Configuration settings and diagnostics methods for better PostgreSQL performance

**Peter Petrov,
Senior DBA,
May 11, 2021**

postgrespro.com

Agenda

- Data checksums and approaches of their activation.
- Tuning PostgreSQL parameters for better performance.
- List of useful extensions for better diagnostics and troubleshooting.
- Some examples of Zabbix metrics and fragments of a workload report.
- Kernel parameters tuning.
- Debug symbols installation.

data_checksums activation methods and validation

- initdb –k
- pg_checksums and streaming replication.

  https://paquier.xyz/postgresql-2/postgres-12-pg-checksums/
- https://github.com/credativ/pg_checksums.git for PostgreSQL version <= 11

  https://gitlab.com/gitlab-com/gl-infra/infrastructure/-/issues/10827

Data validation must be done by backup utility because it checks all blocks in the database cluster. If there is a mismatch, then the utility can display a warning or finish its job with an error. If there is no mistake, then backup procedure successfully completed.

pg_probackup as a tool for a database cluster backup and recovery

pg_probackup is a utility to manage backup and recovery of PostgreSQL database clusters. It offers the following benefits:

- Incremental backup
- Validation and verification
- Multiple threads usage to speed up backup and restore
- Backup from standby

shared_buffers, work_mem and temp_buffers tuning

shared_buffers is used to determine how much memory will be allocated for PostgreSQL database for its data caching. A reasonable starting value is ¼ of the memory on the server.

work_mem is the advice for the planner about available amount of memory for internal algorithms like sorting and hashing. A reasonable starting value is 10MB.

temp_buffers is the maximum amount of memory for storing temporary tables data. If an application doesn’t use it, then this parameter value should be 0.

max_connections tuning

max_connections is the maximum number of allowed client connections.

If max_connections > 1000, consider using connection pooling techniques:

- pgbouncer (https://github.com/pgbouncer/pgbouncer)
- odyssey (https://github.com/yandex/odyssey)
- application server connection pooling (Wildfly)

How much memory should be allocated?

shared_buffers + (work_mem + temp_buffers) * max_connections should not exceed the maximum amount of memory on the server to avoid forced PostgreSQL main process termination by OOM killer.

Logging parameters in PostgreSQL (1)

logging_collector = on

log_temp_files. Allows to detect queries with heavy temporary files generation. It can be essential to detect recursive queries which are in infinite cycle.

Logging parameters in PostgreSQL (2)

Tune log_line_prefix for getting more detailed information in a way like this:

- %m – timestamp when a log entry was written
- %p – PostgreSQL backend identifier
- %l - a log entry number inside a PostgreSQL session
- %u – database username.
- %h – IP-address of PostgreSQL client.
- %e – SQLSTATE error code
- %x - transaction identifier

