pg_stat_monitor:

A feature-rich and enhanced version of pg_stat_statments

PERCONA

Databases run better with Percona

Content for today

- About
  - Percona
  - Who Am I?
- What is the problem?
- What is pg_stat_monitor?
- How does it differ from pg_stat_statements?
- PMM? QAN? PGSM?
- How do I start, report a bug or contribute?

About Percona

About Percona

Founded in 2006

Widely recognized as champions of unbiased open source database solutions

346 staff worldwide

Offerings

Software (PostgreSQL, MySQL, MongoDB)

Consulting

Managed Services

Support 24x7

Training

Who Am I?

Kai Wagner - <kai.wagner@percona.com>

- Open Source enthusiast since the beginning
- I've been a
  - Datacenter administrator
  - Network/Storage administrator
  - Consultant/Pre-Sales support
  - Developer - not a good one
  - ... and now an engineering manager for several years
- Engaged in the Linux and Storage communities, next to PostgreSQL
- Father of two, husband and construction expert after tearing apart our house for five years ;-)

@ImTheKai

https://www.linkedin.com/in/kai-wagner-b1b661152/

What is the problem?

Database Performance Basics

- Most common reason for poor performance are “Bad Queries”
- Users often do not even know they have such queries
- When they may not know the source of such queries
- Or why they are bad and how to fix them

What is pg_stat_monitor?

How did it all start?

- Initial plan was to be an addition to pg_stat_statements and not a replacement
- During the POC and development phase we realized, that this isn't possible due to the nature of PostgreSQL design
- The idea was born to make it a standalone extension that is compatible with pg_stat_statements

What is pg_stat_monitor?

pg_stat_monitor aka PGSM is a query performance observability extension that combines pg_stat_activity, pg_stat_statements and auto_explain to paint a wholistic picture.

It provides:

- Connection and application details [pg_stat_activity]
- Query planning and execution statistics [pg_stat_statements]
- Query execution plan [auto_explain]

What is pg_stat_monitor?

- Query Performance Monitoring extension for PostgreSQL
- Improved insights into
  - Query origins (allows grouping/multidimensional)
    - Are there any queries from origins, that shouldn't be there?
  - Execution
    - Did it fail?
  - Plan Statistics and details
    - Which plan is used the most?
  - Query Information and metadata
- Stores statistics in configurable time-based units aka buckets
- Significantly improves the observability, enabling the users to debug and tune their performance
- pg_stat_statements compatibility (can be used as a replacement)
- Open Source and developed by Percona

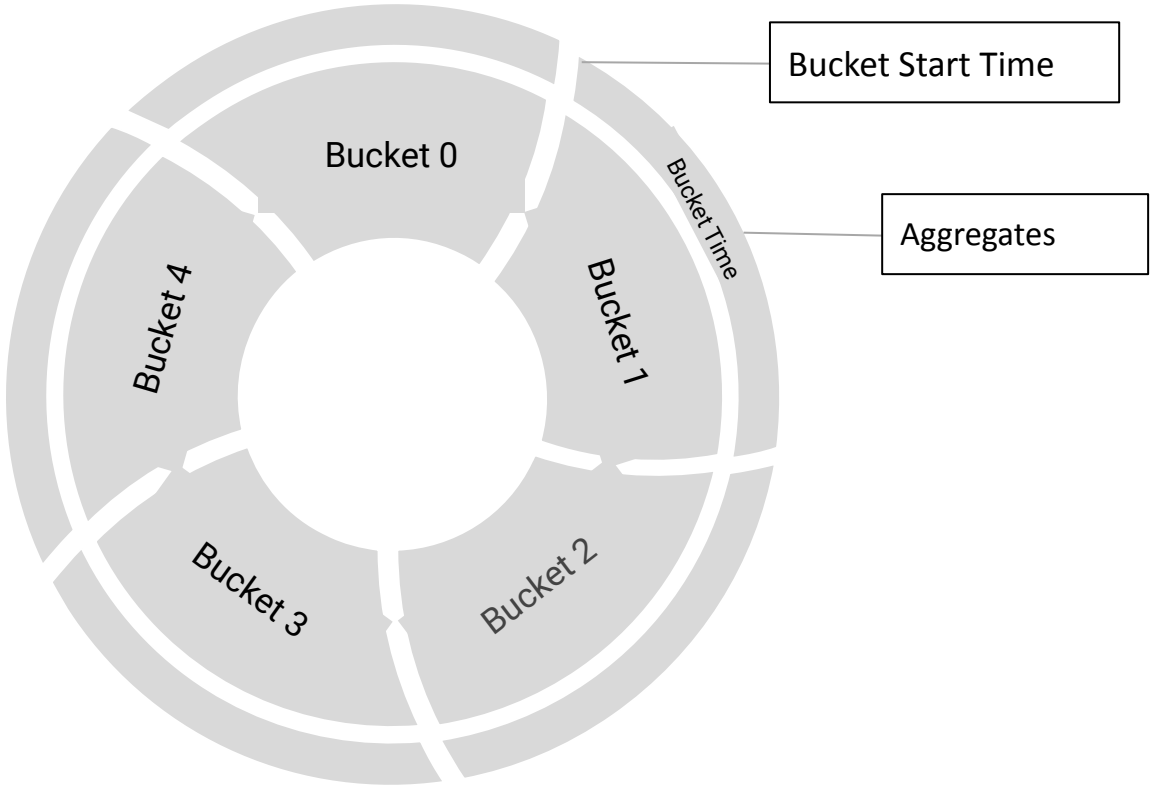
How does it differ from pg_stat_statements?

