New kid on the block

Introduction to PostgreSQL Support in ProxySQL 3

Find me @peppla
and
pep.pla@percona.com

I am Pep Pla

Father of three kids and three cats. In my spare time I'm a Consultant at Percona.

Disclaimer

I'm not a ProxySQL employee.
Percona is not owner or owned by ProxySQL.

I think ProxySQL is a great open source product and I celebrate that they are adding support for the PostgreSQL protocol.

Pgsql support in ProxySQL is still WIP.

What is ProxySQL?

ProxySQL is an open source **high performance**, **high availability**, **database protocol aware** proxy for MySQL and **PostgreSQL**.

Why ProxySQL?

- Advanced features:
  - high availability.
  - load balancing.
  - query based routing.
  - query caching.
  - query rewriting.
  - connection multiplexing.
- Single proxy platform for MySQL and PostgreSQL.
  - Infrastructure simplification.

ProxySQL Architecture Overview

- ProxySQL is a multithreaded daemon with a modular architecture.
  - Main thread.
  - Admin thread.
  - Worker threads.
  - Monitor threads.

ProxySQL Architecture Overview

- Main thread
  - bootstraps and manages other threads.
- Admin thread:
  - Initializes and bootstraps the admin interface.
  - Configuration load from file or database.
  - Web UI and CLI interface.
  - Cluster module.
  - Admin/stats listener.

ProxySQL Architecture Overview

- Worker threads
  - Handle database traffic.
  - For client and backend connections.
    - Client: client application -> proxysql
    - Backend: proxysql -> database
  - A client connection is assigned to a worker thread and all traffic for that connection is handled by the same thread.
  - One thread handles multiple client and database connections.

ProxySQL Architecture Overview

Multiplexing in ProxySQL is a feature that allows multiple frontend connections to reuse the same database backend connection.
ProxySQL uses a **thread pool** and by Multiplexing, ProxySQL further reduces the number of resources being allocated and managed by the database backends and by the proxy itself.

Multiple client connections will use the same ProxySQL thread and the same database connection.

If required, ProxySQL will use a **dedicated** database connection.

ProxySQL Architecture Overview

- Monitor threads
  - Main thread: starts and manages monitor threads.
  - Connection check threads.
    - Database availability and connection latency.
  - Ping check threads.
    - If the server is available.
    - Latency tracking.
  - Read-only check threads.
  - Replication lag check threads.
- Monitor thread pool
  - if required, additional threads will be started.

ProxySQL Configuration Architecture

- Multilayer configuration system.
  - Runtime
  - Memory
  - Disk
- Accessed via the Admin Interface.
  - SQL-like interface.

Runtime Configuration

- Runtime represents in-memory configuration.
- Used by threads to perform their tasks: configure, process traffic and monitor servers.
- Can't be modified directly.

Memory Configuration

- Also know as main or "working" configuration.
- It is stored in memory and is not persistent.
- Can be modified via the Admin Interface.

Disk Configuration

- Persistent configuration.
  - Stored in a file or in the internal SQLite database.
- Read at startup and used to populate the Memory and Runtime configurations.

Configuration flow

1. Changes are performed in Memory.
2. Once we are ready, we copy them to Runtime.
3. Once we know changes work properly, we save them to Disk.

If something breaks, I can move the configuration from Disk to Memory or from Runtime to Memory if needed.

Initial Configuration

- When there is no existing configuration, ProxySQL will read from the initial configuration file.
- This configuration will be saved to the indexed internal SQLite database.
- The initial configuration file is only read once, when there is no existing configuration.

It is a common mistake to change the configuration in the initial configuration file and try to reload it by restarting ProxySQL.

Why this multilayer configuration?

- Changes are made to the Memory configuration.
  - Multiple changes can be made without affecting the Runtime configuration.
- Changes are applied from Memory to Runtime.
  - All changes are applied at once.
  - No partial changes.
- If everything is fine, changes can be saved to disk for persistence.
- If something goes wrong, changes can be reverted by reloading the Runtime configuration from Disk.
- Variable validation only takes place when changes are applied to Runtime.
  - Prevents invalid configuration changes.
  - Previous value will be kept if the new value is invalid.

