the DB.

After doing these operations, execute:

```
FLUSH TABLES;
```

From the MySQL manual: *For InnoDB tables, OPTIMIZE TABLE is mapped to ALTER TABLE, which rebuilds the table to update index statistics and free unused space in the clustered index*.

MySQL special tokens

There are some very “special” tokens in MySQL, which can improve or worsen the performance:

- `innodb_flush_method`:

```
innodb_flush_method = O_DIRECT
```

This important parameter has an effect on how information is written on the disk.

- `innodb_lock_wait_timeout`:

```
innodb_lock_wait_timeout = 90
```

This helps when there is a bottleneck, so that MySQL does not go away and stops. If it lasts more than 90 lock, there is a real problem.

Check fragmentation table by table

Using MySQL CLI, execute this query:

```
SELECT ENGINE, TABLE_NAME,Round( DATA_LENGTH/1024/1024) AS data_length ,
round(INDEX_LENGTH/1024/1024)
 AS index_length, round(DATA_FREE/ 1024/1024) AS data_free,
(data_free/(index_length+data_length))
 AS frag_ratio FROM information_schema.tables
 WHERE TABLE_TYPE = 'BASE TABLE' AND DATA_FREE> 0 ORDER BY frag_ratio DESC;
```

You should get only the tables with some fragmentation index, for example:

```
+--------+-------------------------+-------------+--------------+-----------+---
---------+
| ENGINE | TABLE_NAME              | data_length | index_length | data_free |
frag_ratio |
+--------+-------------------------+-------------+--------------+-----------+---
---------+
| InnoDB | tserver_export_data     |           0 |            0 |         5 |
320.0000 |
| InnoDB | tagent_module_inventory |           0 |            0 |         6 |
25.6000 |
| InnoDB | tagente_datos_inventory |           4 |            0 |        40 |
9.8842 |
| InnoDB | tsesion_extended        |           1 |            0 |         4 |
3.3684 |
| InnoDB | tagent_access           |           2 |            7 |        27 |
2.9845 |
| InnoDB | tpending_mail           |           2 |            0 |         4 |
2.6392 |
| InnoDB | tagente_modulo          |           2 |            0 |         4 |
2.1333 |
| InnoDB | tgis_data_history       |          24 |           11 |        67 |
1.9075 |
| InnoDB | tsesion                 |           2 |            0 |         4 |
1.7778 |
| InnoDB | tupdate                 |           3 |            0 |         3 |
1.1852 |
| InnoDB | tagente_datos           |         186 |          194 |       399 |
1.0525 |
| InnoDB | tagente_datos_string    |          15 |            9 |        24 |
0.9981 |
| InnoDB | tevento                 |         149 |           62 |        46 |
0.2183 |
| InnoDB | tagente_datos           |        2810 |         2509 |        65 |
0.0122 |
| InnoDB | tagente_datos_string    |         317 |          122 |         5 |
0.0114 |
+--------+-------------------------+-------------+--------------+-----------+---
---------+
```

This query works only on tables with more than 10% of fragmentation. Too big tables (like `tagent_data`) can take a lot of time to get optimized if they are very fragmented. This may affect the production system.

Caution is advised when optimizing such large tables. A normal environment could be optimized once a year and larger environments every six months.

To optimize the `tagent_module_inventory` table (in this case the database is called `pandora`):

```
OPTIMIZE tagent_module_inventory TABLE;
```

A warning message will appear:

```
"Table does not support optimize, doing recreate + analyze instead".
```

If you check again, you should see the fragmentation is gone:

```
+--------+------------------------+-------------+--------------+-----------+---
---------+
| ENGINE | TABLE_NAME             | data_length | index_length | data_free |
frag_ratio |
+--------+------------------------+-------------+--------------+-----------+---
---------+
| InnoDB | tserver_export_data    |           0 |            0 |         5 |
320.0000 |
| InnoDB | tagente_datos_inventory |           4 |            0 |        40 |
9.8842 |
| InnoDB | tsesion_extended       |           1 |            0 |         4 |
3.3684 |
| InnoDB | tagent_access          |           2 |            7 |        27 |
2.9845 |
| InnoDB | tpending_mail          |           2 |            0 |         4 |
2.6392 |
| InnoDB | tagente_modulo         |           2 |            0 |         4 |
2.1333 |
| InnoDB | tgis_data_history      |          24 |           11 |        67 |
1.9075 |
| InnoDB | tsesion                |           2 |            0 |         4 |
1.7778 |
| InnoDB | tupdate                |           3 |            0 |         3 |
1.1852 |
| InnoDB | tagente_datos          |         186 |          194 |       399 |
1.0525 |
| InnoDB | tagente_datos_string   |          15 |            9 |        24 |
0.9981 |
| InnoDB | tevento                |         149 |           62 |        46 |
0.2183 |
| InnoDB | tagente_datos          |        2810 |         2509 |        65 |
0.0122 |
| InnoDB | tagente_datos_string   |         317 |          122 |         5 |
0.0114 |
+--------+------------------------+-------------+--------------+-----------+---
---------+
```

To be able to perform this optimization, there must be enough space on the hard disk to perform the operation. Otherwise an error will appear and the operation will not be performed.