How does it differ from pg_stat_statements?

- Time Interval Grouping
- Capture Actual Parameters in the Queries
- Query Plan
- Histogram
- ...this is achieved by additional columns

Time Interval Grouping

```
SELECT bucket,
bucket_start_time,
substr(query,0,50)|| '...' AS query,
calls
FROM pg_stat_monitor;
bucket | bucket_start_time | query | calls
-------+---------------------+---------------------------
0 | 2023-06-21 09:30:00 | INSERT INTO pgbench_tell...| 2
1 | 2023-06-21 09:30:30 | INSERT INTO pgbench_bran...| 1
1 | 2023-06-21 09:30:30 | SELECT relname, relkind ...| 1
1 | 2023-06-21 09:30:30 | SELECT queryid, bucket, ...| 1
2 | 2023-06-21 09:31:00 | CREATE table pgbench_his...| 1
2 | 2023-06-21 09:31:00 | SELECT queryid, bucket, ...| 1
2 | 2023-06-21 09:31:00 | SELECT queryid, bucket  ...| 1
```



Capture Actual Parameters in the Queries

```
SET pg_stat_monitor.pgsm_normalized_query = true;
SELECT a FROM foo where a = 10;
a
---
SELECT queryid, substr(query,0,50)|| '...' AS query,calls FROM pg_stat_monitor;
queryid | query | calls
------------------+----------------------------------------------------+-------
55F88A754A1BC5FF | SELECT a from foo where a = $1... | 1

SET pg_stat_monitor.pgsm_normalized_query = false;

SELECT a,2 FROM foo where a = 10;
a | ?column?
---+----------

SELECT queryid, substr(query,0,50)|| '...' AS query,calls FROM pg_stat_monitor;
queryid | query | calls
------------------+----------------------------------------------------+-------
EF380BA0410F35EC | SELECT a,2 from foo where a = 10;... | 1
```

Query Information

```
postgres=# SELECT userid, datname, queryid, substr(query,0, 50) AS query, calls FROM pg_stat_monitor;

userid | datname | queryid | query | calls
---------+----------+------------------+----------------------------------------------------+-------
vagrant | postgres | 939C2F56E1F6A174 | END | 561
vagrant | postgres | 2A4437C4905E0E23 | SELECT abalance FROM pgbench_accounts WHERE aid = | 561
vagrant | postgres | 4EE9ED0CDF143477 | SELECT userid, datname, queryid, substr(query,$1 | 1
vagrant | postgres | 8867FEEB8A5388AC | vacuum pgbench_branches | 1
vagrant | postgres | 41D1168FB0733CAB | select count(*) from pgbench_branches | 1
vagrant | postgres | E5A889A8FF37C2B1 | UPDATE pgbench_accounts SET abalance = abalance + | 561
vagrant | postgres | 4876BBA9A8FCFCF9 | truncate pgbench_history | 1
vagrant | postgres | 22B76AE84689E4DC | INSERT INTO pgbench_history (tid, bid, aid, delta | 561
vagrant | postgres | F6DA9838660825CA | vacuum pgbench_tellers | 1
vagrant | postgres | 214646CE6F9B1A85 | BEGIN | 561
vagrant | postgres | 27462943E814C5B5 | UPDATE pgbench_tellers SET tbalance = tbalance + | 561
vagrant | postgres | 4F66D46F3D4151E | SELECT userid, dbid, queryid, substr(query,0, 50 | 1
vagrant | postgres | 6A02C123488B95DB | UPDATE pgbench_branches SET bbalance = bbalance + | 561
```