More about configuration

- Configuration is split into multiple tables/groups.
  - Each table/group contains related configuration variables.
- Configuration groups:
  - Users
  - Backend Servers
  - Query rules
  - Protocol variables
  - Admin variables
  - Scheduler

The Admin interface

```
psql -h 127.0.0.1 -p6132 -U admin -d admin
```

- Connect with psql client.
- Meta-commands are not supported.
- SQLite.
- https://proxysql.com/documentation/getting-started/

The Admin interface

```
[root@pgday1 ~]# psql -h 127.0.0.1 -p6132 -U admin -d admin
Password for user admin:
psql (17.6 - Percona Server for PostgreSQL 17.6.1, server 16.1)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off, ALPN: none)
Type "help" for help.

admin=# show databases;
seq |     name      |                file
-----+---------------+-------------------------------------
0   | main          |
2   | disk          | /var/lib/proxysql/proxysql.db
3   | stats         |
4   | monitor       |
5   | stats_history | /var/lib/proxysql/proxysql_stats.db
(5 rows)
```

The Admin interface

```
admin=# show tables from monitor;
              tables
-------------------------------------
...
pgsql server connect log
pgsql server ping log
pgsql_server_read_only_log
...
(13 rows)
```

The Admin interface

```
admin=# show tables like 'pgsql%';
                 tables
-------------------------------------------
pgsql servers
pgsql users
pgsql ldap mapping
pgsql query rules
pgsql query rules fast routing
pgsql hostgroup attributes
pgsql replication hostgroups
pgsql firewall whitelist users
pgsql firewall whitelist rules
pgsql firewall_whitelist_sqli_fingerprints
(10 rows)
```

Configuration Groups: Global variables

```
admin=# select * from global variables where variable name not like 'mysql%' order by variable_name;
                 variable name                  |              variable value
------------------------------------------------+-------------------------------------------
admin-admin credentials                         | admin:admin
admin-checksum admin variables                  | true
admin-checksum ldap variables                   | true
admin-checksum mysql query rules                | true
admin-checksum mysql servers                    | true
admin-checksum mysql users                      | true
admin-checksum_mysql_variables                  | true
...
```

Configuration Groups: Global variables

```
pgsql-throttle connections per sec to hostgroup   | 1000000
pgsql-throttle max bytes per second to client     | 0
pgsql-throttle ratio server to client             | 0
pgsql-unshun algorithm                            | 0
pgsql-use tcp keepalive                           | true
pgsql-verbose query error                         | false
pgsql-wait_timeout                                | 28800000
(194 rows)
```

Not all the variables are dynamic. But most of them are.

Configuration Groups: Global variables

```
admin=# update global_variables set variable_value='PGDayNap25' where variable_name='pgsql-monitor_password';
UPDATE 1
admin=# save pgsql variables to disk;
INSERT 0 144
admin=# load pgsql variables to runtime;
LOAD
```

- WRONG, first test to runtime, later save to disk.
- Validation takes place when setting to runtime.
- save/load **pgsql** variables to runtime/memory/disk
- save/load **admin** variables to runtime/memory/disk.

```
2025-09-24 22:49:16 Admin FlushVariables.cpp:217:flush GENERIC variables  process__database_to_runtime(): [WARNING]
Impossible to set variable threads with value "-1". Resetting to current "4".
```

Error log

- By default ProxySQL generates and error log file in the [datadir]
- This file is important because it contains errors from apparently successful configuration commands.

Servers and Hostgroups

- **Servers** represent backend connection routes.
- **Hostgroups** represent groups of servers that belong to the same topology.
  - Actually hostgroups go in pairs: **Writer** and **reader**.
- Servers in the same hostgroup can receive traffic for that hostgroup.
  - If the server is **read-only** will receive traffic for the **reader** hostgroup.
  - If the server is **read-write** will receive traffic for the **writer** hostgroup.

