# percona/pmm: PMM general repo

https://github.com/percona/pmm

Percona Monitoring and Management

[](https://camo.githubusercontent.com/d3097b303f9cb395548e1666b834804333caf11c/68747470733a2f2f7777772e706572636f6e612e636f6d2f73697465732f64656661756c742f66696c65732f706d6d2d6c6f676f2e706e67)

See the [PMM docs](https://www.percona.com/doc/percona-monitoring-and-management/index.html) for more information.

Main repositories

* [pmm-server](https://github.com/percona/pmm-server)
* [qan-api](https://github.com/percona/qan-api)
* [qan-app](https://github.com/percona/qan-app)
* [grafana-dashboards](https://github.com/percona/grafana-dashboards)
* [pmm-client](https://github.com/percona/pmm-client)
* [qan-agent](https://github.com/percona/qan-agent)
* [kardianos-service](https://github.com/percona/kardianos-service) - based on https://github.com/kardianos/service
* [node\_exporter](https://github.com/percona/node_exporter) - based on github.com/prometheus/node\_exporter
* [mysqld\_exporter](https://github.com/percona/mysqld_exporter) - based on github.com/prometheus/mysqld\_exporter
* [mongodb\_exporter](https://github.com/percona/mongodb_exporter) - based on github.com/dcu/mongodb\_exporter
* [proxysql\_exporter](https://github.com/percona/proxysql_exporter)

# Percona Monitoring and Management Documentation

https://www.percona.com/doc/percona-monitoring-and-management/index.html#

Percona Monitoring and Management Documentation

Percona Monitoring and Management (PMM) is an open-source platform for managing and monitoring MySQL and MongoDB performance. It is developed by Percona in collaboration with experts in the field of managed database services, support and consulting.

PMM is a free and open-source solution that you can run in your own environment for maximum security and reliability. It provides thorough time-based analysis for MySQL and MongoDB servers to ensure that your data works as efficiently as possible.

**Basics**

* [**Architecture Overview**](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html)
* [**Deployment Guide**](https://www.percona.com/doc/percona-monitoring-and-management/deploy/index.html)
* [**Usage Information**](https://www.percona.com/doc/percona-monitoring-and-management/using.html)

**Advanced**

* [**Managing PMM Client**](https://www.percona.com/doc/percona-monitoring-and-management/pmm-admin.html)
* [**Using PMM with Amazon RDS**](https://www.percona.com/doc/percona-monitoring-and-management/amazon-rds.html)
* [**Configuring MySQL for best results**](https://www.percona.com/doc/percona-monitoring-and-management/conf-mysql.html)
* [**Security Features in Percona Monitoring and Management**](https://www.percona.com/doc/percona-monitoring-and-management/security.html)
* [**Metrics Monitor Dashboards**](https://www.percona.com/doc/percona-monitoring-and-management/mm-dashboards.html)

**Reference**

* [**Release Notes**](https://www.percona.com/doc/percona-monitoring-and-management/release-notes/index.html)
* [**Contacting and Contributing**](https://www.percona.com/doc/percona-monitoring-and-management/contact.html)
* [**Frequently Asked Questions**](https://www.percona.com/doc/percona-monitoring-and-management/faq.html)
* [**Glossary**](https://www.percona.com/doc/percona-monitoring-and-management/glossary.html)

## Percona Monitoring and Management Architecture

https://www.percona.com/doc/percona-monitoring-and-management/architecture.html

Percona Monitoring and Management Architecture

The PMM platform is based on a simple client-server model that enables efficient scalability. It includes the following modules:

* [PMM Client](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#pmm-client) is installed on every database host that you want to monitor. It collects server metrics, general system metrics, and query analytics data for a complete performance overview. Collected data is sent to PMM Server.
* [PMM Server](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#pmm-server) is the central part of PMM that aggregates collected data and presents it in the form of tables, dashboards, and graphs in a web interface.

The modules are packaged for easy installation and usage. It is assumed that the user should not need to understand what are the exact tools that make up each module and how they interact. However, if you want to leverage the full potential of PMM, internal structure is important.

* [PMM Client](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#pmm-client)
* [PMM Server](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#pmm-server)
* [Deployment Scenarios](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#deployment-scenarios)
  + [Simple Scenario](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#simple-scenario)
  + [Typical Scenario](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#typical-scenario)

PMM is a collection of tools designed to seamlessly work together. Some are developed by Percona and some are third-party open-source tools.

**Note**

The overall client-server model is not likely to change, but the set of tools that make up each component may evolve with the product.

The following diagram illustrates how PMM is currently structured:

_images/pmm-diagram.png

[**PMM Client**](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#id11)

PMM Client packages are available for most popular Linux distributions:

* DEB for Debian-based distributions (including Ubuntu and others)
* RPM for Red Hat Enterprise Linux derivatives (including CentOS, Oracle Linux, Amazon Linux, and others)

There are also generic tarball binaries that can be used on any Linux system.

For more information, see [Installing PMM Client](https://www.percona.com/doc/percona-monitoring-and-management/deploy/client/index.html#install-client).

PMM Client packages consist of the following:

* pmm-admin is a command-line tool for managing PMM Client, for example, adding and removing database instances that you want to monitor. For more information, see [Managing PMM Client](https://www.percona.com/doc/percona-monitoring-and-management/pmm-admin.html#pmm-admin).
* percona-qan-agent is a service that manages the Query Analytics (QAN) agent as it collects query performance data. It also connects with QAN API in [PMM Server](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#pmm-server) and sends over collected data.
* node\_exporter is a Prometheus exporter that collects general system metrics. For more information, see<https://github.com/prometheus/node_exporter>.
* mysqld\_exporter is a Prometheus exporter that collects MySQL server metrics. For more information, see<https://github.com/percona/mysqld_exporter>.
* mongodb\_exporter is a Prometheus exporter that collects MongoDB server metrics. For more information, see<https://github.com/percona/mongodb_exporter>.
* proxysql\_exporter is a Prometheus exporter that collects ProxySQL performance metrics. For more information, see <https://github.com/percona/proxysql_exporter>.

[**PMM Server**](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#id12)

PMM Server runs on the machine that will be your central monitoring host. It is distributed as an appliance via the following:

* Docker image that you can use to run a container
* Open Virtual Appliance (OVA) that you can run in VirtualBox or another hypervisor
* Amazon Machine Image (AMI) that you can run via Amazon Web Services (AWS)

For more information, see [Running PMM Server](https://www.percona.com/doc/percona-monitoring-and-management/deploy/server/index.html#run-server).

PMM Server consists of the following tools:

* **Query Analytics** (QAN) enables you to analyze MySQL query performance over periods of time. In addition to the client-side QAN agent, it includes the following:
  + **QAN API** is the backend for storing and accessing query data collected by percona-qan-agent running on a [PMM Client](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#pmm-client).
  + **QAN Web App** is a web application for visualizing collected Query Analytics data.
* **Metrics Monitor** (MM) provides a historical view of metrics that are critical to a MySQL or MongoDB server instance. It includes the following:
  + **Prometheus** is a third-party time-series database that connects to exporters running on a [PMM Client](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#pmm-client) and aggregates colleted metrics. For more information, see [Prometheus Docs](https://prometheus.io/docs/introduction/overview/) [[1]](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#id3).
    - **Consul** provides an API that a [PMM Client](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#pmm-client) can use to remotely list, add, and remove hosts for Prometheus. It also stores monitoring metadata. For more information, see [Consul Docs](https://www.consul.io/docs/) [[2]](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#id5).

**Warning**

Although the Consul web UI is accessible, do not make any changes to the configuration.

* + **Grafana** is a third-party dashboard and graph builder for visualizing data aggregated by Prometheus in an intuitive web interface. For more information, see [Grafana Docs](http://docs.grafana.org/) [[3]](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#id7).
    - **Percona Dashboards** is a set of dashboards for Grafana developed by Percona.
* **Orchestrator** is a MySQL replication topology management and visualization tool. For more information, see:[Orchestrator Manual](https://github.com/outbrain/orchestrator/wiki/Orchestrator-Manual) [[4]](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#id9).

All tools can be accessed from the PMM Server web interface (landing page). For more information, see [Using the Percona Monitoring and Management Platform](https://www.percona.com/doc/percona-monitoring-and-management/using.html#using).

[**Deployment Scenarios**](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#id13)

PMM is designed to be scalable for various environments. Depending on the size and complexity of your infrastructure, you can deploy it in several ways.

[Simple Scenario](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#id14)

If you have just one MySQL or MongoDB server, you can install and run both modules (PMM Client and PMM Server) on this one database host.

[Typical Scenario](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#id15)

It is more typical to have several MySQL and MongoDB server instances distributed over different hosts. In this case, you can run PMM Server on a dedicated monitoring host, and install PMM Client on every database host that you want to monitor. Data from hosts will be aggregated on the PMM Server.

References

|  |  |
| --- | --- |
| [**[1]**](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#id4) | <https://prometheus.io/docs/introduction/overview/> |

|  |  |
| --- | --- |
| [**[2]**](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#id6) | <https://www.consul.io/docs/> |

|  |  |
| --- | --- |
| [**[3]**](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#id8) | <http://docs.grafana.org/> |

|  |  |
| --- | --- |
| [**[4]**](https://www.percona.com/doc/percona-monitoring-and-management/architecture.html#id10) | <https://github.com/outbrain/orchestrator/wiki/Orchestrator-Manual> |

## Using the Percona Monitoring and Management Platform

https://www.percona.com/doc/percona-monitoring-and-management/using.html

Using the Percona Monitoring and Management Platform

You can access the PMM web interface using the IP address of the host where PMM Server is running. For example, ifPMM Server is running on a host with IP 192.168.100.1, access the following address with your web browser:http://192.168.100.1.