Query Information

```
postgres=# SELECT application_name, client_ip, substr(query,0,100) AS query FROM pg_stat_monitor;

application_name | client_ip | query
------------------+-----------+----------------------------------------------------------------------------------------------------
pgbench | 127.0.0.1 | truncate pgbench_history
pgbench | 127.0.0.1 | SELECT abalance FROM pgbench_accounts WHERE aid = $1
pgbench | 127.0.0.1 | UPDATE pgbench_accounts SET abalance = abalance + $1 WHERE aid = $2
pgbench | 127.0.0.1 | BEGIN;
pgbench | 127.0.0.1 | INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES ($1, $2, $3
pgbench | 127.0.0.1 | END;
pgbench | 127.0.0.1 | vacuum pgbench_branches
pgbench | 127.0.0.1 | UPDATE pgbench_tellers SET tbalance = tbalance + $1 WHERE tid = $2
pgbench | 127.0.0.1 | vacuum pgbench_tellers
pgbench | 127.0.0.1 | UPDATE pgbench_branches SET bbalance = bbalance + $1 WHERE bid = $2
pgbench | 127.0.0.1 | select o.n, p.partstrat, pg_catalog.count(i.inhparent) from pg_catalog.pg_
psql    | 127.0.0.1 | SELECT application_name, client_ip, substr(query,$1,$2) as query FROM pg_s
pgbench | 127.0.0.1 | select count(*) from pgbench_branches
(13 rows)
```

Error Logging

```
SELECT decode_error_level(elevel) AS elevel,
sqlcode,
query,
message
FROM pg_stat_monitor
WHERE elevel != 0;

elevel | sqlcode | query | message
--------+----------+-----------------------+---------------------------------
ERROR | 16908420 | SELECT * FROM pg_foo; | relation "pg_foo" does not exist
ERROR | 33816706 | SELECT 1/0; | division by zero
```

Plan Statistics

queryid	plans_calls	rows_retrieved	query	calls
BA6EC88C00347CF7	0	0	truncate table pgbench_...	2
1B12EDE3C70B8F88	100	100	insert into pgbench_tel...	100
3FD5C490B83F760D	0	0	create table pgbench_br...	2
B1B991EE89D61CB6	0	0	create table pgbench_ac...	2
29A99577F1695D28	0	11000000	copy pgbench_accounts ...	2
772222CE7E8765	0	0	alter table pgbench_bra...	2
D59D5F4391AA6B3	10	10	insert into pgbench_bra...	10

Query Plan

```
SELECT substr(query,0,50) AS query_plan FROM pg_stat_monitor LIMIT 10;
query_plan
------------------------------------------------------------------------------+
Index Scan using pgbench_accounts_pkey on pgbench_accounts                    |
   Index Cond: (aid = 102232)                                                 |
Limit                                                                         |
   ->  Subquery Scan on pg_stat_monitor                                       |
         ->  Result                                                           |
               ->  Sort                                                       |
                     Sort Key: p.bucket_start_time                            |
                     ->  Hash Join                                            |
                           Hash Cond: (p.dbid = d.oid)                        |
                           ->  Function Scan on pg_stat_monitor_internal p    |
```