Servers and Hostgroups

```
select * from pgsql replication hostgroups;
writer hostgroup | reader hostgroup | check type | comment
-----------------+-----------------+------------+---------
1                | 2                | read_only  |
(1 row)


select * from pgsql servers;
 hostgroup id |    hostname     | port | status | ...
--------------+-----------------+------+--------+ ...
1             | 192.168.10.106 | 5432 | ONLINE | ...
1             | 192.168.10.127 | 5432 | ONLINE | ...
(2 rows)
```

Servers and Hostgroups

```
select * from runtime pgsql replication hostgroups;
writer hostgroup | reader hostgroup | check type | comment
-----------------+------------------+------------+---------
1                | 2                | read_only  |
(1 row)



select hostgroup id,hostname,port,status  from runtime_pgsql_servers;
hostgroup id |    hostname     | port | status
-------------+-----------------+------+--------
1            | 192.168.10.106 | 5432 | ONLINE
2            | 192.168.10.127 | 5432 | ONLINE
(2 rows)
```

192.168.10.127 is a hot standby and ProxySQL moves it to the reader hostgroup.

Configure servers and hostgroups

- No mystery here.
  - Insert/Update/delete **pg_servers** and **pg_replication_hostgroups**.
  - **Load pgsql servers to runtime** and/or **save pgsql servers to disk**.
- Servers have more attributes:
  - Hostgroup.
  - ONLINE/OFFLINE SOFT/OFFLINE HARD/SHUNNED.
  - Maximum number of connections.
  - Weight.
  - Maximum replication lag.
  - Maximum ping latency.
- A physical server can be in more than one ProxySQL server.
  - PK is Hostgroup, Address and Port.

Users

```
select * from pgsql_users;
-[ RECORD 1
]----------+-------------------------------------------------------------------------------------------------------------------------
-------------------------
username                | appuser
password                | appuser
active                  | 1
use ssl                 | 0
default hostgroup       | 1
transaction persistent | 1
fast forward            | 0
backend                 | 1
frontend                | 1
max connections         | 10000
attributes              |
comment                 |
```

The missing part. We need users to connect.
Transaction persistence is implemented by user.

Query rules

- **Query rule** can
  - modify the hostgroup.
  - modify the query.
  - cache the results.
  - timeout the query after an specified time.
  - delay the execution of the query.
  - any combination of the previous.
  - ...
- Based on multiple criteria.
- Are processed in order.
  - Can be chained.

Query rules

```
select * from pgsql_query_rules;
-[ RECORD 1 ]---------+--------------------
rule_id               | 1
active                | 1
username              |
database              |
flagIN                | 0
client_addr           |
proxy_addr            |
proxy_port            |
digest                | 0x4539c37ccf80433c
match_digest          |
match_pattern         |
negate_match_pattern  | 0
re_modifiers          | CASELESS
flagOUT               |
replace_pattern       |
destination_hostgroup |
cache_ttl             | 10000
cache_empty_result    |
cache_timeout         |
reconnect             |
timeout               |
retries               |
delay                 |
next_query_flagIN     |
mirror_flagOUT        |
mirror_hostgroup      |
error_msg             |
OK_msg                |
sticky_conn           |
multiplex             |
log                   |
apply                 | 1
attributes            |
comment               |
```

Read/Write splitting

```
INSERT INTO pgsql query rules (rule id, active, match_pattern, destination_hostgroup, apply) VALUES
(1, 1, '^SELECT.*FOR UPDATE', 1, 1),
(2, 1, '^SELECT', 2, 1);
INSERT 0 2

load pgsql query rules to runtime;
LOAD

psql -h proxysql1 -p 6133 -U appuser -d application_db
Password for user appuser:
psql (17.6 - Percona Server for PostgreSQL 17.6.1, server 16.1)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off, ALPN: none)

application db=> SELECT inet_server_addr();
inet server addr
------------------
192.168.10.127
(1 row)

application db=> SELECT inet_server_addr() for update;
inet server addr
------------------
192.168.10.106
(1 row)
```