System Load

This is more general, but you need to make sure the system IO is not a bottleneck (disk). Execute the following command to collect the system information:

```
vmstat 1 10
```

Look at the last columns (CPU WA), a value higher than 10 means there is a disk I/O problem that should be solved.

Having CPU-US high is normal, but CPU-SY should not be over 10~15.

Usually, SWAP-SI and SWAP-SO should have value zero, if not, it means the system is using SWAP memory, which degrades performance. Increase RAM or decrease RAM usage in your applications (Pandora FMS server threads, Buffers in MySQL, etc).

Sample output of a “normal” system:

```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0  46248  78664 154644 576800    0    0     2   147    0    9  7 10 83  0  0
 0  0  46248  78656 154644 576808    0    0     0     0   49   37  0  0 100  0
0
 2  0  46248  78904 154648 576740    0    0     0   184  728 2484 63  6 31  0  0
 0  0  46248  79028 154648 576736    0    0    16   616  363  979 21  0 79  0  0
 1  0  46248  79028 154648 576736    0    0     0    20   35   37  0  1 98  1  0
 0  0  46248  79028 154648 576736    0    0     0     0   28   22  0  0 100  0
0
 1  0  46248  79028 154648 576736    0    0     0  3852  141  303  0  0 98  2  0
 2  0  46248  78904 154660 576660    0    0     0   188  642 2354 56  4 40  0  0
 1  0  46248  78904 154660 576680    0    0     0    88  190  634 13  0 86  1  0
 1  0  46248  78904 154660 576680    0    0     0    16   35   40  0  0 100  0
0
 1  0  46248  78904 154660 576680    0    0     0     0   26   21  0  0 100  0
0
 0  0  46248  78904 154660 576680    0    0     0     0   27   27  0  0 100  0
0
 1  0  46248  78904 154724 576616    0    0   112   192  608 2214 52  4 44  0  0
 0  0  46248  78904 154724 576616    0    0     0    76  236  771 16  0 84  0  0
 0  0  46248  78904 154724 576616    0    0     0    20   38   38  0  0 100  0
0
 0  0  46248  78904 154724 576616    0    0     0     0   31   21  0  0 100  0
0
 0  0  46248  78904 154740 576608    0    0     0  3192  187  322  1  0 96  3  0
 1  0  46248  79028 154756 576544    0    0    16   192  632 2087 53  5 42  0  0
 0  0  46248  79028 154760 576568    0    0     0    56  255  927 19  2 79  0  0
 0  0  46248  79028 154768 576564    0    0     0    20   33   44  0  0 100  0
0
```

MySQL Table Partitioning

To use MySQL table partitioning, you should also use the *multiples tablespaces* system described above (`innodb_file_per_table`).

MySQL supports table partitioning, which allows very large tables to be distributed into smaller chunks, such as logical subdivisions (see the MySQL manual for details).

If you have large amounts of data in the database (main and historical) Pandora FMS and estimates that many Console operations that refer to that data (for example *drawing graph*) are slow, then it will improve its performance using table partitioning.

Automatic partitioning

Pandora FMS automatically performs a monthly partitioning in the historical database if it is configured from the Web Console, section Historical database, of the general configuration. For more details consult this link.

Manual partitioning

After having installed Pandora FMS and enabled its historical database, this will be the one that holds the largest amount of data and it is the recommended one to perform a table partitioning Precisely the tables indicated for this are `tagente_datos`, `tagente_datos_string`, `tagente_datos_inc` and `tevento`. The manual practical process for the table `tagente_datos` is described below (see also "Automatic partitioning").

You should check at first that the directory `/var/lib/mysql/pandora_history/*.ibd` has many files (one per table). If this is not the case, you will need to do a *dump* of the database, change the configuration of the `my.cnf` file, restart MySQL, delete the current database and recreate it from the *dump*.

Once you have ensured that `innodb_file_per_table` is enabled, separate the two main databases into different partitions.

- This operation will need enough disk space to be completed. You will need to check how big the `tagente_datos.ibd` file is. If for example it occupies 10 gigabytes, you will need 15 GB of free space to start the operation.
- This operation can take a long time, depending on the size of the table. For example, it would take an hour and a half to split a table with approximately 7500 *modules data* for 100 days (more than 50,000,000 rows).