Histogram

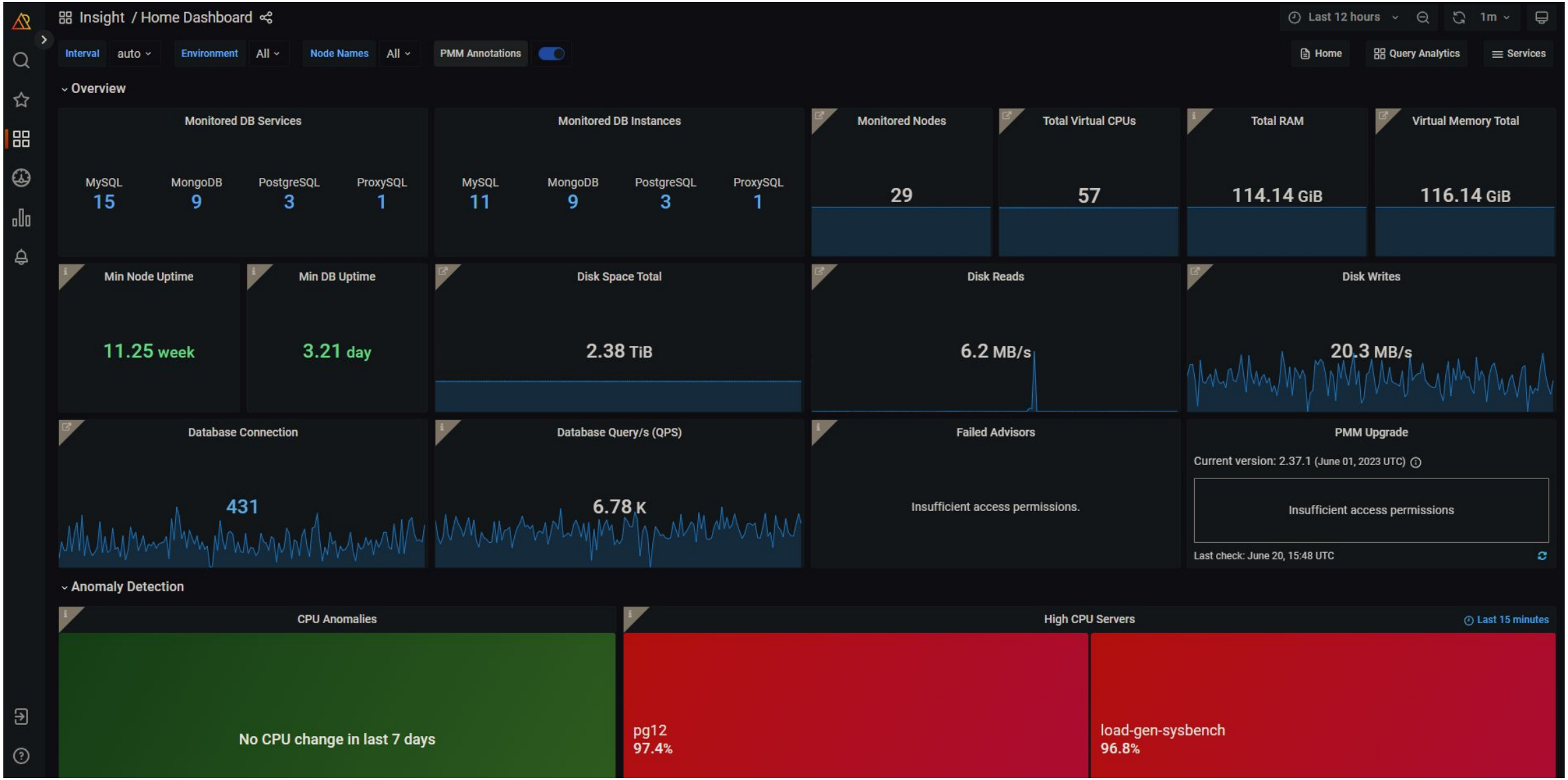
```
SELECT resp_calls, query FROM pg_stat_monitor;
resp_calls | query
-----------------------------------------------+-----------------------------------------------
{1,0,0,0,0,0,0,0,0,0} | select client_ip, query from pg_stat_monitor
{3,0,0,0,0,0,0,0,0,0} | select * from pg_stat_monitor_reset()
{0,0,1,0,0,0,0,0,0,0} | SELECT * FROM foo

SELECT * FROM histogram(0, 'F44CD1B4B33A47AF') AS a(range TEXT, freq INT, bar TEXT);
range | freq | bar
------------------+------+--------------------------------
(0 - 3)} | 2 | ■■■■■■■■■■■■
(3 - 10)} | 0 |
(10 - 31)} | 1 | ■■■■■■■■■■■■■■■■■■■■
(31 - 100)} | 0 |
(100 - 316)} | 0 |
(316 - 1000)} | 0 |
(1000 - 3162)} | 0 |
(3162 - 10000)} | 0 |
(10000 - 31622)} | 0 |
(31622 - 100000)} | 0 |
```

PMM? QAN? PGSM?

What does this even mean and why do I need it?