Read/Write splitting

```
application db=> start transaction;
START TRANSACTION
application db=*> SELECT inet_server_addr();
inet server addr
------------------
192.168.10.106
(1 row)

application_db=*> commit;
COMMIT
application db=> SELECT inet_server_addr();
inet server addr
------------------
192.168.10.127
(1 row)
```

Results caching

```
select hostgroup, database, digest, digest_text from  stats.stats_pgsql_query_digest where digest_text='select now();';
hostgroup |    database    |       digest       |  digest_text
----------+----------------+--------------------+--------------
2         | application_db | 0x4539c37ccf80433c | select now();
1         | application_db | 0x4539c37ccf80433c | select now();

INSERT INTO pgsql_query_rules (rule_id, active, digest, cache_ttl, apply) VALUES
(3, 1,'0x4539c37ccf80433c', 10000, 1);
load pgsql query rules to runtime;

application_db=> select now();
              now
-------------------------------
2025-09-23 22:22:18.977455+00
(1 row)

application_db=> select now();
              now
-------------------------------
2025-09-23 22:22:18.977455+00
(1 row)
```

Results caching

```
A few seconds later...

application db=> select now();
              now
-------------------------------
2025-09-23 22:23:53.486641+00
(1 row)

application db=> select now(),1;
              now              | ?column?
-------------------------------+----------
2025-09-23 22:24:03.669498+00 |        1
(1 row)

application db=> select  now();
              now
-------------------------------
2025-09-23 22:24:10.692627+00
(1 row)
```

Query rewriting

```
INSERT INTO pgsql query rules (rule id, active, match_pattern, replace_pattern, apply) VALUES
(2, 1,'select ([0-9])', 'select \1+1', 1);
load pgsql query rules to runtime;

application_db=> select 1;
?column?
----------
       2
(1 row)


application_db=> select 9;
?column?
----------
      10
(1 row)

application_db=> select  1;
?column?
----------
       1
(1 row)
```

Query rule chaining

- Query rules have three attributes that define chaining:
  - **apply**: if 1 and the rule matches, it will be applied without further processing.
  - Rules with lower **flag_in=0** are processed first.
  - If **apply=0** and **flag_in=flag_out**, I will continue processing **flag_in** rules.
  - If **apply=0** and **flag_in!=flag_out**, I will switch processing to rules with **flag_in=flag_out**
- Query rule chaining is used to reduce the impact of processing many rules.
  - Query rules have a cost.
  - Processed for each and every query.

Query rule fast routing

- The **pgsql_query_rules_fast_routing** table is an extension of **pgsql_query_rules** and is evaluated after that.
- Only filters by **username**, **database** and **flagin**.

Monitor

- ProxySQL has dedicated threads to check latency, replication lag and if the database is writable.
- These threads update the **monitor** database with information about the checks performed.
- Monitor data expires after **pgsql-monitor_history** milliseconds.
- The frequency of monitoring can be changed.

Monitor database connections

```
select * from pgsql server connect log limit 10;
    hostname    | port |  time start us    | connect success time us | connect error
----------------+------+-------------------+-------------------------+---------------
 192.168.10.106 | 5432 | 1758747992249281 | 6330                    |
 192.168.10.127 | 5432 | 1758747992249211 | 6638                    |
 192.168.10.106 | 5432 | 1758748112249395 | 4764                    |
 192.168.10.127 | 5432 | 1758748112249348 | 6598                    |
 192.168.10.127 | 5432 | 1758748232249432 | 6376                    |
 192.168.10.106 | 5432 | 1758748232249597 | 6449                    |
 192.168.10.106 | 5432 | 1758748352249681 | 5356                    |
 192.168.10.127 | 5432 | 1758748352249628 | 6503                    |
 192.168.10.106 | 5432 | 1758748472249610 | 6988                    |
 192.168.10.127 | 5432 | 1758748472249651 | 8154                    |
(10 rows)
```