The landing page has links to corresponding PMM tools:

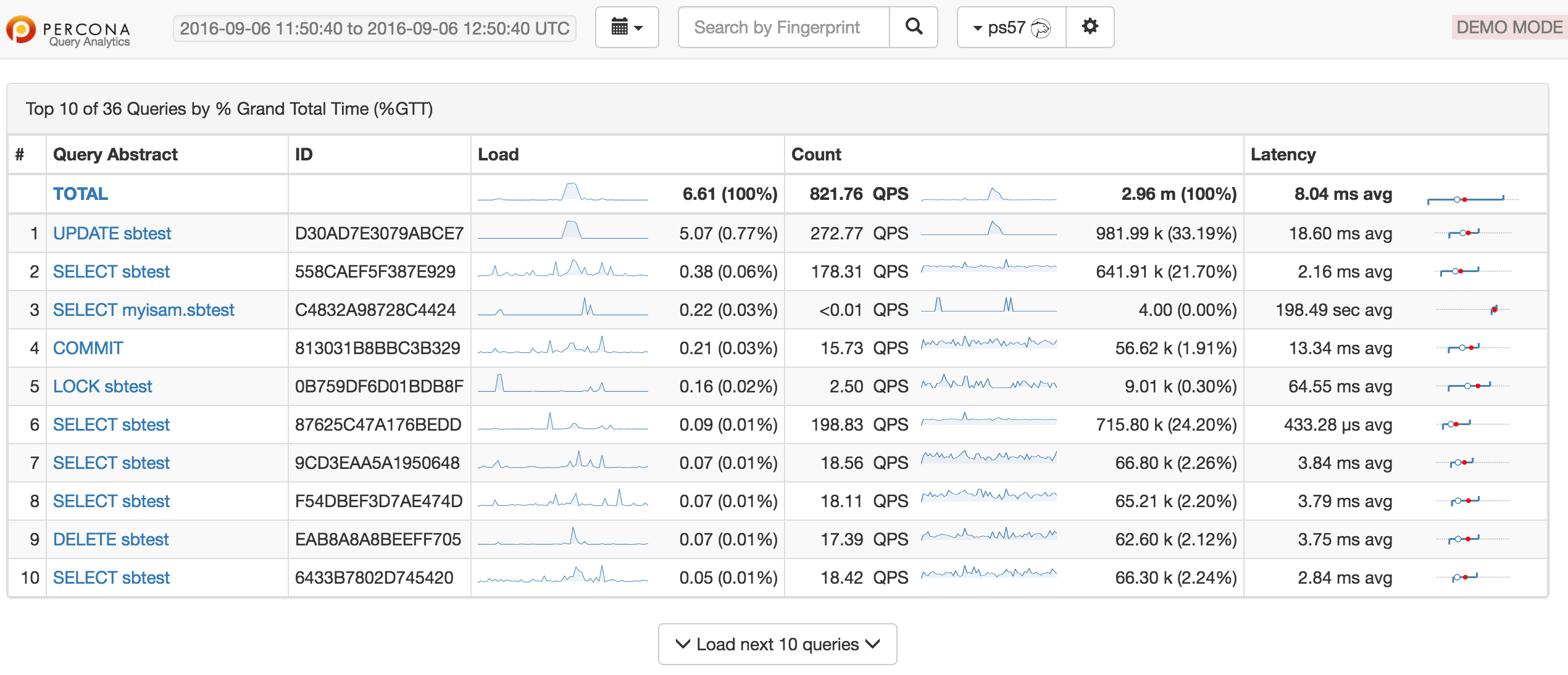
* [Query Analytics](https://www.percona.com/doc/percona-monitoring-and-management/using.html#query-analytics)
* [Metrics Monitor](https://www.percona.com/doc/percona-monitoring-and-management/using.html#metrics-monitor)
* [Orchestrator](https://www.percona.com/doc/percona-monitoring-and-management/using.html#orchestrator)

These tools provide comprehensive insight into the performance of a MySQL host.

[**Query Analytics**](https://www.percona.com/doc/percona-monitoring-and-management/using.html#id2)

The Query Analytics tool enables database administrators and application developers to analyze MySQL queries over periods of time and find performance problems. Query Analytics helps you optimize database performance by making sure that queries are executed as expected and within the shortest time possible. In case of problems, you can see which queries may be the cause and get detailed metrics for them.

The following image shows the Query Analytics app.

[](https://www.percona.com/doc/percona-monitoring-and-management/_images/query-analytics.png)

The summary table contains top 10 queries ranked by **%GTT** (percent of grand total time), which is the percentage of time that the MySQL server spent executing a specific query, compared to the total time it spent executing all queries during the selected period of time.

You can select the period of time at the top, by selecting a predefined interval (last hour, 3 hours, 6 hours, 12 hours, last day, or 5 days), or select a specific inteval using the calendar icon.

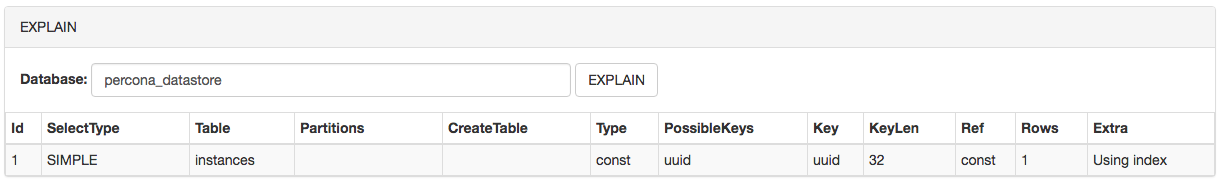
If you have multiple MySQL hosts with PMM Client installed, you can switch between those hosts using the drop-down list at the top.

To configure the QAN agent running on a MySQL host with PMM Client, click the gear icon at the top.

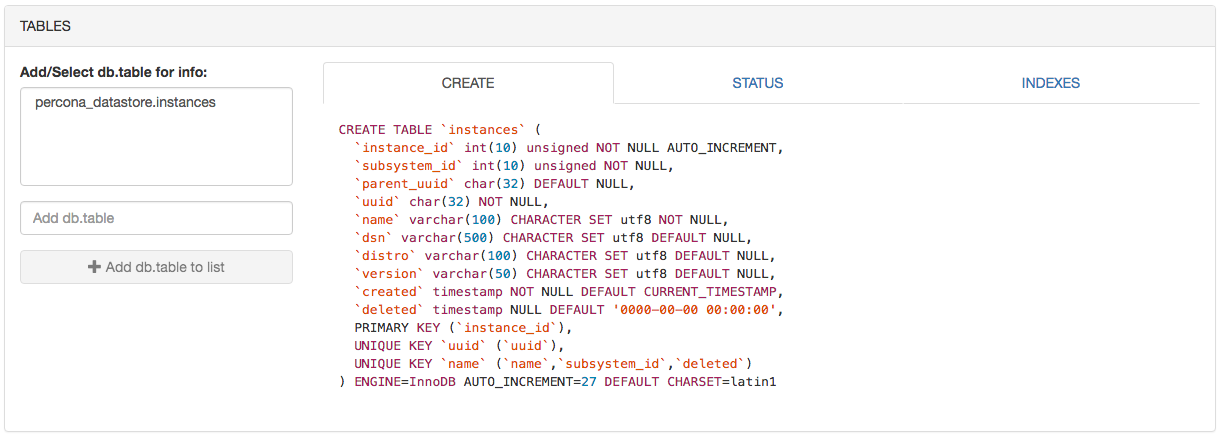
Query Details

You can get details for a query if you click it in the summary table. The details contain all metrics specific to that particular query, such as, bytes sent, lock time, rows sent, and so on. You can see when the query was first and last seen, get an example of the query, as well as its fingerprint.

The details section enables you to run EXPLAIN on the selected query directly from the PMM web interface (simply specify the database).

[](https://www.percona.com/doc/percona-monitoring-and-management/_images/qan-realtime-explain.png)

At the bottom, you can run Table Info for the selected query. This enables you to get SHOW CREATE TABLE, SHOWINDEX, and SHOW TABLE STATUS for each table used by the query directly from the PMM web interface.

[](https://www.percona.com/doc/percona-monitoring-and-management/_images/qan-create-table.png)

Performance Schema

The default source of query data for PMM is the slow query log. It is available in MySQL 5.1 and later versions. Starting from MySQL 5.6 (including Percona Server 5.6 and later), you can select to parse query data from the Performance Schema. Starting from MySQL 5.6.6, Performance Schema is enabled by default.

Performance Schema is not as data-rich as the slow query log, but it has all the critical data and is generally faster to parse. If you are running Percona Server, a [properly configured slow query log](https://www.percona.com/doc/percona-monitoring-and-management/conf-mysql.html#slow-log-settings) will provide the most amount of information with the lowest overhead. Otherwise, using [Performance Schema](https://www.percona.com/doc/percona-monitoring-and-management/conf-mysql.html#perf-schema-settings) will likely provide better results.

**To use Performance Schema:**

1. Make sure that the performance\_schema variable is set to ON:
2. mysql> SHOW VARIABLES LIKE 'performance\_schema';
3. +--------------------+-------+
4. | Variable\_name | Value |
5. +--------------------+-------+
6. | performance\_schema | ON |
7. +--------------------+-------+

If not, add the the following lines to my.cnf and restart MySQL:

[mysql]

performance\_schema=ON

**Note**

Performance Schema instrumentation is enabled by default in MySQL 5.6.6 and later versions. It is not available at all in MySQL versions prior to 5.6.

1. Configure QAN agent to collect data from Performance Schema:

If the instance is already running:

* 1. In the Query Analytics web UI, click the gear button at the top.
  2. Under **Query Analytics**, select **Performance Schema** in the **Collect from** drop-down list.
  3. Click **Apply** to save changes.

If you are adding a new monitoring instance with the pmm-admin tool, use the --query-source perfschemaoption. For example:

sudo pmm-admin add mysql --user root --password root --create-user --query-source perfschema

For more information, run pmm-admin add mysql --help.

[**Metrics Monitor**](https://www.percona.com/doc/percona-monitoring-and-management/using.html#id3)

The Metrics Monitor tool provides a historical view of metrics that are critical to a database server. Time-based graphs are separated into dashboards by themes: some are related to MySQL or MongoDB, others provide general system metrics.

When you open Metrics Monitor for the first time, it loads the **Cross Server Graphs** dashboard. The credentials used to sign in to Grafana depend on the options that you specified when [starting PMM Server](https://www.percona.com/doc/percona-monitoring-and-management/deploy/server/index.html#run-server):

* If you did not specify either SERVER\_USER or SERVER\_PASSWORD, you will be signed in anonymously. You can change to a different existing Grafana user.
* If you specified both SERVER\_USER and SERVER\_PASSWORD, then these credentials will be used to sign in to Grafana.
* If you specified only SERVER\_PASSWORD, a single user (pmm) will be used to sign in to all components (including QAN, Prometheus, Grafana, etc.). You will not be able to change to a different Grafana user.
* If you specified only SERVER\_USER, this parameter will be ignored.