PMM

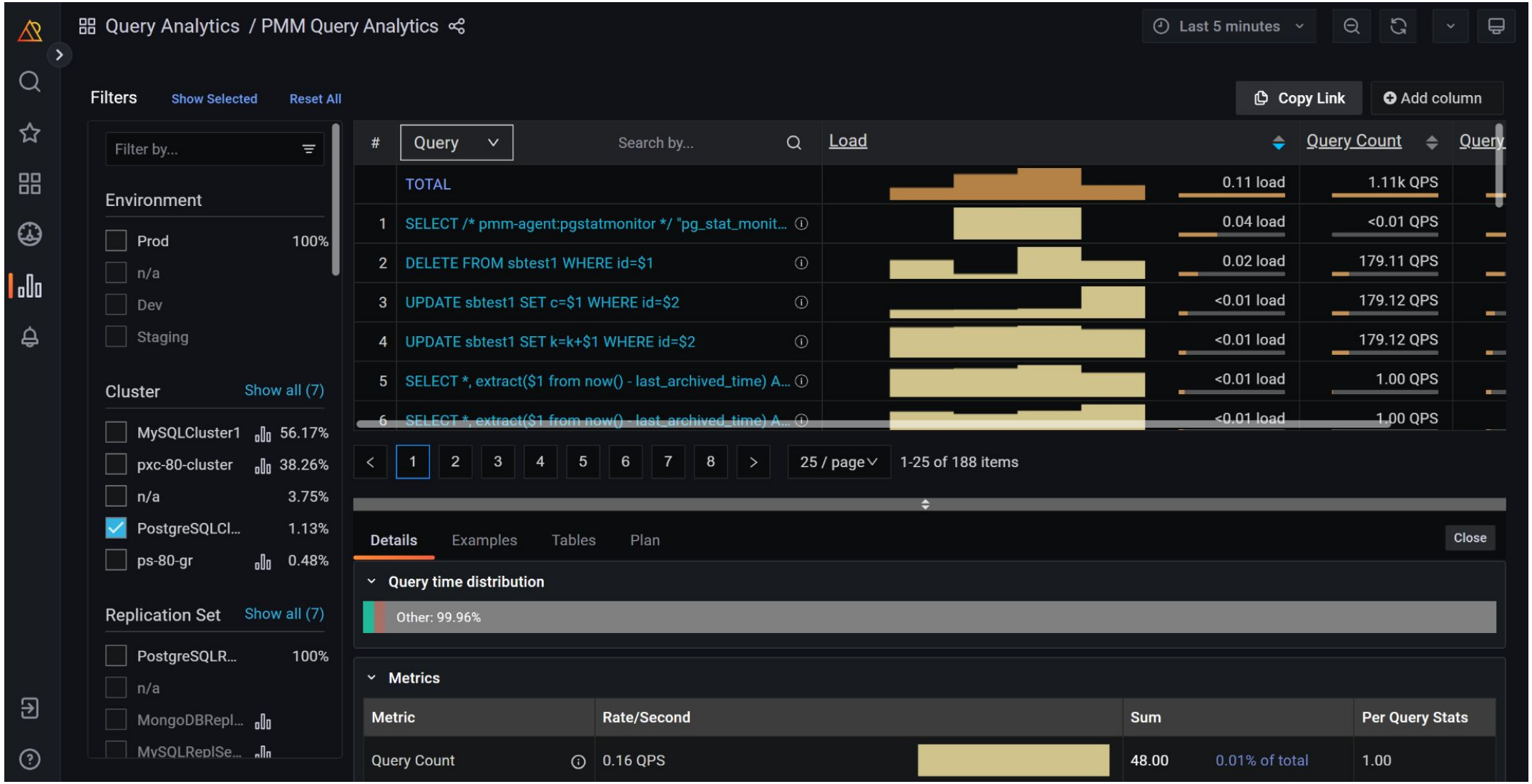
- Percona Monitoring and Management (PMM)
- Open Source database monitoring, management and observability tool for PostgreSQL, MySQL and MongoDB
- PMM collects thousands of out-of-the-box performance metrics from databases and their hosts.
- The PMM web UI visualizes data in dashboards.
- Additional features include advisors for database health assessment
- Live demo available at pmmdemo.percona.com



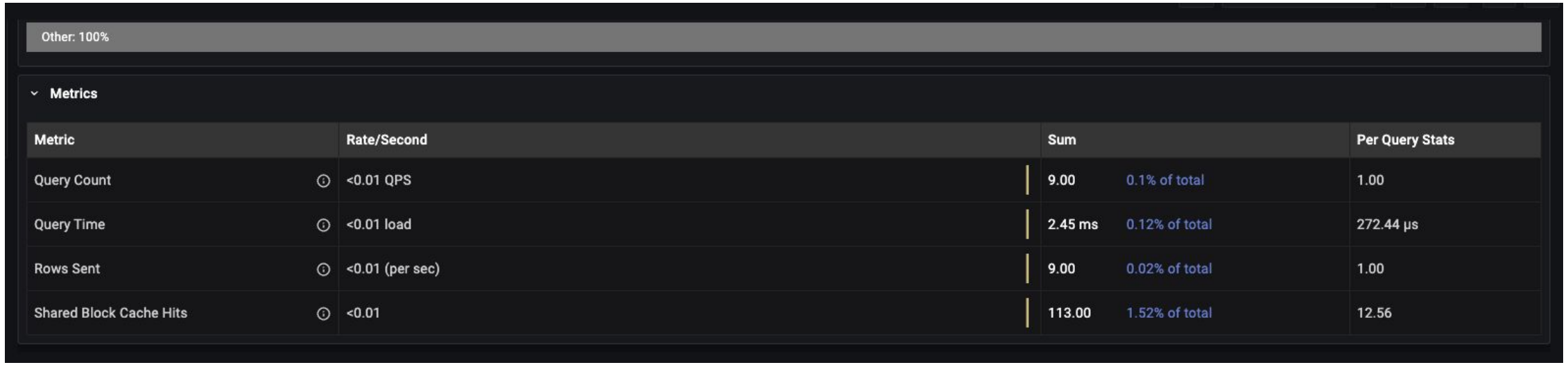
QAN

- QAN is query analytics tool inside the PMM server
- Helping users/customers to identify problems with their specific database
  - Helps to quantify and visualize system (CPU, Disk, etc) impact of queries
- Captures all queries from your DBs
- Analyzes historical data
- It supports all Percona databases (PostgreSQL, MySQL and MongoDB)
- Open source