This is an example for partitioning all of 2023 so far and for future months. To start the process you will need to execute the following query in the MySQL CLI:

```
ALTER TABLE tagente_datos PARTITION BY RANGE (utimestamp) (
PARTITION Jan23 VALUES LESS THAN (UNIX_TIMESTAMP('2023-02-01 00:00:00')),
PARTITION Feb23 VALUES LESS THAN (UNIX_TIMESTAMP('2023-03-01 00:00:00')),
PARTITION Mar23 VALUES LESS THAN (UNIX_TIMESTAMP('2023-04-01 00:00:00')),
PARTITION Apr23 VALUES LESS THAN (UNIX_TIMESTAMP('2023-05-01 00:00:00')),
PARTITION May23 VALUES LESS THAN (UNIX_TIMESTAMP('2023-06-01 00:00:00')),
PARTITION Jun23 VALUES LESS THAN (UNIX_TIMESTAMP('2023-07-01 00:00:00')),
PARTITION Jul23 VALUES LESS THAN (UNIX_TIMESTAMP('2023-08-01 00:00:00')),
PARTITION Aug23 VALUES LESS THAN (UNIX_TIMESTAMP('2023-09-01 00:00:00')),
PARTITION Sep23 VALUES LESS THAN (UNIX_TIMESTAMP('2023-10-01 00:00:00')),
PARTITION Oct23 VALUES LESS THAN (UNIX_TIMESTAMP('2023-11-01 00:00:00')),
PARTITION Nov23 VALUES LESS THAN (UNIX_TIMESTAMP('2023-12-01 00:00:00')),
PARTITION Dec23 VALUES LESS THAN (UNIX_TIMESTAMP('2024-01-01 00:00:00')),
PARTITION pActual VALUES LESS THAN (MAXVALUE)
);
```

Then the following query would have to be run every month to reorganize the partitioning:

```
ALTER TABLE tagente_datos REORGANIZE PARTITION pActual INTO (
PARTITION Jan24 VALUES LESS THAN (UNIX_TIMESTAMP('2024-02-01 00:00:00')),
PARTITION pActual VALUES LESS THAN MAXVALUE);
```

Changing “Jan24” to the month you are in.

Remember again that this operation could take hours, depending on how big the `tagente_datos` table is. You can check the process by watching the size of the partitioning files by running:

```
[root@histdb pandora_history]# ls -lah | grep "#sql"

 -rw-rw----  1 mysql mysql 424M feb 24 05:58 #sql-76b4_3f7c#P#Jan23.ibd
 -rw-rw----  1 mysql mysql 420M feb 24 05:51 #sql-76b4_3f7c#P#Feb23.ibd
 -rw-rw----  1 mysql mysql 128K feb 24 05:40 #sql-76b4_3f7c#P#Mar23.ibd
 -rw-rw----  1 mysql mysql 840M feb 24 05:44 #sql-76b4_3f7c#P#Apr23.ibd
 -rw-rw----  1 mysql mysql 440M feb 24 05:47 #sql-76b4_3f7c#P#May23.ibd
 -rw-rw----  1 mysql mysql  10M feb 24 05:42 #sql-76b4_3f7c#P#Jun23.ibd
 -rw-rw----  1 mysql mysql 404M feb 24 05:56 #sql-76b4_3f7c#P#Jul23.ibd
 -rw-rw----  1 mysql mysql 436M feb 24 05:54 #sql-76b4_3f7c#P#Aug23.ibd
 -rw-rw----  1 mysql mysql 400M feb 24 05:49 #sql-76b4_3f7c#P#Sep23.ibd
 -rw-rw----  1 mysql mysql 408M feb 24 05:52 #sql-76b4_3f7c#P#Oct23.ibd
 -rw-rw----  1 mysql mysql  72M feb 24 06:03 #sql-76b4_3f7c#P#Nov23.ibd
 -rw-rw----  1 mysql mysql 404M feb 24 06:03 #sql-76b4_3f7c#P#Dec23.ibd
 -rw-rw----  1 mysql mysql 416M feb 24 06:00 #sql-76b4_3f7c#P#jan23.ibd
```

DDBB Rebuilding

To find out more abour Pandora FMS backup and data recovery, go to this link.

Partial Rebuilding

MySQL database management system, same as other SQL engines, such as Oracle® is degraded with time due to causes such as data fragmentation produced by deleting and continuous insertion in large tables. In large environments, with a lot traffic volume, there is a very easy way to improve the performance and prevent performance from degrading. This is rebuilding the DB from time to time.

To that end, schedule a service stop, which could last approximately 1 hour.

In this service stop, stop the Pandora FMS WEB console and the server too. Attention: you may leave the Tentacle server so that it can still receive data and these will be processed as soon as the server works again.