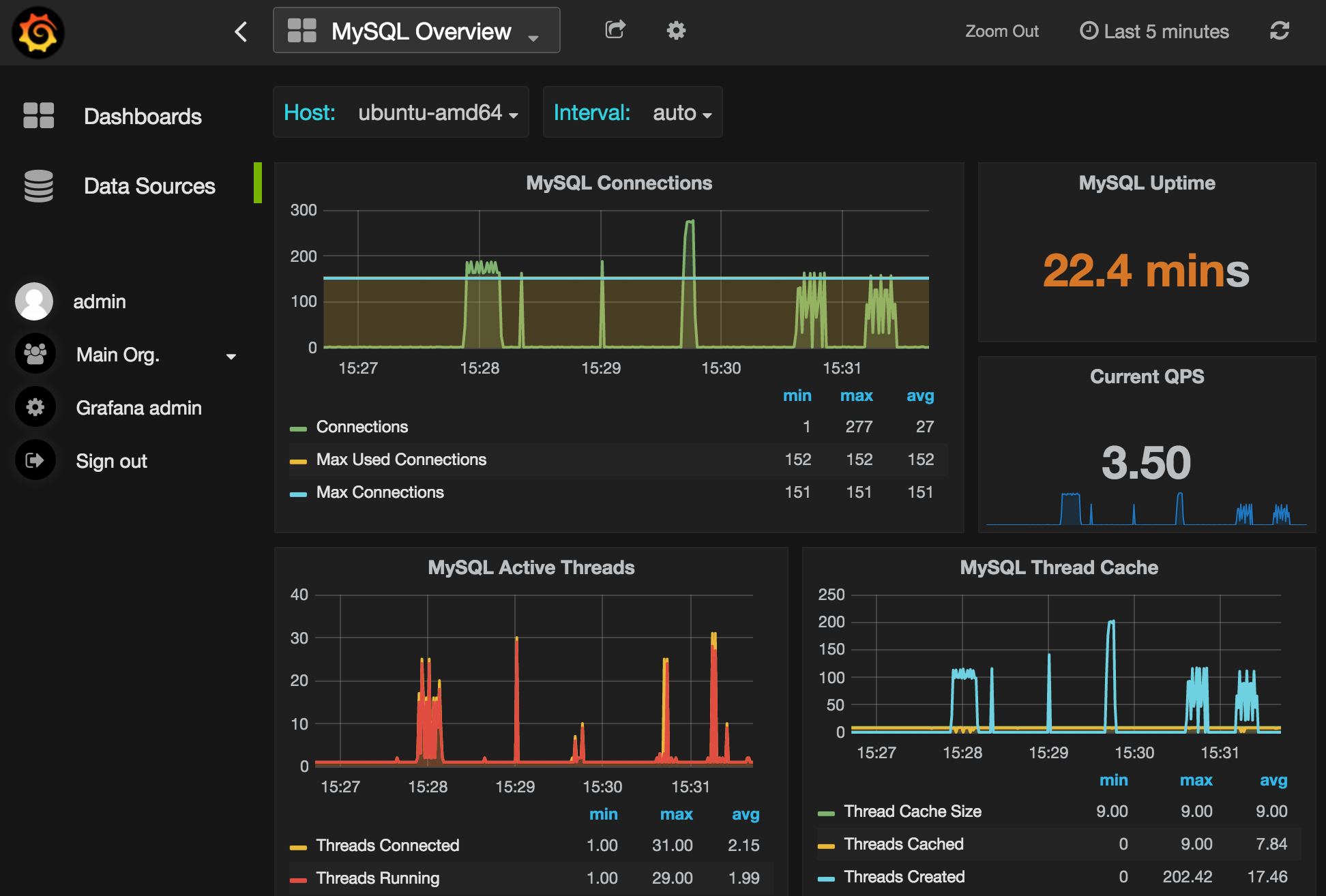
**Warning**

Do not include the # or : symbols in SERVER\_USER.

To access the dashboards, provide default user credentials:

* User: admin
* Password: admin

On the Home screen, select a dashboard from the list of available Percona Dashboards. For example, the following image shows the **MySQL Overview** dashboard:

[](https://www.percona.com/doc/percona-monitoring-and-management/_images/metrics-monitor.png)

[**Orchestrator**](https://www.percona.com/doc/percona-monitoring-and-management/using.html#id4)

**Note**

Orchestrator was included into PMM for experimental purposes. It is a standalone tool, not integrated with PMM other than that you can access it from the landing page.

Orchestrator is a MySQL replication topology management and visualization tool. You can access it using the/orchestrator URL after PMM Server address. Alternatively, you can click the **MySQL Replication Topology Manager** button on the main PMM Server landing page.

To use it, create a MySQL user for Orchestrator on all managed instances:

GRANT SUPER, PROCESS, REPLICATION SLAVE, RELOAD ON \*.\* TO 'orc\_client\_user'@'%' IDENTIFIED BY 'orc\_client\_password’;

**Note**

The credentials in the previous example are default. If you use a different user name or password, you have to pass them when [running PMM Server](https://www.percona.com/doc/percona-monitoring-and-management/deploy/server/index.html#run-server) using the following options:

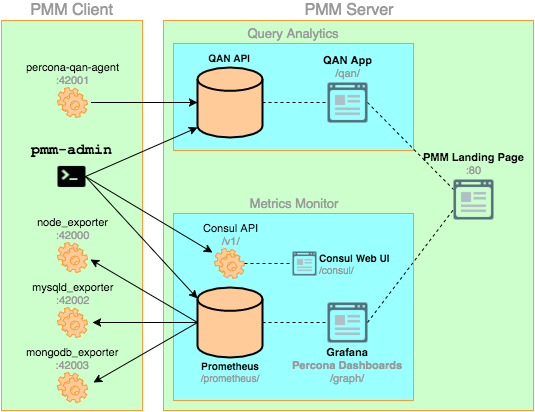
-e ORCHESTRATOR\_USER=name -e ORCHESTRATOR\_PASSWORD=pass

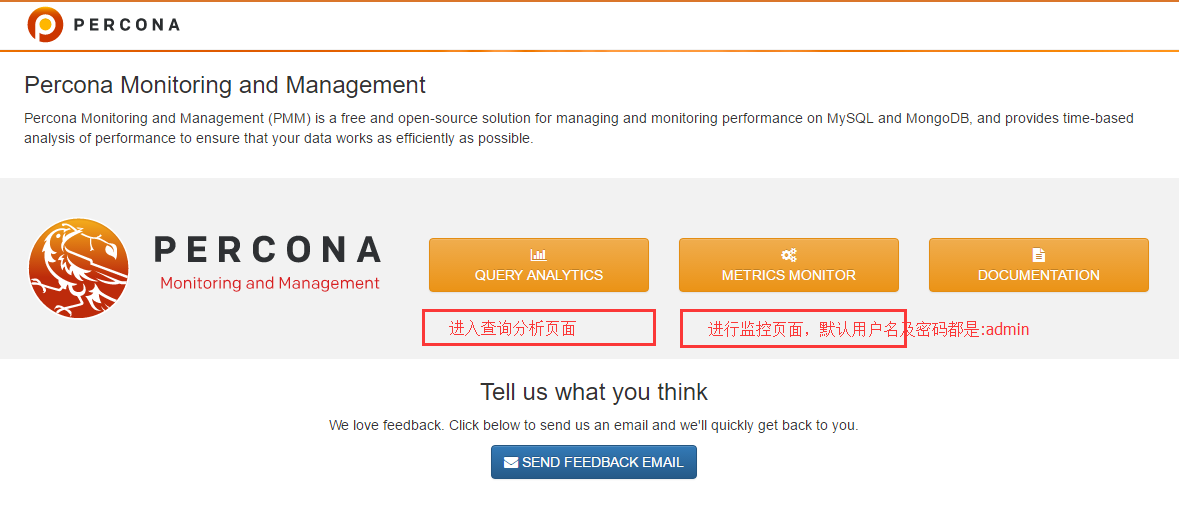
Then you can use the **Discover** page in the Orchestrator web interface to add the instances to the topology.

# Percona Monitoring and Management(PMM) 初体验

- thundermeng的博客 - 博客频道 –

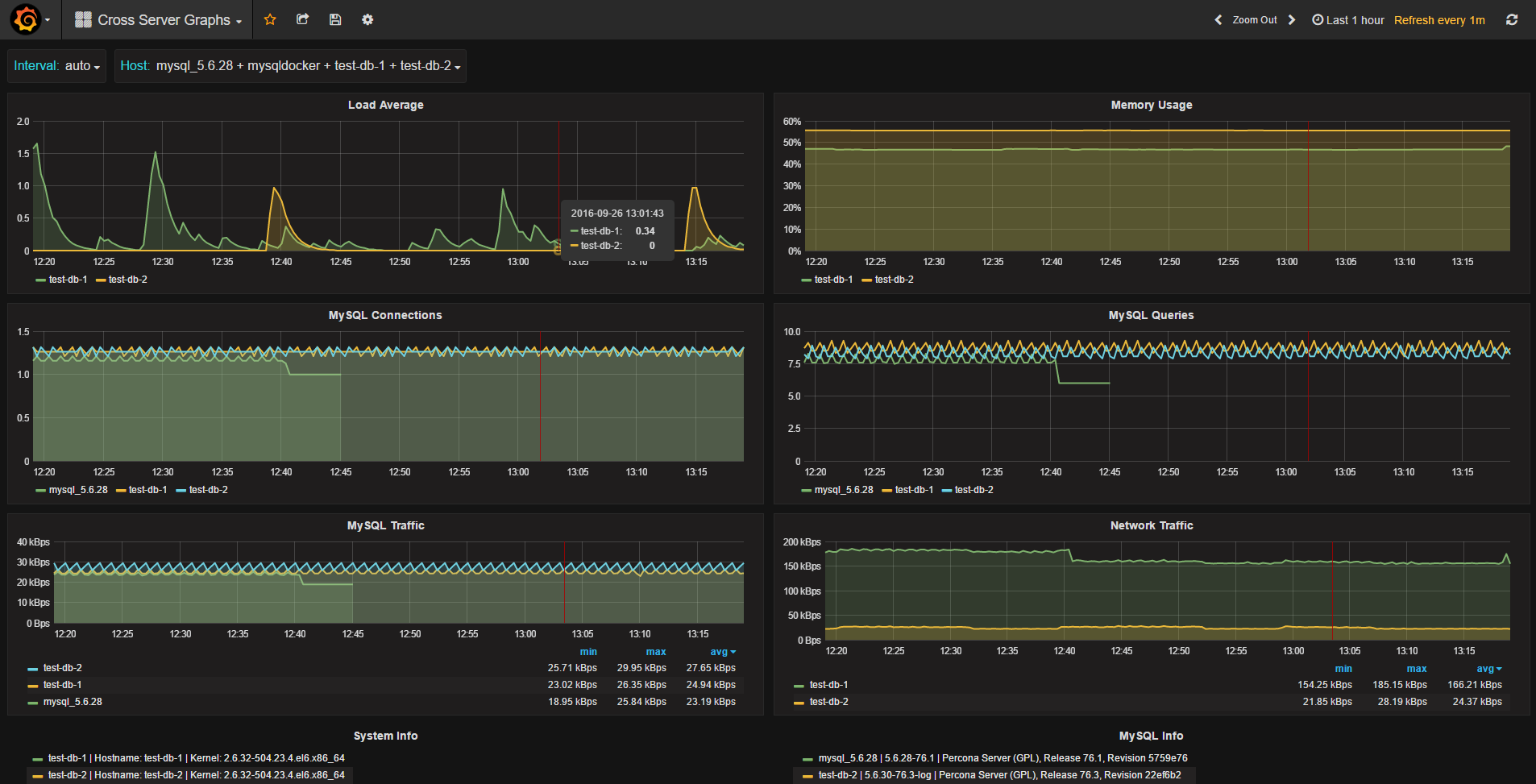
CSDN.NET http://blog.csdn.net/thundermeng/article/details/52669506