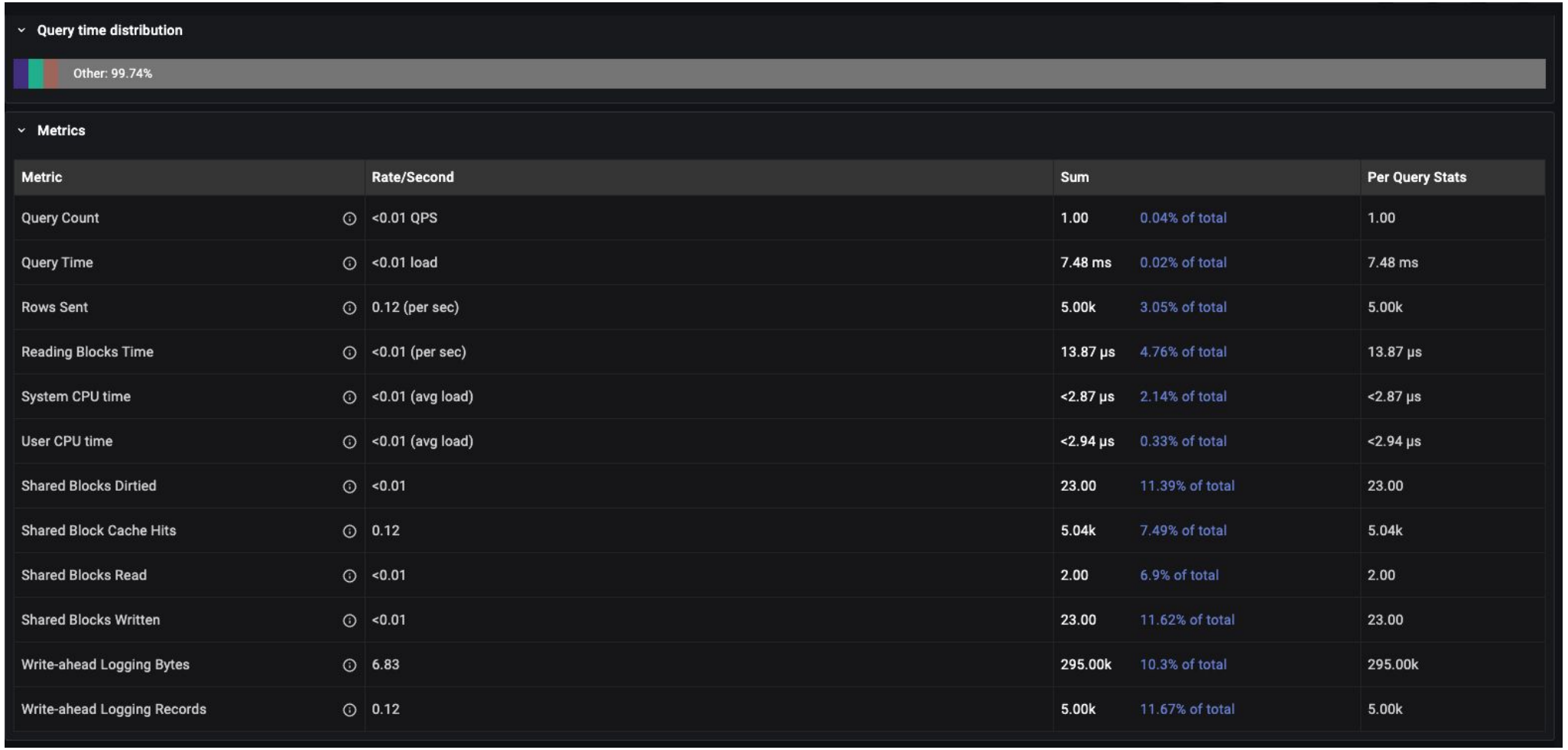
PMM + QAN + PGSM = Insights



pg_stat_statements metrics for insert query



PGSM metrics for insert query



Metric	Rate/Second	Sum		Per Query Stats
Query Count	<0.01 QPS	1.00	0.04% of total	1.00
Query Time	<0.01 load	7.48 ms	0.02% of total	7.48 ms
Rows Sent	0.12 (per sec)	5.00k	3.05% of total	5.00k
Reading Blocks Time	<0.01 (per sec)	13.87 µs	4.76% of total	13.87 µs
System CPU time	<0.01 (avg load)	<2.87 µs	2.14% of total	<2.87 µs
User CPU time	<0.01 (avg load)	<2.94 µs	0.33% of total	<2.94 µs
Shared Blocks Dirtied	<0.01	23.00	11.39% of total	23.00
Shared Block Cache Hits	0.12	5.04k	7.49% of total	5.04k
Shared Blocks Read	<0.01	2.00	6.9% of total	2.00
Shared Blocks Written	<0.01	23.00	11.62% of total	23.00
Write-ahead Logging Bytes	6.83	295.00k	10.3% of total	295.00k
Write-ahead Logging Records	0.12	5.00k	11.67% of total	5.00k