Once they have been stopped, do a DB dump (Export); in this example, the database is called `pandora3` and the user must be root:

```
mysqldump -u root -p pandora3> /tmp/pandora3.sql
Enter password:
```

Delete the DB:

```
> mysql -u root -p
 Enter password:
```

```
mysql> drop database pandora3;
Query OK, 87 rows affected (1 min 34.37 sec)
```

Create the DB and import the previous data export from the dump you did at first:

```
mysql> create database pandora3;
Query OK, 1 row affected (0.01 sec)
mysql> use pandora3;
mysql> source /tmp/pandora3.sql
```

This may take a few seconds or several minutes, depending on the size of the database and the resources available on the machine.

It is possible to automatize this process, but, because it is very delicate, the best option is carry it out manually.

Total Rebuilding

This section affects only InnoDB databases. Pandora FMS is built on Innodb databases.

Unfortunately, MySQL is degraded with time, and this affects the global performance of the system.There is no other solution that does not involve rebuilding all the database schemes from scratch, rebuilding the data binary file that MySQL uses to store all the information and the files used to rebuild the transactions.

If you take a look at the `/var/lib/mysql` directory, you can see that there are three files, that have always the same name, and that are, depending on the severity of the case, huge. In this example:

```
-rw-rw----  1 mysql mysql 4.8G 2012-01-12 14:00 ibdata1
-rw-rw----  1 mysql mysql 5.0M 2012-01-12 14:00 ib_logfile0
-rw-rw----  1 mysql mysql 5.0M 2012-01-12 14:00 ib_logfile1
```

The `ibdata1` file is the one that stores all the system Innobd data. In a very fragmented system that has not been “rebuilt” or “installed” for a long time, this system will be big but little efficient. The `innodb_file_per_table` parameter, that has been mentioned before, regulates part of this performance.

Similarly, each database has in the `/var/lib/mysql` directory, one directory to define its structure. Delete them too.

The process is quite easy:

1. Dump (via mysqldump) all the schemes to the disk:

```
mysqldump -u root -p -A> ALL.sql
```

- Stop MySQL.
- Delete `ibdata1`, `ib_logfile0`, `ib_logfile1` and the InnoDB database directories
- Start MySQL.
- Create `pandora` database again (`create database pandora;`)
- Import the backup file (`all.sql`)

```
mysql -u root -p
mysql> source all.sql;
mysql> use pandora;
mysql> source all.sql;
```

The system should work faster now.

Optional Indexes

There are some situations when you can optimize MySQL performance, but giving up other system resources.

This index optimizes speed on graph rendering (a lot), but it uses more disk storage space, and could entail a slightly decrease on INSERT/DELETE operation, due to the Index overhead:

```
ALTER TABLE `pandora`.`tagente_datos`  ADD  INDEX  `id_agente_modulo_utimestamp`
(  `id_agente_modulo`  , `utimestamp`  );
```

At the moment, in the heaviest tables of Pandora FMS in MySQL, this optimization is there by default. It is advisable to have expert help to optimize MySQL tables.

Slow queries study

In some systems, depending on the type of information you have, you can find some “slow queries” that make the system work worse. You may enable logs of this type of queries over a short period of time (since it harms the system performance) in order to consider trying to optimize queries to tables with indexes. To enable these settings, do the following:

- Edit `my.cnf` and add the following lines:

```
slow_query_log = 1
long_query_time = 2
slow_query_log_file = / var / log / mysql_slow.log
```

- To be able to use it and set the admin rules:

```
touch /var/log/mysql_slow.log
chown mysql:mysql /var/log/mysql_slow.log
chmod 640 /var/log/mysql_slow.log
```

- Restart mysql.
- When finishing analyzing which ones are the slow queries, remember to reset the file `my.cnf` commenting the aggregated lines and restarting again MySQL service.

External references

- http://dev.mysql.com/tech-resources/presentations/presentation-oscon2000-20000719/index.html
- http://jeremy.zawodny.com/mysql/mysql-optimization.html

Measuring Pandora FMS for High Capacity

This section describes different methods to configure Pandora FMS in a high capacity environment. It also describes different tools to make load tests, which are useful to adjust the environment to the highest possible process capacity.

Pandora FMS has been configured to support a load of around 2500 agents in systems where database, console and server are in the same machine. The maximum recommended number is around 2500 agents (about 60000 modules) per system, but this number varies greatly depending on whether they are XML agents, remote modules, with high or low intervals, or with systems with high capacity or low memory.

All factors greatly alter the number of agents that a system can manage efficiently. In laboratory tests, 10000 agents have been executed in a single server with basic hardware, but strongly optimized.

Example of High Capacity Servers Configuration

Assuming a RHEL 8 machine with 16 GB of RAM and 8 CPU to be optimized for the maximum processing capacity of the data server (XML).

my.cnf

Only the most important parameters are shown.