Percona Monitoring and Management(PMM) 初体验   
简介   
PMM是一款管理和监控[**MySQL**](http://lib.csdn.net/base/mysql),[**MongoDB**](http://lib.csdn.net/base/mongodb)性能的开源平台。其由percona公司开发，支持和咨询。   
PMM是免费和开源的解决方案，可以运行在自己的环境中提供最大的安全性和可靠性。它提供了全面的基于时间的分析MySQL和MongoDB服务器来确保您的数据尽可能高效地工作。   
PMM[**架构**](http://lib.csdn.net/base/architecture)如下：（注；图片来源percona公司官网 <https://www.percona.com/doc/percona-monitoring-and-management/_images/pmm-diagram.png>）   


安装部署：   
安装PMM Server:   
PMM Server现在官方只提供了[**Docker**](http://lib.csdn.net/base/docker)镜像，不过官方回应，现在正计划做VM镜像及AWS EC2镜像，所以机器需要部署docker服务，简单的docker服务安装直接用Yum就可以了。   
yum install docker   
service docker start   
创建数据容器：   
docker create \   
-v /opt/prometheus/data \   
-v /opt/consul-data \   
-v /var/lib/mysql \   
–name pmm-data \   
percona/pmm-server /bin/true //，如果此步报错，则需要翻墙下载，因国内GFW问题，通过docker pull percona/pmm-server 先把docker pmm-server的镜像拉取下来，再创建容器   
创建pmm-server容器   
docker run -d \   
-p 80:80 \   
–volumes-from pmm-data \   
–name pmm-server \   
–restart always \   
percona/pmm-server   
此时输入机器IP，就可以看到如下页面了。<http://10.237.81.191/>   
   
安装pmm client   
rpm -ivh <https://www.percona.com/downloads/pmm-client/LATEST/pmm-client-1.0.4-1.x86_64.rpm>   
配置连接到pmm-server   
pmm-admin config –server 10.237.81.191   
增加数据项   
pmm-admin add mysql   
pmm-admin add mongodb   
列出当前监控项   
pmm-admin list   
pmm-admin 1.0.4

PMM Server | 10.237.81.191   
Client Name | centos-demo.ledo.com   
Client Address | 10.241.80.47   
Service manager | unix-systemv

SERVICE TYPE NAME CLIENT PORT RUNNING DATA SOURCE OPTIONS

mysql:metrics mysqldocker 42003 NO pmm:*\**@unix(/export/mysql/mysqldocker/mysql.sock)   
添加一台机器中的其它实例：   
pmm-admin add mysql:metrics mysqldocker –user root –password root –host 10.237.81.193 –create-user   


# mysql慢查日志分析工具 percona-toolkit - cmsd - 博客园

http://www.cnblogs.com/cmsd/p/4872258.html

**备忘自： http://blog.csdn.net/seteor/article/details/24017913**

**1. 工具简介**

pt-query-digest是用于分析mysql慢查询的一个工具，它可以分析binlog、General log、slowlog，也可以通过SHOWPROCESSLIST或者通过tcpdump抓取的MySQL协议数据来进行分析。可以把分析结果输出到文件中，分析过程是先对查询语句的条件进行参数化，然后对参数化以后的查询进行分组统计，统计出各查询的执行时间、次数、占比等，可以借助分析结果找出问题进行优化。

pt-query-digest是一个perl脚本，只需下载并赋权即可执行。

[root@test1 ]# wget percona.com/get/pt-query-digest

[root@test1 ]# chmod u+x pt-query-digest

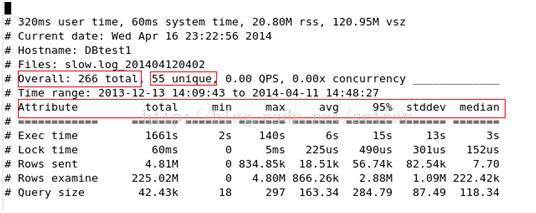
**2.语法及重要选项**

pt-query-digest [OPTIONS] [FILES] [DSN]

**--create-review-table**当使用--review参数把分析结果输出到表中时，如果没有表就自动创建。  
**--create-history-table**当使用--history参数把分析结果输出到表中时，如果没有表就自动创建。 **--filter**对输入的慢查询按指定的字符串进行匹配过滤后再进行分析  
**--limit**限制输出结果百分比或数量，默认值是20,即将最慢的20条语句输出，如果是50%则按总响应时间占比从大到小排序，输出到总和达到50%位置截止。  
**--host  mysql服务器地址  
--user**mysql用户名 **--password**mysql用户密码  
**--history** 将分析结果保存到表中，分析结果比较详细，下次再使用--history时，如果存在相同的语句，且查询所在的时间区间和历史表中的不同，则会记录到数据表中，可以通过查询同一CHECKSUM来比较某类型查询的历史变化。  
**--review** 将分析结果保存到表中，这个分析只是对查询条件进行参数化，一个类型的查询一条记录，比较简单。当下次使用--review时，如果存在相同的语句分析，就不会记录到数据表中。  
**--output** 分析结果输出类型，值可以是report(标准分析报告)、slowlog(Mysql slow log)、json、json-anon，一般使用report，以便于阅读。  
**--since** 从什么时间开始分析，值为字符串，可以是指定的某个”yyyy-mm-dd [hh:mm:ss]”格式的时间点，也可以是简单的一个时间值：s(秒)、h(小时)、m(分钟)、d(天)，如12h就表示从12小时前开始统计。  
**--until** 截止时间，配合—since可以分析一段时间内的慢查询。

**3.      标准分析报告解释**

**第一部分：总体统计结果,如下图**

****

Overall: 总共有多少条查询，上例为总共266个查询。

Time range: 查询执行的时间范围。

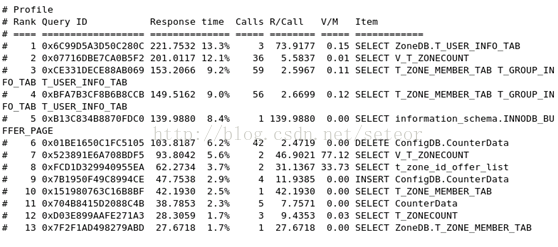
unique: 唯一查询数量，即对查询条件进行参数化以后，总共有多少个不同的查询，该例为55。

total: 总计   min:最小   max: 最大  avg:平均

95%: 把所有值从小到大排列，位置位于95%的那个数，这个数一般最具有参考价值。

median: 中位数，把所有值从小到大排列，位置位于中间那个数。

**第二部分：查询分组统计结果,如下图**



由上图可见，这部分对查询进行参数化并分组，然后对各类查询的执行情况进行分析，结果按总执行时长，从大到小排序。

Response: 总的响应时间。

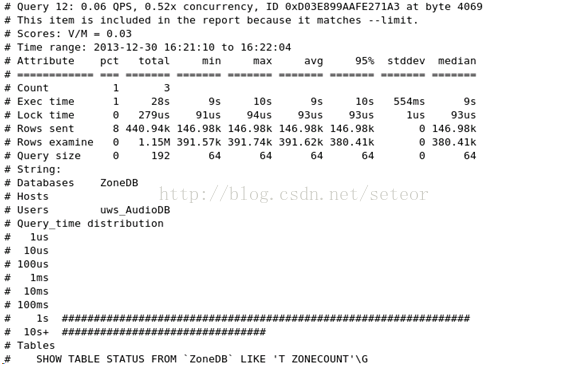
time: 该查询在本次分析中总的时间占比。

calls: 执行次数，即本次分析总共有多少条这种类型的查询语句。

R/Call: 平均每次执行的响应时间。

Item : 查询对象

**第三部分：每一种查询的详细统计结果，如下图：**



由上图可见，12号查询的详细统计结果，最上面的表格列出了执行次数、最大、最小、平均、95%等各项目的统计。

Databases: 库名

Users: 各个用户执行的次数（占比）

Query\_time distribution : 查询时间分布, 长短体现区间占比，本例中1s-10s之间查询数量是10s以上的两倍。

Tables: 查询中涉及到的表

Explain: 示例

**4.用法示例**

**(1)直接分析慢查询文件:**

pt-query-digest  slow.log > slow\_report.log

**(2)分析最近12小时内的查询：**

pt-query-digest  --since=12h  slow.log > slow\_report2.log

**(3)分析指定时间范围内的查询：**

pt-query-digest slow.log --since '2014-04-17 09:30:00' --until '2014-04-17 10:00:00'> > slow\_report3.log

**(4)分析指含有select语句的慢查询**  
pt-query-digest--filter '$event->{fingerprint} =~ m/^select/i' slow.log> slow\_report4.log  
  
**(5) 针对某个用户的慢查询**  
pt-query-digest--filter '($event->{user} || "") =~ m/^root/i' slow.log> slow\_report5.log  
  
**(6) 查询所有所有的全表扫描或full join的慢查询**  
pt-query-digest--filter '(($event->{Full\_scan} || "") eq "yes") ||(($event->{Full\_join} || "") eq "yes")' slow.log> slow\_report6.log  
 **(7)把查询保存到query\_review表**pt-query-digest  --user=root –password=abc123 --review  h=localhost,D=test,t=query\_review--create-review-table  slow.log

**(8)把查询保存到query\_history表**pt-query-digest  --user=root –password=abc123 --review  h=localhost,D=test,t=query\_ history--create-review-table  slow.log\_20140401pt-query-digest  --user=root –password=abc123--review  h=localhost,D=test,t=query\_history--create-review-table  slow.log\_20140402  
  
**(9)通过tcpdump抓取mysql的tcp协议数据，然后再分析**tcpdump -s 65535 -x -nn -q -tttt -i any -c 1000 port 3306 > mysql.tcp.txt  
pt-query-digest --type tcpdump mysql.tcp.txt> slow\_report9.log  
  
**(10)分析binlog**mysqlbinlog mysql-bin.000093 > mysql-bin000093.sql  
pt-query-digest  --type=binlog  mysql-bin000093.sql > slow\_report10.log  
  