Parameters for PostgreSQL planner

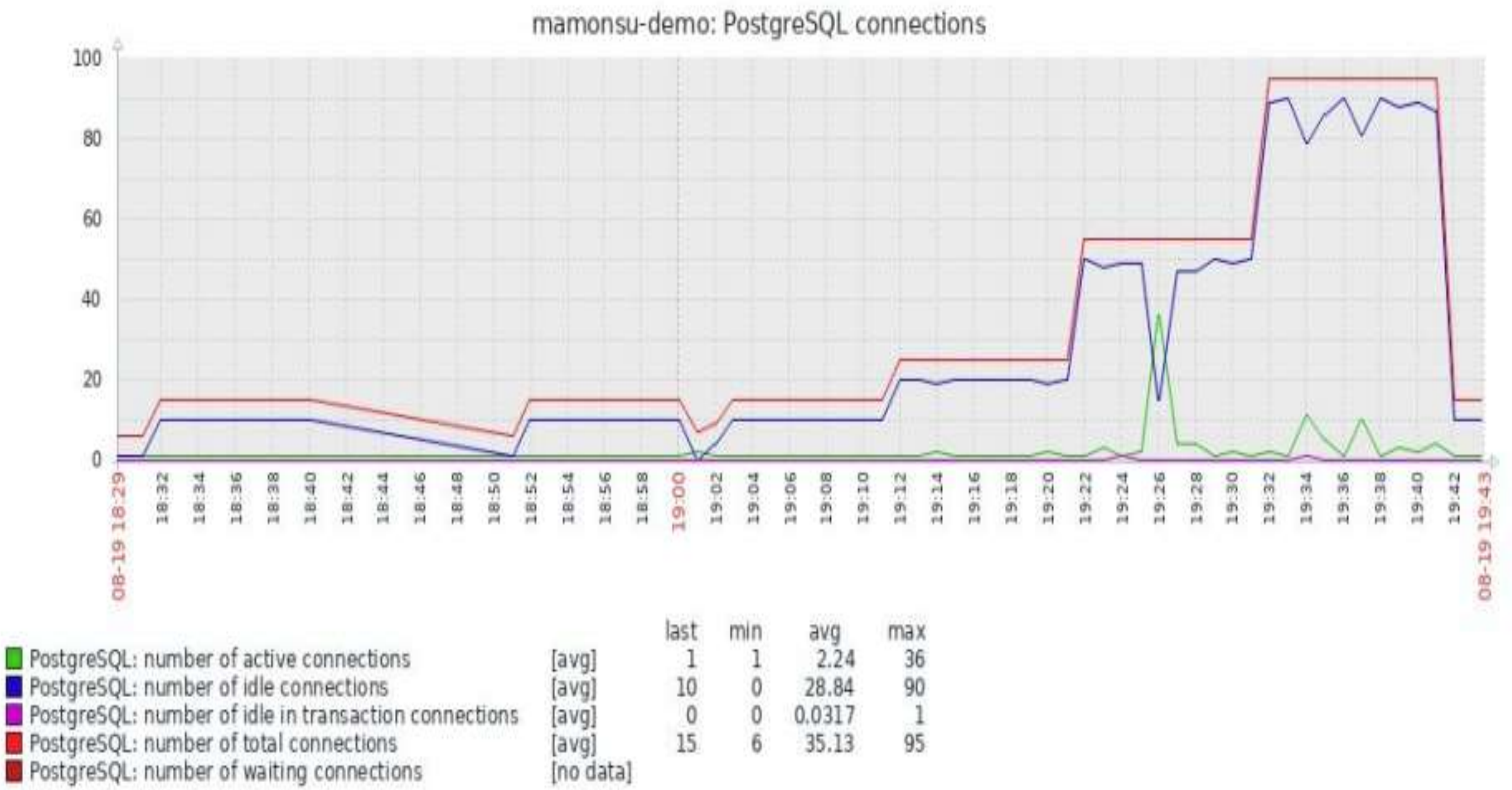
join_collapse_limit = 30. If the value of this parameter is low, then planner can choose non optimal JOINs order.

mamonsu as an active Zabbix agent

Mamonsu is a monitoring agent for collecting PostgreSQL and system metrics and sending them to Zabbix server:

- Works with various operating systems / OSs
- 1 agent = 1 database instance
- Works with PostgreSQL version >= 9.5
- Provides various metrics related to PostgreSQL activity