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
character-set-server=utf8mb4
skip-character-set-client-handshake
# Disabling symbolic-links is recommended to prevent assorted    security risks
symbolic-links=0
# MySQL optimizations for Pandora FMS
```

```
# Please check the documentation in http://pandorafms.com for  better results
max_allowed_packet = 64M
innodb_buffer_pool_size = 6400M
innodb_lock_wait_timeout = 90
innodb_file_per_table
innodb_flush_log_at_trx_commit = 0
innodb_flush_method = O_DIRECT
innodb_log_file_size = 64M
innodb_log_buffer_size = 16M
innodb_io_capacity = 300
thread_cache_size = 8
thread_stack    = 256K
max_connections = 100
key_buffer_size=4M
read_buffer_size=128K
read_rnd_buffer_size=128K
sort_buffer_size=128K
join_buffer_size=4M
sql_mode=""

[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

pandora_server.conf

Only the most relevant parameters are shown.

```
verbose 3
server_threshold 5
xxxxserver_threads 8
max_queue_files 5000
```

Aspects to take into account:

- The number established in the `verbose` parameter refers to the amount of information that is written in the logs, being advisable not to exceed 3. The higher the number, the lower the performance of Pandora FMS due to the large amount of information to write in the logs.
- A high value (15) of the `server_threshold` parameter causes the DB to suffer less, while the increase in the maximum number of files processed causes the server to *seek files* and fill the buffers every time. These two elements of the configuration are closely linked. In the case of optimizing the network server it may be interesting to lower the `server_threshold` to 5 or 10.
- The very high number of threads (more than 5) set in xxxxserver_threads' only benefits processes with long I/O waits, such as the *network server* or *plugin server.* In the case of the dataserver, which is in process all the time, setting too many threads can even hurt performance. On systems with a slow DB: try different combinations between 1 and 10; with faster disks and multi-core CPUs it could be increased. In the case of optimizing the system for the networkserver, the number can be much

higher, between 10 and 30.

Capacity analysis Tools(Capacity)

Pandora FMS has several tools that can help you to measure properly its hardware and software for the amount of data that it expects to obtain. One of them is useful to “attack” directly the database with fictitious data (dbstress) and the other generates fictitious XML files (xml_stress).

Pandora FMS XML Stress

This is an small script that generates XML data files like the ones sent by Pandora FMS agents. By default it is placed on:

```
/usr/share/pandora_server/util/pandora_xml_stress.pl
```

The scripts read agent names from a text file and generate XML data files for each agent according to a configuration file, where modules are defined as templates.

Modules are filled with random data. An initial value and the probability of the module data changing may be specified.

Run the script like this:

```
./pandora_xml_stress.pl < configuration file >
```

Sample configuration file called `pandora_xml_stress.conf`:

```
# Maximum number of threads, 10 by default.
max_threads 10

# File containing a list of agent names (one per line).
agent_file agent_names.txt

# Directory where XML data files will be placed, /tmp by default.
temporal /var/spool/pandora/data_in

# Pandora FMS XML Stress log file, logs to stdout by default.
log_file pandora_xml_stress.log

# XML version, 1.0 by default.
xml_version 1.0

# XML encoding, ISO-8859-1 by default.
encoding ISO-8859-1
```

```
# Operating system (shared by all agents), Linux by default.
os_name Linux

# Operating system version (shared by all agents), 2.6 by default.
os_version 2.6

# Agent interval, 300 by default.
agent_interval 300

# Data file generation start date, now by default.
time_from 2009-06-01 00:00:00

# Data file generation end date, now by default.
time_to 2009-06-05 00:00:00

# Delay after generating the first data file for each agent to avoid
# race conditions when auto-creating the agent, 2 by default.
startup_delay 2

# Address of the Tentacle server where XML files will be sent (optional).
# server_ip 192.168.50.1

# Port of the Tentacle server, 41121 by default
# server_port 41121

# Module definitions. Similar to pandora_agent.conf.

module_begin
module_name Module 1
module_type generic_data
module_description A long description.
module_max 100
module_min 10
module_exec type=RANDOM;variation=60;min=20;max=80
module_end

module_begin
module_name Module 2
module_type generic_data
module_description A long description.
module_max 80
module_min 20
module_exec type=SCATTER;prob=1;avg=40;min=0;max=80
module_end

module_begin
module_name Module 3
module_type generic_data
module_description A long description.
module_max 80
module_min 20
module_exec type=CURVE;min=20;max=80;time_wave_length=3600;time_offset=0
```

```
module_end

module_begin
module_name Module 4
module_type generic_data_string
module_description A long description.
module_max 100
module_min 10
module_exec type=RANDOM;variation=60;min=20;max=80
module_end

module_begin
module_name Module_3
module_type generic_proc
module_descripcion Module 3 description.
# Initial data.
module_data 1
module_end
```

Send and Receive the Agent Local Configuration

If you activate in your `pandora_xml_stress.conf` the `get_and_send_agent_conf` configuration value to 1, you can make the test load agents work as normal agents, so that they send their configuration file and also the md5.