**(11)分析general log**pt-query-digest  --type=genlog  localhost.log > slow\_report11.log

# ProxySQL

## ProxySQL首页、文档和下载 - 数据中间层项目 - 开源中国社区

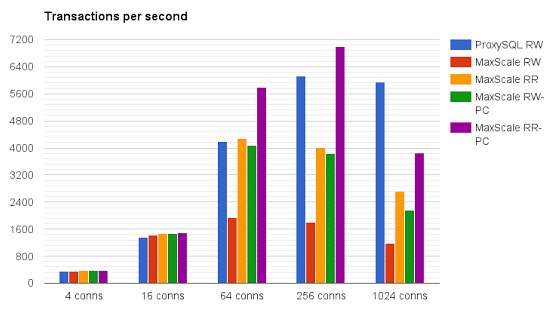
http://www.oschina.net/p/proxysql

ProxySQL

ProxySQL 是一个高性能，高可用性，的数据中间层项目。

高性能

ProxySQL 具有先进的多核架构。 它从根本上构建，支持数十万个并发连接，复用到可能数百个后端服务器。 最大的 ProxySQL 部署跨越了几百个代理。



特性比较：<http://www.proxysql.com/compare>

## ProxySQL之性能测试对比 - 推酷

http://www.tuicool.com/articles/iiEj2m

时间 2017-04-21 15:14:19  [Sean's Notes](http://www.tuicool.com/sites/3iQRB3N)

原文  [http://seanlook.com/2017/04/20/mysql-proxysql-performance-test/](http://seanlook.com/2017/04/20/mysql-proxysql-performance-test/?utm_source=tuicool&utm_medium=referral)

主题 [ProxySQL](http://www.tuicool.com/topics/11030140)[性能测试](http://www.tuicool.com/topics/11350023)

本文会通过sysbench对ProxySQL进行基准测试，并与直连的性能进行对比。与此同时也对 Maxscale 和 Qihu360 Atlas 放在一起参考。

提示：压测前确保把query cache完全关掉。

1. proxysql vs 直连

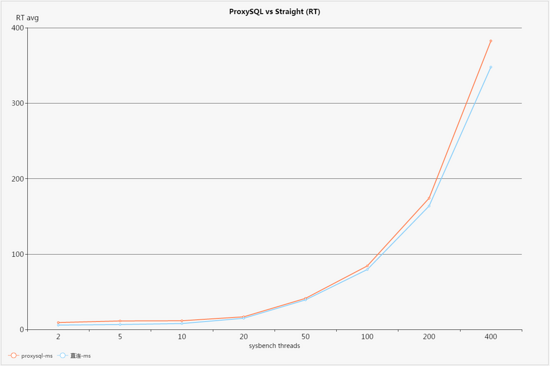
1.1 select nontrx

./bin/sysbench --test=/root/sysbench2/sysbench/tests/db/oltp.**lua** --mysql-host=10.0.100.36 --mysql-port=6033 --mysql-user=myuser --mysql-password=mypass \

--mysql-db=db15 --oltp-tables-count=20 --oltp-table-size=5000000 --report-interval=20 --oltp-dist-type=uniform --rand-init=**on** --max-requests=0 --oltp-test-**mode**=nontrx --oltp-nontrx-**mode**=select \

--oltp-**read**-**only**=**on** --oltp-skip-trx=**on** --max-time=120 --num-threads=2 run

num-threads依次加大 2 5 10 20 50 100 200 400



sysbench线程并发数达到10以下，性能损失在30%以上；达到20，性能损失减少到10%左右。看到proxysql承载的并发数越高，性能损失越少；最好的时候在50线程数，相比直连损失5%。

1.2 oltp dml

混合读写测试。proxysql结果图应该与上面相差无几，因为是主要好在计算 query digest 和规则匹配，与select无异，可参考下节的图示。

sysbench 压测命令：

./bin/sysbench --test=/root/sysbench2/sysbench/tests/db/oltp.**lua** --mysql-host=10.0.100.34 --mysql-port=3306 --mysql-user=myuser --mysql-password=mypass \

--mysql-db=db15 --oltp-tables-count=20 --oltp-table-size=5000000 --report-interval=20 --oltp-dist-type=uniform --rand-init=**on** --max-requests=0 --oltp-**read**-**only**=off --max-time=120 \

--num-threads=2 run

num-threads依次加大 2 5 10 16 20 50 100 200 400

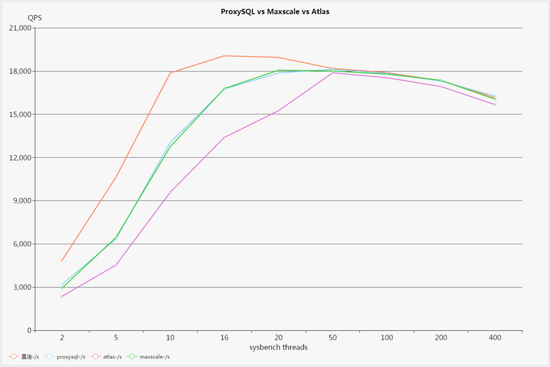
分别对PrxoySQL, Maxscale, Atlas, 直连，四种情况做基准测试

2. proxysql vs maxscale vs atlas

作者自己也有指出，在客户端并发数不高的情况下，maxscale表现比proxysql要好。这里我也特意对maxscale和atlas一起做了个对比。配置基本是最小化的，没有很复杂的规则，只是中间转发。

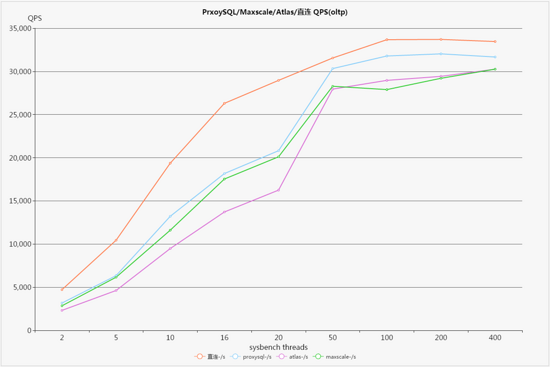
* ProxySQL (v1.3.5): mysql-threads=4
* Atlas 360 (v2.2.1): event-threads=4
* maxscale (v1.4.5): threads=4

2.1 select nontrx

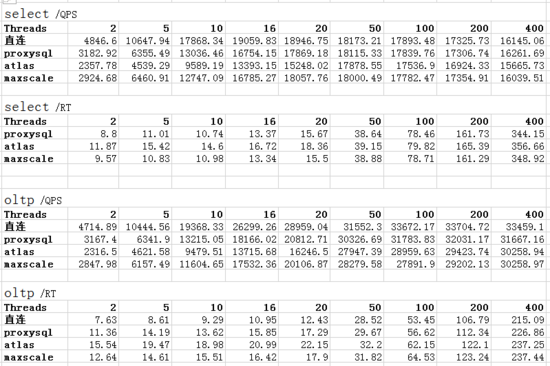


oltp混合读写基准测试，没有复杂配置的情况下，ProxySQL与Maxscale神奇般的几乎重合，Qihu360的atlas要弱一些。

2.2 oltp dml



原始数据：



3. rewrite vs non-rewrite

下面来测一下 query rewrite 对性能的影响，考虑到将来如果要分表，可以在ProxySQL这一层做，应用端无需改动表名。

为了达到效果，这里rewrite只是为表增加了个别名：

-- proxysql admin cli

**update** mysql\_query\_rules **set** match\_pattern="(.\*)(sbtest\d+)(.\*)",replace\_pattern="\1\2 as ttt \3" **where** rule\_id >=61 **and** rule\_id <=92;

**load** mysql **query** **rules** **to** run;

sysbench num-threads=20 的结果：

| **replace?** | **qps** | **response time avg(ms)** |
| --- | --- | --- |
| proxysql replace | 15734.49 | 17.79 |
| proxysql no-replace | 16764.66 | 16.70 |
| 直连 | 18778.43 | 14.91 |

在20个并发线程下，有 rewrite 是 no-rewrite 性能的 93.9% 。测试线程数继续加大到 50，差别更小。

4. lots of rules

测试ProxySQL定义的 query rules 数量（并匹配但不apply），对性能的影响。

测试的规则时批量插入大量能匹配sysbench查询的规则，但 mysql\_query\_rules.apply=0 :

**insert** **into** **mysql\_query\_rules**(active,schemaname,apply,flagIN) **values**

(1,'db15',0,0),(1,'db15',0,0),(1,'db15',0,0),(1,'db15',0,0),(1,'db15',0,0), ...

# **2** **100** **200** **400** **800** **1200** **2000**

这里偶然发现一个问题，flagIN=0的规则必须要在 !=0 的规则前面，否则flagOUT找不到下一个新链入口.(经作者回复是参数 **mysql-query\_processor\_iterations** 控制的)

下面的结果是 sysbench num-threads=20 的几轮数据：（由于结果接近，没作图）

| **matched rules** | **QPS** | **RT avg** | **CPU%** |
| --- | --- | --- | --- |
| 2 | 16741.54 | 16.69 | 151 |
| 100 | 16743.54 | 16.69 | 152 |
| 200 | 16749.94 | 16.71 | 159 |
| 400 | 16556.09 | 16.91 | 176 |
| 800 | 16522.02 | 16.94 | 203 |
| 1200 | 16477.70 | 16.99 | 220 |
| 2000 | 16333.59 | 17.14 | 263 |

看到匹配到的规则随着增多，QPS变化不大，只是略微下降；平均响应时间增加在3%以内；倒是ProxySQL对CPU的负载增加比较明显，匹配的规则从 2 个增加到 2000，cpu使用增加了 74% 。

参考：

* [***https://www.percona.com/blog/2017/04/10/proxysql-rules-do-i-have-too-many/#comment-10967989***](https://www.percona.com/blog/2017/04/10/proxysql-rules-do-i-have-too-many/#comment-10967989)

## sysown/proxysql: High-performance MySQL proxy with a GPL license.

https://github.com/sysown/proxysql

Introduction

ProxySQL is a high performance, high availability, protocol aware proxy for MySQL and forks (like Percona Server and MariaDB).  
All the while getting the unlimited freedom that comes with a GPL license.