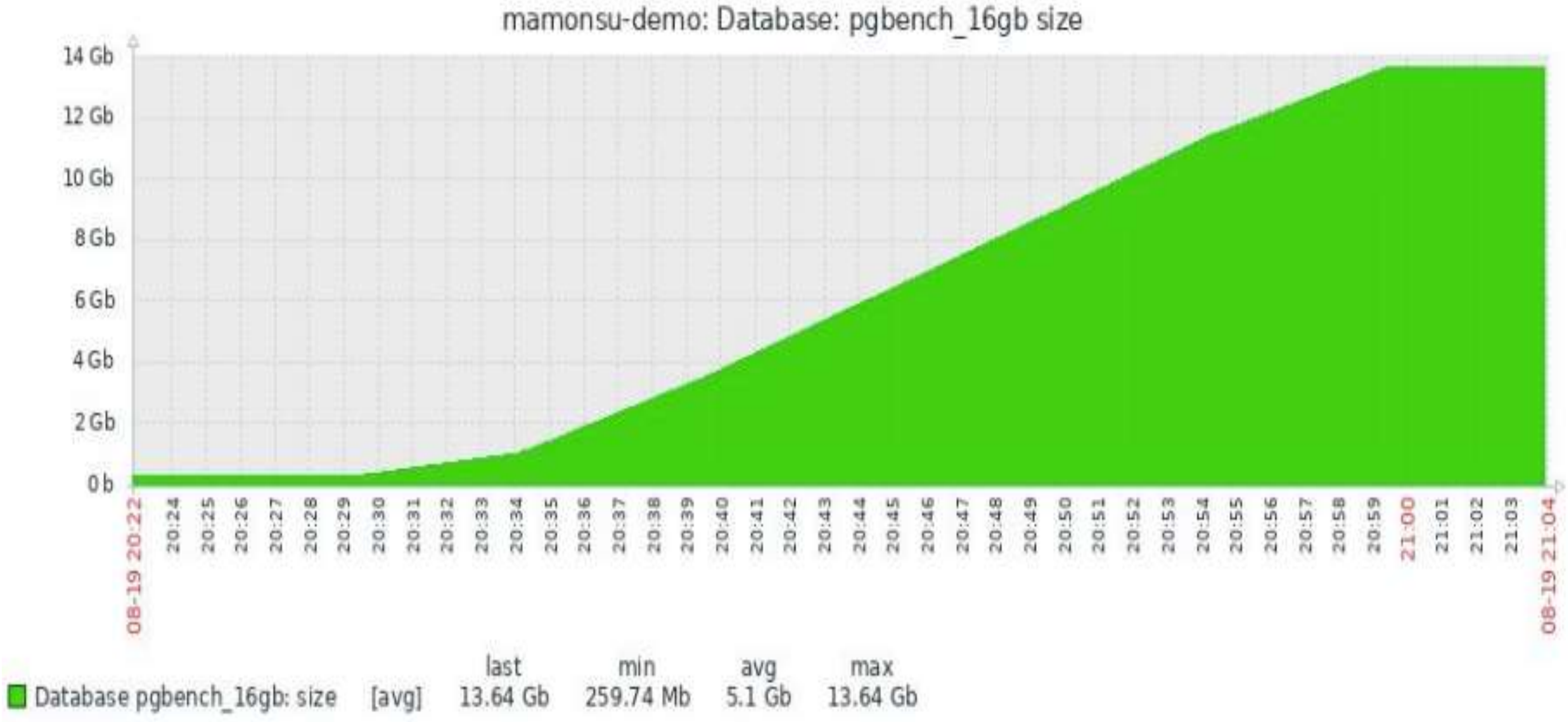
PostgreSQL statistics connection



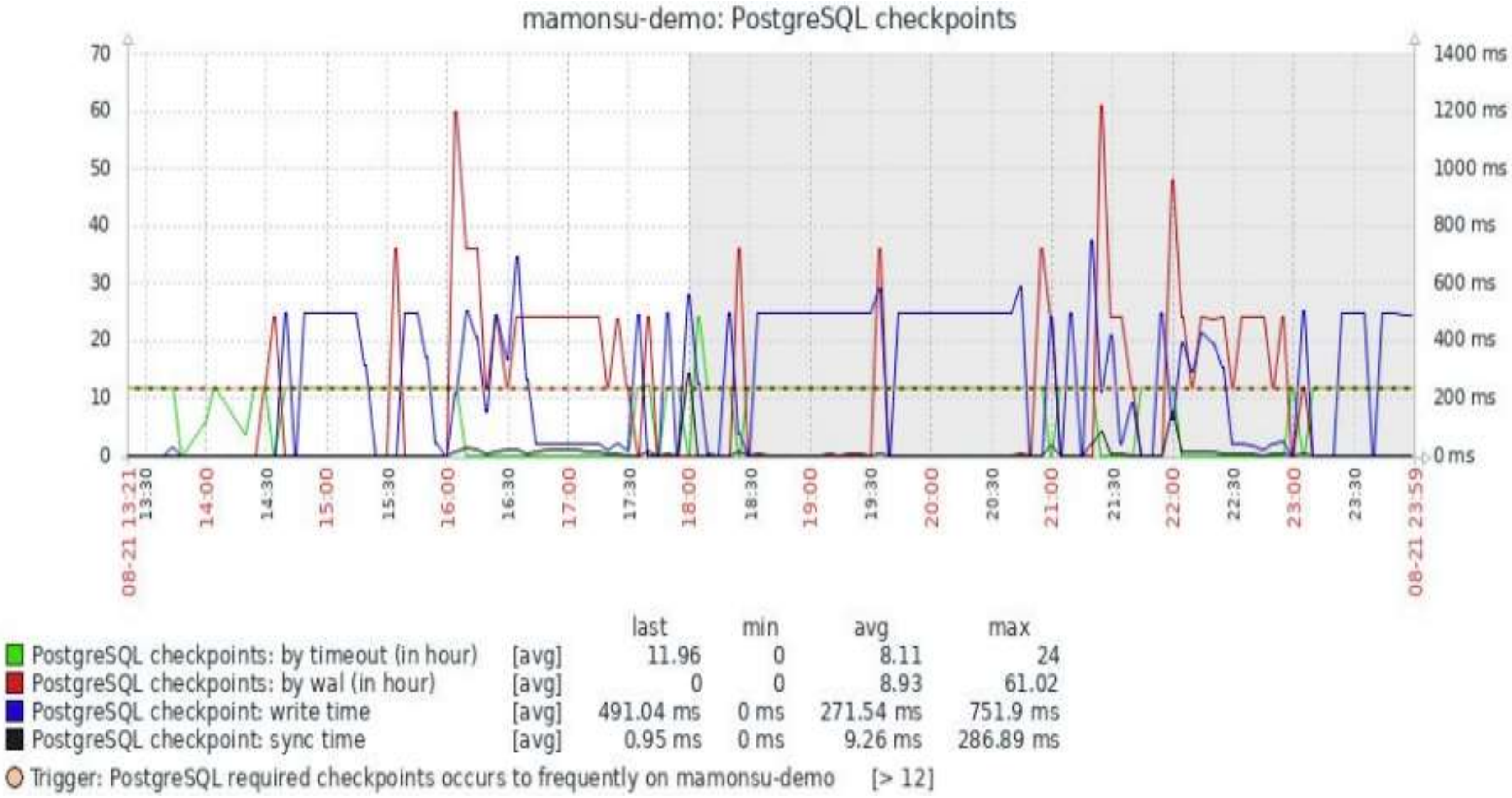
mamonsu-demo: PostgreSQL connections

		last	min	avg	max
🟩 PostgreSQL: number of active connections	[avg]	1	1	2.24	36
🟦 PostgreSQL: number of idle connections	[avg]	10	0	28.84	90
🟪 PostgreSQL: number of idle in transaction connections	[avg]	0	0	0.0317	1
🟥 PostgreSQL: number of total connections	[avg]	15	6	35.13	95
🟥 PostgreSQL: number of waiting connections	[no data]				