Monitor ping

```
admin=# select * from pgsql server ping log limit 10;
    hostname    | port |  time start us   | ping success time us | ping error
----------------+------+------------------+----------------------+------------
 192.168.10.106 | 5432 | 1758748281809717 | 291                  |
 192.168.10.127 | 5432 | 1758748281809677 | 680                  |
 192.168.10.106 | 5432 | 1758748289810143 | 331                  |
 192.168.10.127 | 5432 | 1758748289810061 | 639                  |
 192.168.10.106 | 5432 | 1758748297810390 | 327                  |
 192.168.10.127 | 5432 | 1758748297810383 | 733                  |
 192.168.10.106 | 5432 | 1758748305810542 | 270                  |
 192.168.10.127 | 5432 | 1758748305810514 | 553                  |
 192.168.10.106 | 5432 | 1758748313810730 | 278                  |
 192.168.10.127 | 5432 | 1758748313810757 | 604                  |
10 rows
```

Monitor read only

```
select * from pgsql_server_read_only_log limit 10;
    hostname    | port |  time_start_us   | success_time_us | read_only | error
----------------+------+------------------+-----------------+-----------+-------
 192.168.10.106 | 5432 | 1758748338194404 | 300             | 0         |
 192.168.10.127 | 5432 | 1758748338194368 | 626             | 1         |
 192.168.10.106 | 5432 | 1758748339194863 | 454             | 0         |
 192.168.10.127 | 5432 | 1758748339194915 | 702             | 1         |
 192.168.10.106 | 5432 | 1758748340195358 | 300             | 0         |
 192.168.10.127 | 5432 | 1758748340195401 | 548             | 1         |
 192.168.10.106 | 5432 | 1758748341195676 | 319             | 0         |
 192.168.10.127 | 5432 | 1758748341195618 | 584             | 1         |
 192.168.10.106 | 5432 | 1758748342196254 | 376             | 0         |
 192.168.10.127 | 5432 | 1758748342196369 | 590             | 1         |
(10 rows)
```

Statistics

- ProxySQL collects multiple types of statistics.

Statistics

```
show tables from stats;
              tables
---------------------------------------
stats pgsql client host cache
stats pgsql client host cache_reset
stats pgsql commands counters
stats pgsql connection pool
stats pgsql connection_pool_reset
stats pgsql errors
stats pgsql errors reset
stats pgsql free connections
stats pgsql global
stats pgsql processlist
stats pgsql query digest
stats pgsql query digest_reset
stats pgsql query_rules
stats pgsql users
stats proxysql message metrics
stats proxysql message metrics reset
stats proxysql servers checksums
stats proxysql servers clients_status
stats proxysql servers metrics
stats proxysql_servers_status
(39 rows)
```

Statistics

```
admin=# select * from stats_pgsql_query_rules;
 rule id | hits
---------+------
 1       | 5
 2       | 10
 4       | 0
 5       | 75
(4 rows)
```

The bad news is that

"The PostgreSQL module is still in beta, and users should be aware that it is still undergoing active development and testing."

- PostgreSQL's Extended Query Protocol Support in ProxySQL - https://github.com/sysown/proxysql/issues/5018
- Mirroring is not supported yet.
- Additional checks (replica lag).
- Different backend and frontend users.
- Local password hashing.
- ProxySQL Cluster support.
- Persistent statistics.

The good news is that

- ProxySQL is committed to PostgreSQL.
- Missing functionalities are under development right now.

How does ProxySQL compare to the others?

- ProxySQL is intended to run on the application server.
  - HA is solved at application server level.
  - No need for specific/dedicated infrastructure.
- It is fast.
- Protocol aware.
- Distributed configuration
  - not yet for pgsql but trivial to implement.

Thank you!

Questions?