Its development is driven by the lack of open source proxies that provide high performance.  
Official website: <http://www.proxysql.com/>  
Benchmarks and old blog posts can be found at <http://www.proxysql.blogspot.com/>  
Forum: <https://groups.google.com/forum/#proxysql/>

Wiki: <https://github.com/sysown/proxysql/wiki/>

## ProxySQL--灵活强大的MySQL代理层

- 奋进的K - 博客频道 - CSDN.NET http://blog.csdn.net/kai404/article/details/52664838

本文是我在学习和验证ProxySQL的过程中，从初识（对其机制猜想或凭几次命令的结果臆断其原理），到逐渐深入（模拟各种场景[**测试**](http://lib.csdn.net/base/softwaretest)、抓包分析、与作者交流）过程中的思路和方法和结论的记录。   
笔者初识proxysql的时候是1.2.1版本，现在几经演进，已经到了1.3.5版本，本文也几经修改，力求跟得上软件的最新进度。

[ProxySQL项目网址](https://github.com/sysown/proxysql)

**一、亮点**

1. 几乎所有的配置均可在线更改（其配置数据基于SQLite存储），无需重启proxysql
2. 基于正则和client\_addr的强大和灵活的路由规则
3. 详细的状态统计，统计结果和pt-query-digest对慢日志的分析结果类似，相当于有了统一的查看sql性能和sql语句统计的入口（Designed by a DBA for DBAs）
4. 自动重连和重新执行机制（auto-reconnect and automatic re-execution of queries using it’s Connections Pool   
   ）：若一个请求在链接或执行过程中意外中断，proxysql会根据其内部机制重新执行该操作
5. query cache功能：比mysql自带QC更灵活，可在mysql\_query\_rules表中依据digest,match\_pattern,client\_addr等维度控制哪类语句可以缓存
6. 支持连接池（connection pool）,区别于atlas之流的连接池实现。文中有详细对比说明

**二、安装**

**rpm包下载地址**

<https://github.com/sysown/proxysql/releases> 推荐rpm形式安装

**Installing from source**

Make sure you have installed the equivalent for each of these packages for your operating system:

automake

bzip2

cmake

make

gcc #>4.4版本

gcc-c++

git

openssl

openssl-devel

patch

[**Git**](http://lib.csdn.net/base/git) clone <https://github.com/sysown/proxysql.git>

[**Go**](http://lib.csdn.net/base/go) to the directory where you cloned the repo (or unpacked the tarball) and run:

make

sudo make install

Compilation time should be around a couple of minutes for the first time around. The configuration file will be found at **/etc/proxysql.cnf**afterwards.

在make这一步遇到了错误：

g++ -fPIC -c -o obj/ProxySQL\_GloVars.oo ProxySQL\_GloVars.cpp -std=c++11 -I../include -I../deps/jemalloc/jemalloc/include/jemalloc -I../deps/mariadb-client-library/mariadb\_client/include -I../deps/libconfig/libconfig-1.4.9/lib -I../deps/re2/re2 -I../deps/sqlite3/sqlite3 -O2 -ggdb -Wall

cc1plus: 错误：无法识别的命令行选项“-std=c++11”

make[1]: \*\*\* [obj/ProxySQL\_GloVars.oo] 错误 1

make[1]: Leaving directory `/usr/local/src/proxysql-master/lib'

make: \*\*\* [build\_lib] 错误 2

网查是由于gcc版本低导致，centos 6的yum源（以及epel源）都只能获取到4.4.7版本

包 gcc-4.4.7-17.el6.x86\_64 已安装并且是最新版本

包 gcc-c++-4.4.7-17.el6.x86\_64 已安装并且是最新版本

而centos7上为4.8版本

换到centos7上，将上述软件安装/更新之后，**make**步骤完成，但是**make install**步骤又出了问题：

install -m 0755 src/proxysql /usr/local/bin

install -m 0600 etc/proxysql.cnf /etc

install -m 0755 etc/init.d/proxysql /etc/init.d

if [ ! -d /var/lib/proxysql ]; then mkdir /var/lib/proxysql ; fi

update-rc.d proxysql defaults

make: update-rc.d：命令未找到

make: \*\*\* [install] 错误 127

update-rc.d是ubuntu的自启动脚本管理软件,未成功安装不影响使用。

安装完成后，自动在/etc/init.d/proxysql增加服务管理脚本（需要把/usr/local/bin/加入$PATH或者软链至 $PATH目录下，脚本中直接用到proxysql命令）

**三、配置**

配置文件/etc/proxysql.cnf和配置[**数据库**](http://lib.csdn.net/base/mysql)文件/var/lib/proxysql/proxysql.db，如果存在 “proxysql.db”文件，则启动过程不解析proxysql.cnf文件；配置文件只在第一次启动的时候读取

官方推荐用admin interface方式

登陆admin interface：

mysql -uadmin -padmin -P6032 -h127.0.0.1

登陆成功后，可通过对main库（默认登陆后即在此库）的global\_variables表中的

admin-admin\_credentials

admin-mysql\_ifaces

两个变量进行更改来修改登录认证

注意：admin interface对配置的存储室基于sqlite的，sqlite支持标准的SQL语法，与[**MySQL**](http://lib.csdn.net/base/mysql)也基本兼容。但是无法用use语句切换数据库，作者对use语句做了兼容（不报错），但是却没有实际效果。

**配置后端DB server：**

两种方式，区别在于：

1. 一种是在往mysql\_servers表中添加server时就为其划分好hostgroup\_id（例如0表示写组，1表示读组）

2. 另一种往mysql\_servers表中添加server时不区分hostgroup\_id（例如全部设为0），然后通过mysql\_replication\_hostgroups表中的值，根据proxysql检测到的各server的read\_only变量值来自动为后端server设置hostgroup\_id

这里强烈推荐用第一种方式：

因为第一种是完全由我们控制的;而第二种假如我们误将读server的read\_only属性设置为0，则proxysql会将其重新分配到写组，这绝对是不期望的。

* 1
* 2
* 3
* 4
* 5
* 6
* 1
* 2
* 3
* 4
* 5
* 6

**四、功能测试**

**实验环境**

MySQL [(none)]> select \* from mysql\_servers;

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

| hostgroup\_id | hostname | port | status | weight | compression | max\_connections | max\_replication\_lag | use\_ssl | max\_latency\_ms | comment |

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

| 0 | 192.168.1.21 | 3307 | ONLINE | 1 | 0 | 1000 | 0 | 0 | 0 | |

| 1 | 192.168.1.10 | 3306 | ONLINE | 1 | 0 | 1000 | 0 | 0 | 0 | |

| 1 | 192.168.1.4 | 3306 | ONLINE | 1 | 0 | 1000 | 0 | 0 | 0 | |

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

#proxysql server的IP为：192.168.1.34

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

**负载均衡测试**

配置好一主(db1，hostgroup0)两从(db2和db3，hostgroup1) ，并且在’mysql\_query\_rules’表中增加一条路由规则

insert into mysql\_query\_rules (rule\_id,active,match\_digest,destination\_hostgroup,apply) values (10,1,'^SELECT',1,1)

* 1
* 1

意为所有以select开头的语句路由到hostgroup1，其余语句路由到hostgroup0

在两台server的mysql-client分别连接至proxysql的6033端口，执行’select @@hostname’观察其分配到的后端server

以mysql -e的形式执行该命令则能够看到请求在两台读server之间变换

[root@db1 ~]# mysql -udm -p'dm' -h192.168.1.34 -P6033 -e "select @**@hostname**" -s -N.

db1

[root@db1 ~]# mysql -udm -p'dm' -h192.168.1.34 -P6033 -e "select @**@hostname**" -s -N

db5

[root@db1 ~]# mysql -udm -p'dm' -h192.168.1.34 -P6033 -e "select @**@hostname**" -s -N

db5

[root@db1 ~]# mysql -udm -p'dm' -h192.168.1.34 -P6033 -e "select @**@hostname**" -s -N

db1

[root@db1 ~]# mysql -udm -p'dm' -h192.168.1.34 -P6033 -e "select @**@hostname**" -s -N

db1

[root@db1 ~]# mysql -udm -p'dm' -h192.168.1.34 -P6033 -e "select @**@hostname**" -s -N

db5

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12

再实验下mysql -e跟多条语句会如何

[root@db1 ~]# mysql -udm -p'dm' -P6033 -h192.168.1.34 -e "select @@hostname;select @@hostname;select @@hostname" -s -N

dm-web5

dm-web5

dm-web5

由以上结果可能会猜想并可印证：在一个client的一个链接周期内，所有query路由到同一台后端。   
**但是：**这只是个假象(是因为正好用到了select @ 语句。按作者所说： sends a query that implicitly disables multiplexing. For example, if you run “SELECT @a” , ProxySQL will disable multiplexing for that client and will always use the same backend connection.)，**proxysql的负载方式目前仅为加权轮询一种（经过作者确认的）**，并无其他机制。

**后端server宕机测试**

对上述[**架构**](http://lib.csdn.net/base/architecture)用sysbench做只读测试，过程中关闭(service mysqld stop)某一台server **(测试均在mysql-monitor\_enabled=false的前提下)**

测试命令

alias sysbench\_test='sysbench --test=/usr/share/doc/sysbench/tests/db/oltp.lua\

--mysql-user=dm --mysql-password='dm' --mysql-port=6033\

--mysql-host=192.168.1.34 --oltp-tables-count=16\

--num-threads=8 run --oltp-skip-trx=on --oltp-read-only=on'

* 1
* 2
* 3
* 4
* 1
* 2
* 3
* 4

结果如下

[root@db3 ~]# sysbench\_test

sysbench 0.5: multi-threaded system evaluation benchmark

Running the test with following options:

Number of threads: 8

Random number generator seed is 0 and will be ignored

Threads started!

ALERT: mysql\_drv\_query() for query 'SELECT c FROM sbtest16 WHERE id=4964' failed: 2013 Lost connection to MySQL server during query

ALERT: mysql\_drv\_query() for query 'SELECT c FROM sbtest12 WHERE id=4954' failed: 2013 Lost connection to MySQL server during query

ALERT: mysql\_drv\_query() for query 'SELECT c FROM sbtest7 WHERE id BETWEEN 4645 AND 4645+99' failed: 2013 Lost connection to MySQL server during query

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12

不是说好的自动重连/重执行吗？为毛报错了呢（atlas有同样问题）

但是两者通过上述的sysbench命令所抛出的错误经过多次比较，有不同：

proxysql报错就一种

failed: 2013 Lost connection to MySQL server during query

atlas报错则是两种

failed: 2013 Lost connection to MySQL server during

failed: 1317 Query execution was interrupted

是不是说明re-execute有效呢？(no，其实这个思路就不对)

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 1
* 2
* 3
* 4
* 5
* 6
* 7

**测试方法错了**：其实关闭后端mysql服务来测试“reconnect”特性应该说从本质上就是不对的，mysql正常关闭会kill掉其上的所有processlist，我们可以用mysql-client来做一些验证

mysql> select @@hostname;

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

| @@hostname |

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

| db1 |

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

1 row in set (0.00 sec)

\*\*通过别的tty重启该mysql\*\*

mysql> select @@hostname;

ERROR 2006 (HY000): MySQL server has gone away

No connection. Trying to reconnect...