From Pandora FMS Web console, you can change the remote configuration so that in following executions of `pandora_xml_stress`, it uses the customized configuration from the Pandora FMS Web Console instead of doing it through the `pandora_xml_stress.conf` definition.

Besides this, you may configure where to store locally the `.conf` files of your testing agents with the `directory_confs` configuration token in the `pandora_xml_stress.conf` file.

Configuration File

- max_threads. Number of threads where the script will be executed. This improves the E/S.
- agent_file. Path of the name list file path, separated by a new line.
- temporal. Path of the directory where the made-up XML data files are generated.
- log_file. Path of the log where it will report about its execution script.
- xml_version. Version of the XML data file (by default 1.0).
- encoding XML data file encoding (by default ISO-8859-1).
- os_name. Name of the made-up agent Operative System (Linux by default).
- os_version. Version of the made-up agents Operative System (2.6 by default)
- agent_interval. Interval of the made-up agents in seconds (300 by default).
- time_from. Time from which made-up XML data files are generated, in format `yyyy-MM-dd HH:mm:ss`.
- time_to. Time until which made-up XML data files are generated, in format `yyyy-MM-dd HH:mm:ss`.
- get_and_send_agent_conf. Boolean value 0 or 1. When it is active the made-up agents will try to download by remote configuration a more updated version of the standard configuration file of an agent. And they can be edited through the Pandora FMS Web console.
- startup_delay. Time numeric value in seconds before each agent starts to generate files. It is used to

avoid *race conditions*.
- timezone_offset. Numeric value of the time zone offset.
- timezone_offset_range. Numeric value that is useful to generate the timezone in this range randomly.
- latitude_base Numeric value. It is the latitude geographic area to be used to define fictitious agents.
- longitude_base Numeric value. It is the longitude geographic area to be used to define fictitious agents.
- altitude_base Numeric value. It is the altitude geographic area to be used to define fictitious agents.
- position_radius Numeric value. Defines the radius of the circumference in which the geographic position of the fictitious agent will be set (randomly and based on the parameters latitude_base and longitude_base).

The definition of one module in the script configuration file. If remote configuration has been activated, it will also be the same. It is:

```
module_begin
module_name <name of the module>
module_type <type, p.e: generic_data>
module_description <description>
module_exec type=<type_generation_xml_stress>;<other options separated by ;>
module_unit <units>
module_min_critical <value>
module_max_critical <value>
module_min_warning <value>
module_max_warning <value>
module_end
```

Each one can be configured as:

- type_generation_xml_stress: It can have the values RANDOM, SCATTER, CURVE.
- module_attenuation <value>: The generated module value is multiplied by the specified value, usually between 0.1 and 0.9.
- module_attenuation_wdays <value> <value> ... <value>: The module value is only attenuated during the given days, ranging from Sunday (0) to Saturday (6). For example, the following module simulates a 50% drop in network traffic on Saturdays and Sundays:

```
module_begin
module_name Network Traffic
module_type generic_data
module_description Incoming network traffic (Kbit/s)
module_exec type = RANDOM;variation =50;min =0;max =1000000
module_unit Kbit/s
module_min_critical 900000
module_attenuation 0.5
module_attenuation_wdays 0 6
module_end
```

- module_incremental <value>: If set to 1, the module's previous value is always added to a new value, resulting in an increasing function.
- Others: See below which options are available, depending on the execution type.

Note that `min_critical`, `max_critical`, `min_warning` and `max_warning` are only

available in version 5.0 or later versions.

RANDOM

These have the following options:

- variation. Probability percentage of change regarding the previous value.
- min. Minimum value that the the value could have.
- max. Maximum value that the the value could have.

Numeric

It generates random numeric values between the range values min and max

Booleans

It generates values between 0 and 1.

String

It generates a length string between values min and max. The characters are random between A and Z and capital, lower case letters and also numeric ciphers are included.

External data source (SOURCE)

It allows to use a plain text file as a data source. Options:

- src: Source data file.

The file contains one data per line, there is no limit for lines. For example:

```
4
5
6
10
```

Both numbers and strings are allowed as values. These kinds of modules will use file data to generate module data in Pandora FMS. Data are retrieved sequentially. For example data above will be shown as follows:

```
4 5 6 10 4 5 6 10 4 5 6 10 4 5 6 10 4 5 6 10 4 5 6 10
```

SCATTER

It is only useful for numeric data, and the generated graphics are similar to the ones of a heartbeat, that is, a normal value, and from time to time a *"beat"*.