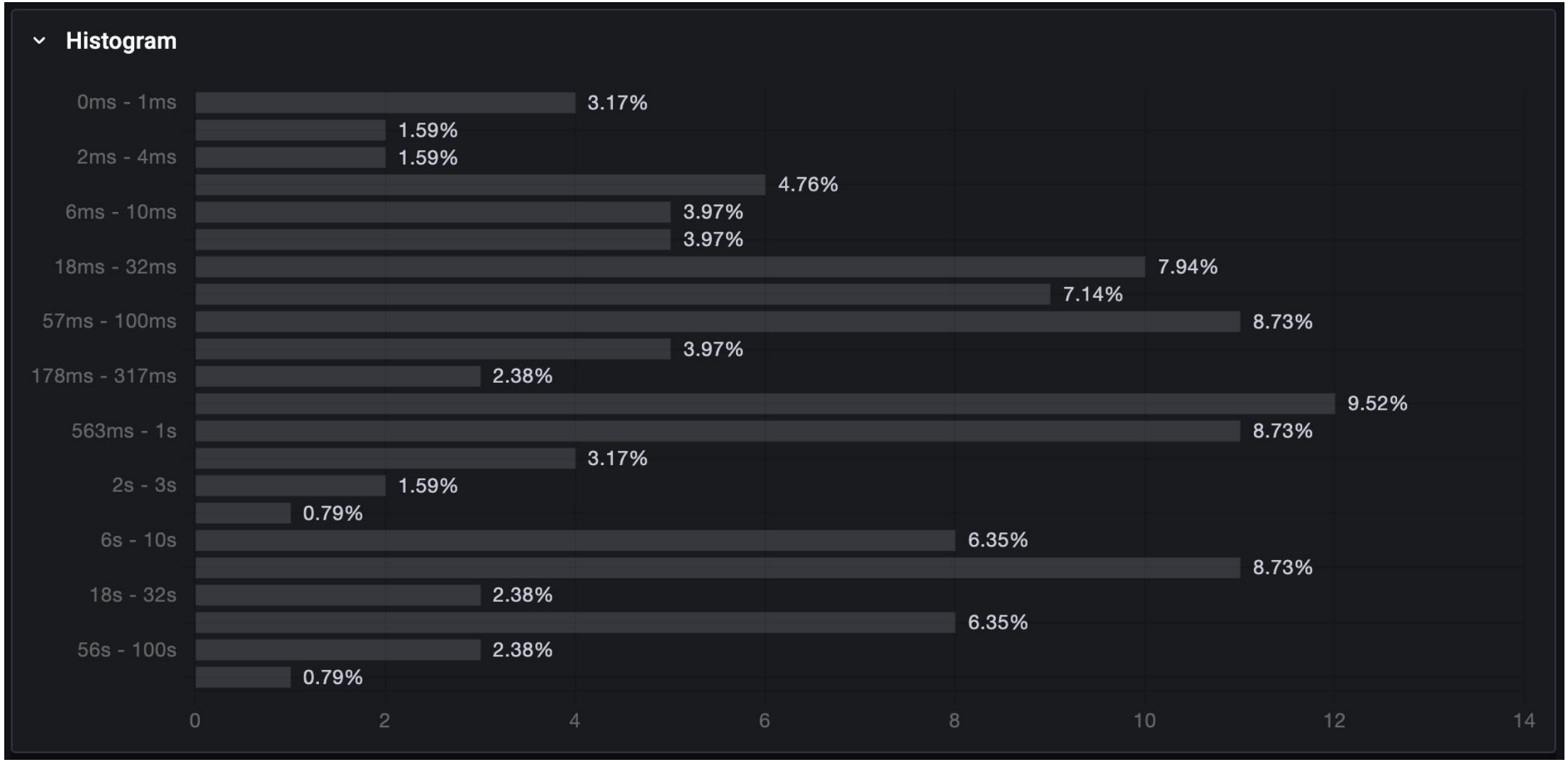
Query Plan

Details | Examples | Tables | **Plan** | Close

```
Subquery Scan on pg_statio_all_tables
  ->  HashAggregate
        Group Key: c.oid, n.nspname, c.relname, t.oid, x.indrelid
        ->  Hash Right Join
              Hash Cond: (x.indrelid = t.oid)
              ->  Seq Scan on pg_index x
              ->  Hash
                    Buckets: 1024  Batches: 1  Memory Usage: 8kB
                    ->  Hash Left Join
                          Hash Cond: (c.reltoastrelid = t.oid)
                          ->  Hash Right Join
                                Hash Cond: (i.indrelid = c.oid)
                                ->  Seq Scan on pg_index i
                                ->  Hash
                                      Buckets: 1024  Batches: 1  Memory Usage: 8kB
                                      ->  Hash Join
                                            Hash Cond: (c.relnamespace = n.oid)
                                            ->  Seq Scan on pg_class c
                                                  Filter: (relkind = ANY ('{r,t,m}'::"char"[]))
```