Connection id: 4

Current database: \*\*\* NONE \*\*\*

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

| @@hostname |

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

| db1 |

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

1 row in set (0.00 sec)

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19

可以看到mysql-client是具有重连(reconnect)功能的，然后我们来做一个kill掉mysql-client线程（也就是mysql在关闭时会kill掉所有线程）的操作

mysql> select @@hostname;

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

| @@hostname |

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

| db1 |

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

1 row in set (0.00 sec)

\*\*在mysql服务端查询并kill掉这个链接\*\*

mysql> select @@hostname;

ERROR 2013 (HY000): Lost connection to MySQL server during query

mysql>

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11

我们看到了之前sysbenche测试过程中关闭一台从库时的报错。也就是说这个场景下这个报错不是proxysql的‘reconnect’机制的问题，倒更像是‘re-execute’机制的问题。**而kill操作应该理解为是mysql“主动”的动作，而非“异常”，所以这就不符合proxysql 的”re-execute”特性了，故而会报错**。

**应该换一种方式来进行这个测试**：

* 模拟网络层无法通信的异常

我们在两台slave上通过iptables阻断peoxysql到本机的3306端口来模拟无法链接和链接中断异常

在sysbench开始之后再重启iptables已经建立的链接会被保留，所以新规则要放到第一条的位置   
-A INPUT -s 192.168.1.34 -p tcp -m tcp –dport 3306 -j DROP

首先 ：启动sysbench只读测试   
然后 ：在测试结束前，修改db1的iptables，禁止来自proxysql的请求进入   
然后 ：观察sysbench的输出和proxysql.log的输出   
结果 ：sysbench等待很长时间，依然无法完成，同时，proxysql也不会把db1标记为SHUNNED   
proxysql.log输出：

2016-09-02 11:37:54 MySQL\_Session.cpp:49:kill\_query\_thread(): [WARNING] KILL QUERY 133 on 192.168.1.4:3306

2016-09-02 11:37:54 MySQL\_Session.cpp:49:kill\_query\_thread(): [WARNING] KILL QUERY 136 on 192.168.1.4:3306

2016-09-02 11:37:54 MySQL\_Session.cpp:49:kill\_query\_thread(): [WARNING] KILL QUERY 135 on 192.168.1.4:3306

2016-09-02 11:37:54 MySQL\_Session.cpp:49:kill\_query\_thread(): [WARNING] KILL QUERY 137 on 192.168.1.4:3306

2016-09-02 11:37:54 MySQL\_Session.cpp:49:kill\_query\_thread(): [WARNING] KILL QUERY 138 on 192.168.1.4:3306

2016-09-02 11:37:54 MySQL\_Session.cpp:49:kill\_query\_thread(): [WARNING] KILL QUERY 134 on 192.168.1.4:3306

可以看到在mysql-default\_query\_timeout=30000，30s之后proxysql确实kill了超时的语句。

30s这个时间可以通过执行‘service iptables restart’的时间和proxysql.log里日志的时间来判定，这个设置30s是有效的

* + 1
  + 2
  + 3
  + 4
  + 5
  + 6
  + 7
  + 8
  + 1
  + 2
  + 3
  + 4
  + 5
  + 6
  + 7
  + 8

然后，直接对后端node（db1）进行同样的测试，发现结果是一样的，sysbench等待很长时间依然无法完成也无报错。   
从这一点似乎可以判定在这个测试方法下，未达到预期结果的原因可能不在proxysql，而在sysbench本身。

通过对重启iptables之后的通信进行抓包（分别对针对proxysql和mysql在这个测试场景下进行抓包）,命令如下：   
~]# date; service iptables restart; tcpdump -i em2 host 192.168.1.35 and port 3306 and host not 192.168.1.10 -w /tmp/sysbench-proxysql-network-issue.pacp   
~]# date; service iptables restart; tcpdump -i em2 host 192.168.1.34 and port 3306 and host not 192.168.1.10 -w /tmp/sysbench-proxysql-network-issue.pacp

发现，sysbench“一直”在重传由于iptables新规则而无法返回的几个请求，所以就成了“无穷尽等待的样子” (atlas 在这个场景下有同样问题)

照理说，proxysql kill掉了一些查询，会返回给sysbench错误，但为什么sysbench并未将错误展示出来呢(可能是因为re-execute机制)

**最后，通过跟作者沟通，发现是由于没开启monitor模块，导致proxysql无法检测到后端出了什么类型的错误，也就无法执行对应各种后端错误的一些操作（之前我是特意关掉了monitor模块）**

* 测试后端链接已满的异常

在‘mysql-monitor\_enabled=false’的前提下，通过将一台slave的max\_connections值调低并通过proxysql以外的链接将其占满，来测试后端链接已满的异常

重启proxysql,将所有统计信息清空 systemctl restart proxysql.service   
(重启后，无论后端真实情况如何，在这个阶段在proxysql里的状态都是ONLINE)   
db1的mysql上设置 set global max\_connections=5;   
(然后在别的tty打开个多到db1 mysql的里链接，直到其报错达到最大链接数)

用上面的sysbench语句测试，因为proxysql并不知道db1的最大连接数只有5，   
所以他会按照他的[**算法**](http://lib.csdn.net/base/datastructure)来分配请求，可以预见，在几次尝试后，proxysql会将db1标记为SHUNNED

最后通过观察sysbench测试结果以及proxysql本身的统计来分析这个过程，截取三次测试中的一次

OLTP test statistics:

queries performed:

read: 140000

write: 0

other: 0

total: 140000

transactions: 0 (0.00 per sec.)

read/write requests: 140000 (10289.40 per sec.)

other operations: 0 (0.00 per sec.)

ignored errors: 0 (0.00 per sec.)

reconnects: 0 (0.00 per sec.)

General statistics:

total time: 13.6062s

total number of events: 10000

total time taken by event execution: 108.7750s

response time:

min: 7.78ms

avg: 10.70ms

\*\*max: 1014.20ms\*\*

approx. 95 percentile: 13.09ms

Threads fairness:

events (avg/stddev): 1250.0000/13.15

execution time (avg/stddev): 13.5969/0.00

real 0m13.624s

user 0m3.952s

sys 0m4.977s

看到response time中max（三次值为1014.20ms 261.59ms 291.91ms），而正常max为233ms（三次均值）

proxysql admin接口，通过下面的sql看到三次预设的异常场景都如期的出现了错误，这里截取其一：

MySQL [(none)]> select \* from stats\_mysql\_connection\_pool;

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

| hostgroup | srv\_host | srv\_port | status | ConnUsed | ConnFree | ConnOK | ConnERR | Queries | Bytes\_data\_sent | Bytes\_data\_recv | Latency\_ms |

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

| 0 | 192.168.1.21 | 3307 | ONLINE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| 1 | 192.168.1.10 | 3306 | ONLINE | 0 | 8 | 8 | 0 | 139982 | 6120100 | 384510898 | 0 |

| 1 | 192.168.1.4 | 3306 | SHUNNED | 0 | 0 | 8 | 12 | 18 | 660 | 14508 | 0 |

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

同样场景针对atlas进行测试，也没有报错出现（atlas和proxysql都是一主/db1/两从/db2 db3/）

* + 1
  + 2
  + 3
  + 4
  + 5
  + 6
  + 7
  + 8
  + 9
  + 10
  + 11
  + 12
  + 13
  + 14
  + 15
  + 16
  + 17
  + 18
  + 19
  + 20
  + 21
  + 22
  + 23
  + 24
  + 25
  + 26
  + 27
  + 28
  + 29
  + 30
  + 31
  + 32
  + 33
  + 34
  + 35
  + 36
  + 37
  + 38
  + 39
  + 40
  + 41
  + 42
  + 43
  + 1
  + 2
  + 3
  + 4
  + 5
  + 6
  + 7
  + 8
  + 9
  + 10
  + 11
  + 12
  + 13
  + 14
  + 15
  + 16
  + 17
  + 18
  + 19
  + 20
  + 21
  + 22
  + 23
  + 24
  + 25
  + 26
  + 27
  + 28
  + 29
  + 30
  + 31
  + 32
  + 33
  + 34
  + 35
  + 36
  + 37
  + 38
  + 39
  + 40
  + 41
  + 42
  + 43

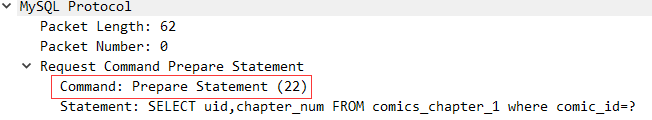
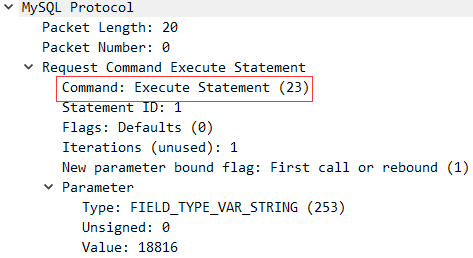
**测试对**prepare**语句的支持**

好多框架用prepare语句来避免SQL注入等安全问题，同时能在MySQL解析查询方面降低一些开销，所以对于prepare语句支持与否也很重要

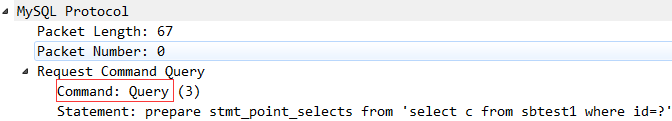
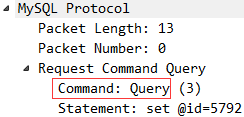
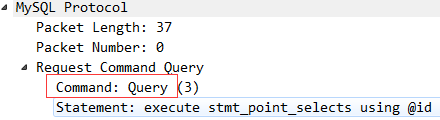
首先要从MySQL协议层面了解prepare语句，分为两种([参考](http://dev.mysql.com/doc/refman/5.6/en/sql-syntax-prepared-statements.html))

* Prepared Statements in Application Programs   
  或者称为BINARY protocol

抓包分析一次prepare,set,execute过程，可以观察到，客户端发送的是[COM\_STMT\_PREPARE](https://dev.mysql.com/doc/internals/en/com-stmt-prepare.html)

* Prepared Statements in SQL Scripts   
  或者称为TEXT protocol

抓包分析一次prepare,set,execute过程，可观察到客户端发送的是[COM\_QUERY](https://dev.mysql.com/doc/internals/en/com-query.html)   
   
   


**关于prepare语句，作者给出的回复和计划如下：**   
这是我在实验1.2.1版本时跟作者的沟通。现在已经发布1.3.2了，从1.3版本开始,两种协议都支持了。**但是在设置字符集方面，对于binary protocol 的prepare set names xxx语句还无法正确处理**（例如，[**PHP**](http://lib.csdn.net/base/php)的laravel框架默认就是以这种形式设置字符集）

MySQL supports two type of prepared statements:   
**using API**   
**using SQL** Further details here   
SQL support currently not supported either, because PREPARE doesn’t disable multiplexing, so it is possible that ProxySQL sends PREPARE in a connection, and EXECUTE into another.