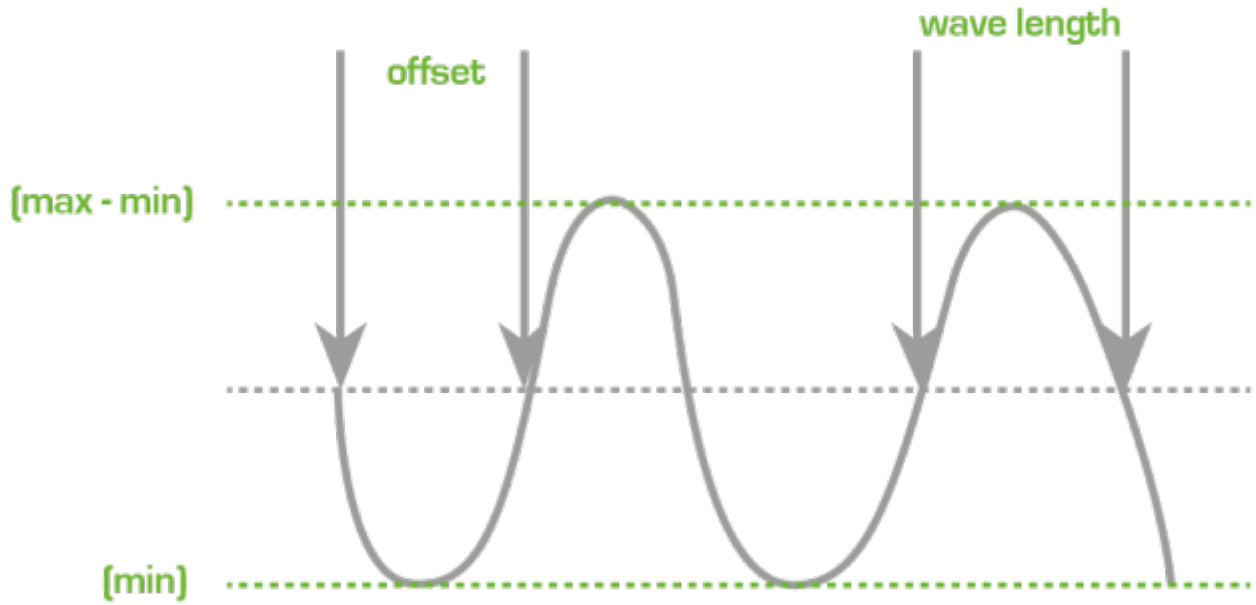
It has the following options:

- min. Minimum value that the value could have.
- max. Maximum value that the value could have.
- prob. Probability percentage that it generates a *"beat"*.
- avg. Average value that should be shown by default if there is no *"beat"*.

CURVE

It generates module data following a trigonometric curve. They have the following options:

- min. Minimum value that the value could have.
- max. Maximum value that the value could have.
- time_wave_length. Numeric value in seconds of the duration of the *"crest"* of the wave.
- time_offset. Numeric value in seconds from the starting point of the wave from time zero with module value zero (similar to the sine graph).



Interesting remarks

- This tool is preconfigured to look for, in all agents, "random", "curve" or boolean name modules that use an interval between 300 seconds and 30 days.

How to measure Data server Processing Capacity

There is a small script called `pandora_count.sh` that is found in the `/usr/share/pandora_server/util/` directory in Pandora FMS server directory. This script is used to measure the processing rate of XML files by the data server, and it uses as reference all the files yet to be processed at `/var/spool/pandora/data_in`, so to be able to use it, thousands of packages yet to be processed are needed (or they must be generated with the tool mentioned before). Once running, you may stop it pressing the keys CTRL+C.

This script simply calculates the file processing rate of the server. It is a somewhat crude measure, but it is useful to get an idea of the effectiveness of the server configuration settings.

Pandora FMS DB Stress

This is a small tool to test database performance. It could also be used to generate periodical or random data (using trigonometric functions) and fill in made-up modules.

Create an agent and allocate the modules to that agent for automatic data injection with this tool. The names should be these ones:

- *random*: To generate random data.
- *curve*: To generate a matching curve using trigonometric functions. It is useful to use the interpolating work with different intervals, etc.
- *boolean*: To generate random boolean data. This way, it is possible to use any name that contains the words: *«random»*, *«curve»* and/or *«boolean»*. For example:

- random_1
- curve_other

Only the “data_server” module type can be chosen.

Pandora FMS DB Stress Fine Adjustment

This tool is preconfigured in order to search, in all agents, the module names *“random”*, *“curve”* or *“boolean”*, that use an interval between 300 seconds and 30 days.