Histogram

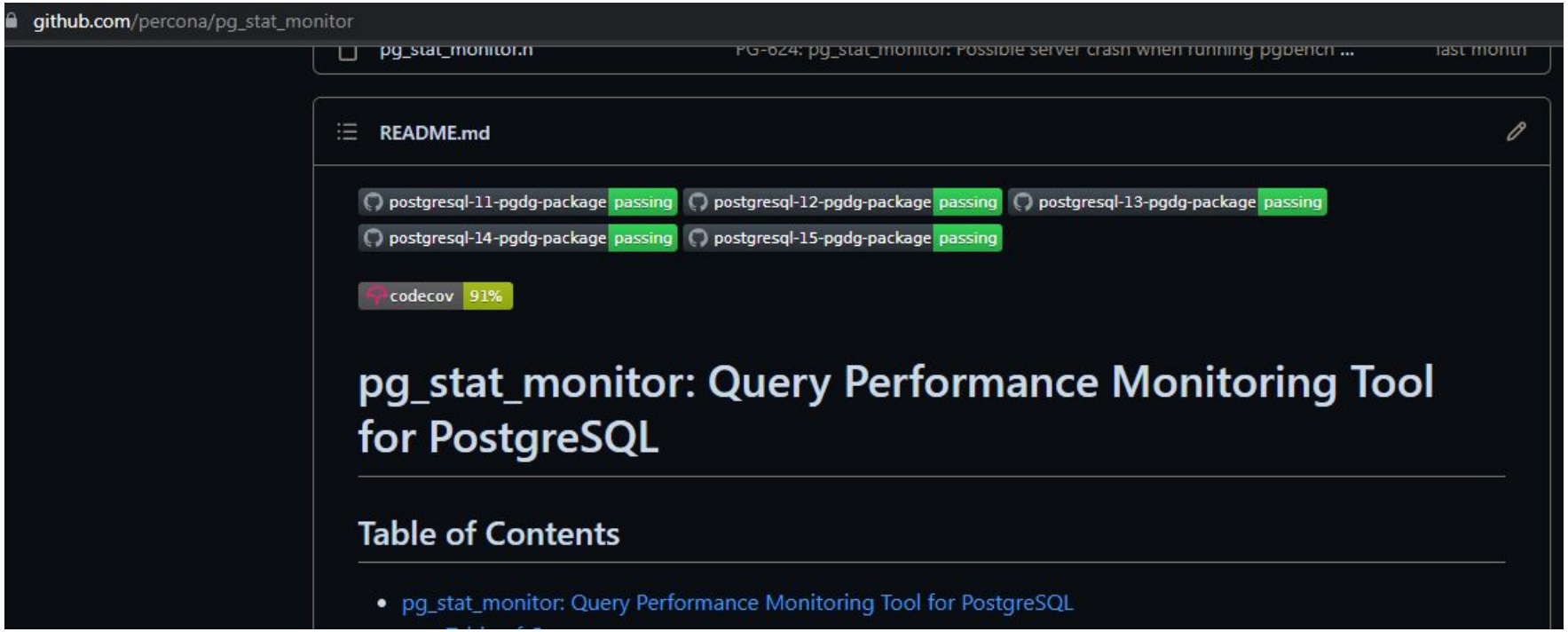
- Visualization of the same queries done in certain amount of time (bucket)
- Grouped to ranges based on query time



How do I start, report a bug or contribute?

Where do I find the project?

GitHub - https://github.com/percona/pg_stat_monitor



Install pg_stat_monitor

- pg_stat_monitor library can be loaded by setting shared_preload_libraries in postgresql.conf file.
  - shared_preload_libraries = 'pg_stat_monitor' # (change requires restart)

- The same parameter shared_preload_libraries can be changed using ALTER SYSTEM command.
  - postgres=# ALTER SYSTEM SET shared_preload_libraries=pg_stat_monitor;
  - ALTER SYSTEM

- Restart the server
  - # sudo systemctl restart postgresql*

- Create the extension using the create extension command.
  - postgres=# CREATE EXTENSION pg_stat_monitor;
  - CREATE EXTENSION

How to report a bug?

- If you found a bug or have a feature request in pg_stat_monitor, please submit the report to https://jira.percona.com/projects/PG/issues

- If there is no existing report, submit your report following these steps:
  - Sign in to Jira issue tracker. You will need to create an account if you do not have one.
  - In the *Summary*, *Description*, *Steps To Reproduce*, *Affects Version* fields describe the problem you have detected.

As a general rule of thumb, try to create bug reports that are:

- Reproducible: describe the steps to reproduce the problem.
- Unique: check if there already exists a JIRA ticket to describe the problem.
- Scoped to a Single Bug: only report one bug in one JIRA ticket.

How to contribute?

- Simply create a Pull Request (PR) in our GitHub repository
- We do follow, the OneFlow development model

Questions?

percona.com

THANK YOU!

percona.com