SQL support will be fixed in v1.2.3 , see [#684](https://github.com/sysown/proxysql/issues/684) .   
API support is planned in ProxySQL 1.3 .

**五、connection-pool和multiplexing（并对比Atlas）**

首先来精确理解一下这两个词，依作者回复

They are very related to each other   
Connection pool is a cache of connections that can be reused.   
Multiplexing is a technique to reuse these connections.

**测试场景：**

10条`select \* from test.test\_table`，10条`select @@hostname`；

ProxySQL/Atlas IP 192.168.1.35；两个读节点IP分别为192.168.1.37和192.168.1.38；

每次测试完之前重启ProxySQL/Atlas；

分两次测试，第一次测试脚本如下（每条命令一次连接）

!#/bin/sh

for i in {1..10};do

mysql -uuser -p'passwd' -P6033 -h192.168.1.35 -e "select @@hostname;" #select @xxx，会禁用multiplexing

done

for i in {1..10};do

mysql -uuser -p'passwd' -P6033 -h192.168.1.35 -e "select id from test.test\_table;" #普通查询

done

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 1
* 2
* 3
* 4
* 5
* 6
* 7

第二次测脚本如下（一次连接，执行所有命令）

for i in {1..10};do

query1 += 'select @@hostname;' #select @xxx，会禁用multiplexing

query2 += 'select id from test.test\_table;' #普通查询

done

echo $query1$query2 | mysql -uuser -p'passwd' -P6033 -h192.168.1.35

* 1
* 2
* 3
* 4
* 5
* 1
* 2
* 3
* 4
* 5

**对ProxySQL进行测试分析**   
通过tcpdump抓包wireshark分析，通过ProxySQL对20条查询的路由情况及其和后端MySQL之间的建立连接情况（通过ProxySQL上的原端口号）来分析连接池和禁用multiplexing的情况。结果汇总如下

第一次测试（每条命令一次连接）

select @xxx #会禁用multiplexing

1.35源端口

1.35---->1.37 42094 42096 42097 42099 42102 （转发5次）

1.35---->1.38 37971 37974 37976 37977 37979 （转发5次）

普通查询

1.35源端口

1.35---->1.37 42105 （转发3次）

1.35---->1.38 37980 （转发7次）

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

第二次测试（一次连接，执行所有命令）

select @xxx #会禁用multiplexing

1.35源端口

1.35---->1.37 （转发0次）

1.35---->1.38 37817 （转发10次）

普通查询

1.35源端口

1.35---->1.37 （转发0次）

1.35---->1.38 37817 （转发10次）

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

**对比来看Atlas的分时分析**   
第一次测试（每条命令一次连接）

select @xxx

1.35源端口

1.35---->1.37 （转发0次）

1.35---->1.38 38405 38407 38409 38411 38413 （转发10次）

38415 38417 38419 38421 38423

普通查询

1.35源端口

1.35---->1.37 （转发0次）

1.35---->1.38 38385 38387 38389 38391 38393 （转发10次）

38395 38397 38399 38401 38403

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

第二次测脚本如下（一次连接，执行所有命令）

select @xxx

1.35源端口

1.35---->1.37 42435 （转发5次）

1.35---->1.38 38312 （转发5次）

普通查询

1.35源端口

1.35---->1.37 42435 （转发5次）

1.35---->1.38 38312 （转发5次）

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

由上面测试分析结果可以明显的看到

ProxySQL的负载均衡策略是基于权重的轮询，但并不是严格的逐条轮询。而且可以看到在一次连接之内：由于某些语句（select @xx 或者prepare语句）导致ProxySQL自动关闭multiplexing之后，在本次链接之后的所有语句都会被路由到同一台MySQL

atlas做的仅仅是轮询转发，他不会去区分查询类型（例如有些查询是要路由到后端唯一的MySQL）

而且，在`第一次测试（每条命令一次连接）`中，atlas将所有的20条请求都路由到了1.38这台MySQL，并且每次都要新建连接（并没有用到其连接池）

**六、proxysql对后端server健康检查**

1. 可以主动
2. 来看一下其相关参数：
3. | mysql-monitor\_enabled | true |
4. | mysql-monitor\_history | 600000 |
5. | mysql-monitor\_connect\_interval | 120000 |
6. | mysql-monitor\_connect\_timeout | 200 |
7. | mysql-monitor\_ping\_interval | 60000 |
8. | mysql-monitor\_ping\_max\_failures | 3 |

| mysql-monitor\_ping\_timeout | 100 |

* + 1
  + 2
  + 3
  + 4
  + 5
  + 6
  + 7
  + 8
  + 1
  + 2
  + 3
  + 4
  + 5
  + 6
  + 7
  + 8

这两种检测的区别，据作者回复和抓包分析，总结如下：

**Ping** is done using [mysql\_ping()](http://dev.mysql.com/doc/refman/5.6/en/mysql-ping.html)   
通过抓包分析，是通过现有的链接（连接池中）发送一个    Request Ping语句   
**Connect** is done using [mysql\_real\_connect()](http://dev.mysql.com/doc/refman/5.6/en/mysql-real-connect.html)   
这是一个客户端到服务器    建立链接，登录，退出登录，关闭链接的完整过程

这两个函数的返回值不同，可以帮助proxysql理解与后端链接除了哪些问题。

**模拟场景，验证以上各设置并加深理解其故障检测机制：**   
两个前提

* + 在web1和web5在proxysql均为online状态下；
  + 无客户端访问ProxySQL，即只验证monitor模块的行为

修改web1 MySQL配置文件中max\_connections = 3；重启web1 MySQL，在其他tty打开几个MySQL链接以确保proxysql无法链接该MySQL。同时在proxysql server上打开抓包功能tcpdump -i em2 host 192.168.1.4 and port 3306 -w /tmp/web\_shun.pcap，然后通过wireshark对比上面各值进行分析。   
[附件：数据包文件](http://note.youdao.com/noteshare?id=90e238cc510f711d74caddfbd5c01685)

通过分析可验以以下设置的有效性

mysql-monitor\_connect\_interval

mysql-monitor\_ping\_interval

mysql-monitor\_ping\_max\_failures

mysql-shun\_recovery\_time\_sec

mysql-ping\_interval\_server\_msec

* + 1
  + 2
  + 3
  + 4
  + 5
  + 1
  + 2
  + 3
  + 4
  + 5

1. 可以被动

被动就是将全局变量‘mysql-monitor\_enabled’置为false，这种情况下，后端server故障后，proxysql不会主动探知，而是在有请求被“正常”路由到该server之后才会在runtime层更改该server状态为‘SHUNNED’ 或者重新变为‘ONLINE’。这个过程，应用无感知（表现为mysql-client命令和sysbench均无报错）   
相关变量为

| mysql-shun\_on\_failures | 5 |

| mysql-shun\_recovery\_time\_sec | 60 |

| mysql-query\_retries\_on\_failure | 1 |

| mysql-connect\_retries\_on\_failure | 5 |

| mysql-connect\_retries\_delay | 1 |

| mysql-connection\_max\_age\_ms | 0 |

| mysql-connect\_timeout\_server | 1000 |

| mysql-connect\_timeout\_server\_max | 10000 |

* + 1
  + 2
  + 3
  + 4
  + 5
  + 6
  + 7
  + 8
  + 1
  + 2
  + 3
  + 4
  + 5
  + 6
  + 7
  + 8

**七、关于proxysql和mysql中的最大连接数**

首先明确下MySQL中的最大连接数由max\_connections变量控制，proxysql中的最大连接数有两个方面的设置mysql\_users.max\_connections和mysql\_servers.max\_connections

**下面就直接说下我的结论** [摘自我github上的issue](https://github.com/sysown/proxysql/issues/969)

1. if the mysql\_servers.max\_connections is reached, some of the connections will wait until the mysql-connect\_timeout\_server\_maxis reached. then proxysql will return error messageSQLSTATE[HY000]: General error: 9001 Max connect timeout reached while reaching hostgroup 1 after 10000ms.
2. if the mysql\_users.max\_connections is reached, then the client will see the error message 1040: Too many connections
3. If backend’s global variable ‘max\_connections’ is reached and proxysql has no ConnFree with the backend, then client can accomplish the connection with proxysql but all queries will return ERROR 1040 (#HY00): Too many connections, and the monitor will consider this backend is shun and will be loged to the proxysql.log

**八、据我所知的bug和不足**

**bug：**   
1. 对于prepare语句（Prepared Statements in Application Programs，见上文解释。例如laravel框架的prepare语句），如果在backend上更改了表结构，在一些情况下会导致proxysql返回如下错误   
SQLSTATE[HY000]: General error: 2057 A stored procedure returning result sets of different size was called. This is not supported by libmysql  
2. 对于prepare语句的set names xxx语句还不能有效处理   
3. 对于非prepare语句的set names xxx collate xxxx还不能有效处理   
**不足：**   
1. 前后端账号分离：可以多个frontend用户对应一个或少数banckend用户，简化后端MySQL上的授权操作，尤其是项目多而且之间库表关联较紧密时；还能将前端用户和stats中相关表或者错误日志关联，非常方便的在众多接入到ProxySQL的项目中定位到异常项目。   
2. 错误日志中对于SQL类错误只记录了SQL语句和MySQL返回的错误信息，没有关联到用户和库表，在众多项目中仅凭一条SQL语句去寻找源头难度可想而知（其实这个跟第一条是有关联的）。   
3. 通过admin接口进行管理时，在作出多处修改时，无法一次性将修改应用到内存，也无法一次性将所有修改保存至磁盘。例如对mysql\_servers, mysql\_users,mysql\_query\_rules都做了修改，就得分别使这三个方面的更改生效和保存至磁盘。反过来同理，假如做了一些更改但是最终想丢弃这些更改也无法方便的将其回复到修改之前的状态。

**总结**

**稳定性方面：**目前我们已经部分业务切换到了ProxySQL上，运行一直稳定，未遇到cpu负载高或者占用内存多等情况。

**运维/DBA友好：**借助于ProxySQL的错误日志，我们发现了之前没注意到的一些SQL问题（如主键重复类的问题等）。还通过stats库里的相关表定位了一些问题。

**性能方面：**也强于atlas（[参见文章](http://blog.csdn.net/kai404/article/details/52832848)）

个人觉得，解决掉上述几点bug和不足的话（且不说可能会出现的其他feature），ProxySQL就会更加强大和完美。