If you wish to modify this performance, edit the `pandora_dbstress` *script* and change some variables at the beginning of the file:

```
# Configure here target (AGENT_ID for Stress)
my $target_module = -1; # -1 for all modules of that agent
my $target_agent = -1;
my $target_interval = 300;
my $target_days = 30;
```

1. The first line of the variable corresponding with `target_module` should be fixed for a fix module or

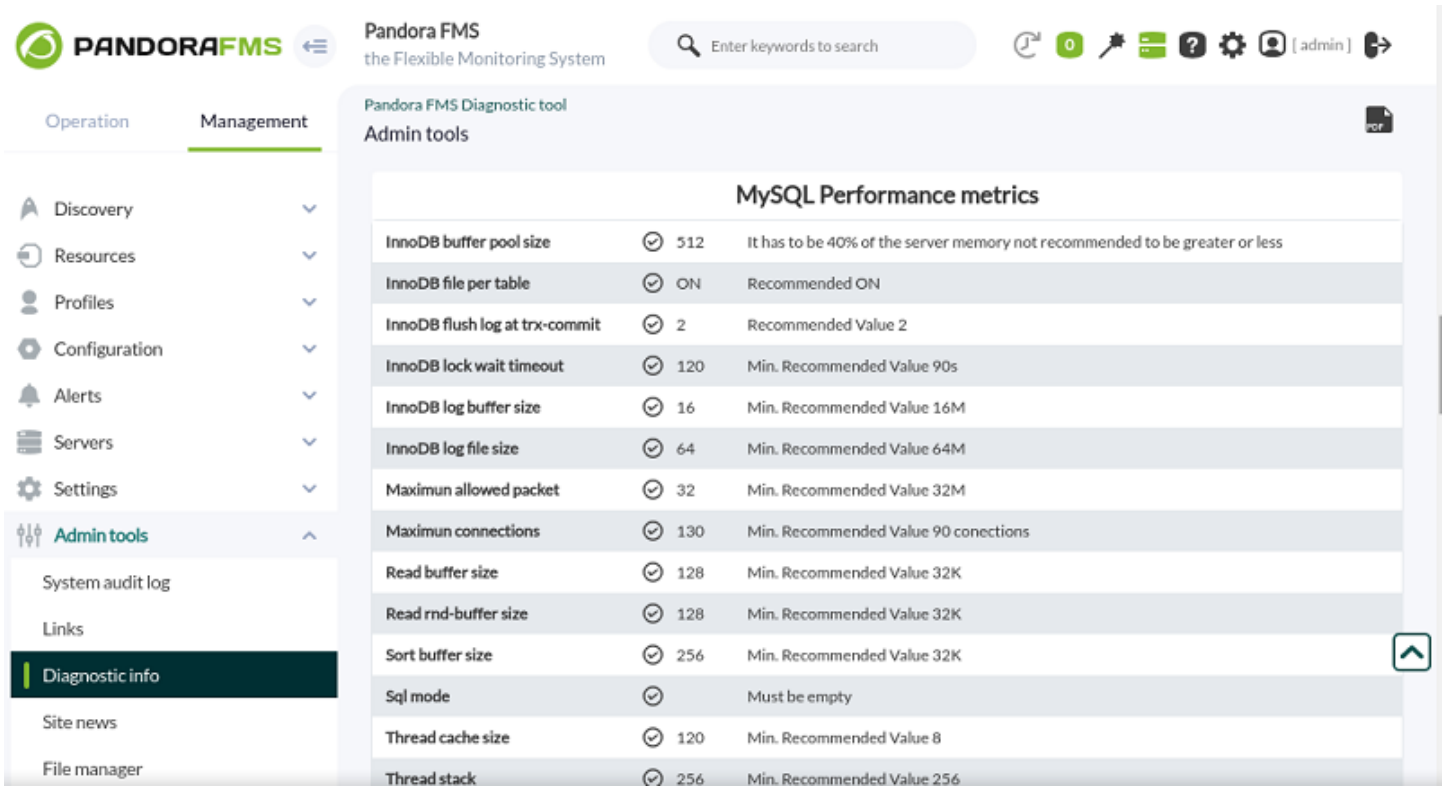
-1 to process all the matching targets.
2. The second line of variable must match `target_agent`, for a specific agent.
3. The third line must match `target_interval`, defined in seconds and which represents the module predefined periodical interval.
4. The fourth line is `target_days` and represents the number of days in the past since the date, in the current *timestamp*.

Diagnostic tools in Pandora FMS

Sometimes there are problems for which direct help from Pandora FMS support is needed. To facilitate the communication with the Support team, Pandora FMS servers have some tools for this.

Diagnostic Info

This tool is located in the Management → Admin tools → Diagnostic Info section, and is designed to provide the most important information about Pandora FMS configuration and its database.



Go back to Pandora FMS documentation index