Database cluster size statistics



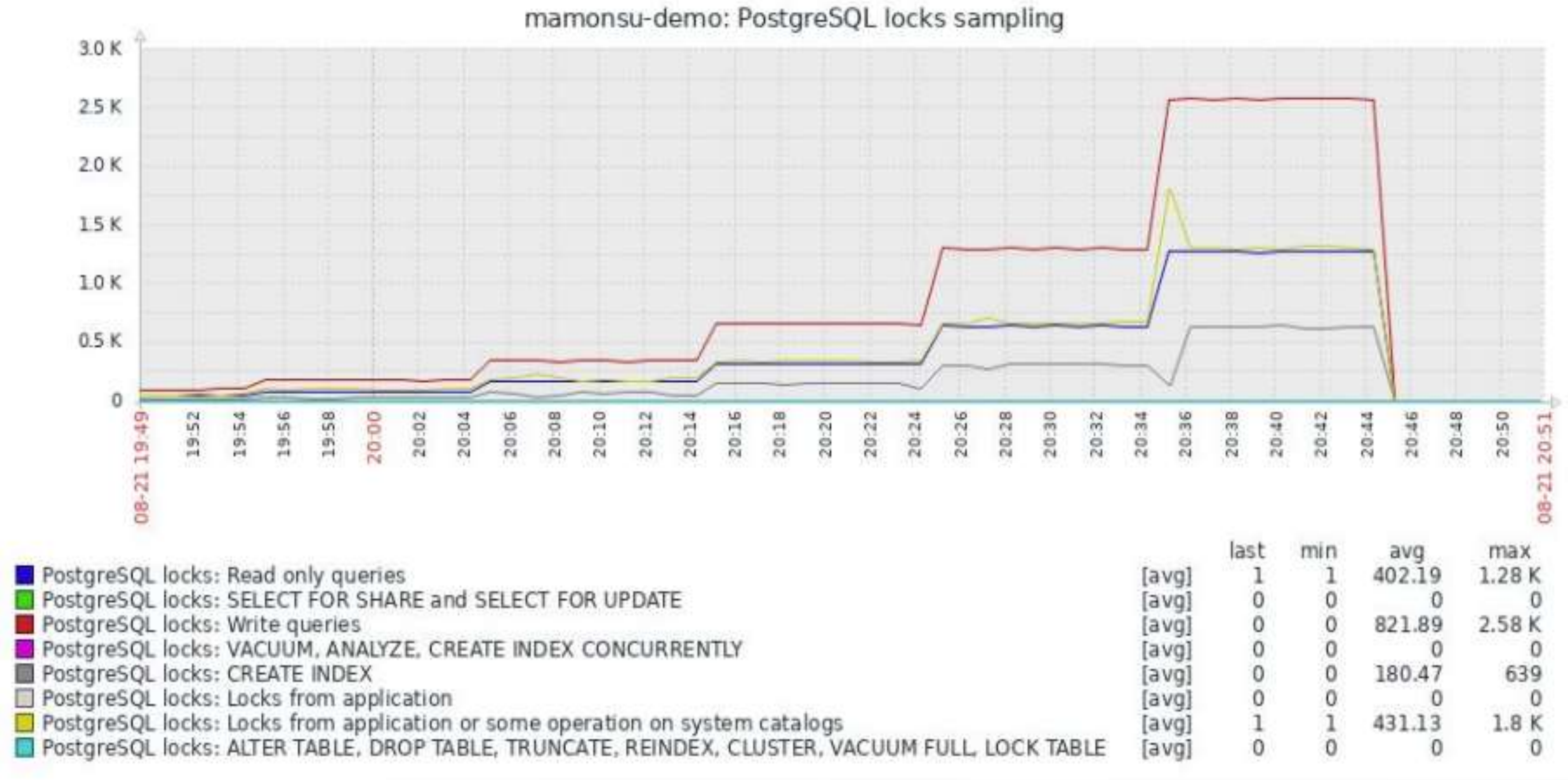
PostgreSQL checkpoint statistics



		last	min	avg	max
🟩 PostgreSQL checkpoints: by timeout (in hour)	[avg]	11.96	0	8.11	24
🟥 PostgreSQL checkpoints: by wal (in hour)	[avg]	0	0	8.93	61.02
🟦 PostgreSQL checkpoint: write time	[avg]	491.04 ms	0 ms	271.54 ms	751.9 ms
■ PostgreSQL checkpoint: sync time	[avg]	0.95 ms	0 ms	9.26 ms	286.89 ms

○ Trigger: PostgreSQL required checkpoints occurs to frequently on mamonsu-demo [> 12]

PostgreSQL locks sampling



		last	min	avg	max
🟦 PostgreSQL locks: Read only queries	[avg]	1	1	402.19	1.28 K
🟩 PostgreSQL locks: SELECT FOR SHARE and SELECT FOR UPDATE	[avg]	0	0	0	0
🟥 PostgreSQL locks: Write queries	[avg]	0	0	821.89	2.58 K
🟪 PostgreSQL locks: VACUUM, ANALYZE, CREATE INDEX CONCURRENTLY	[avg]	0	0	0	0
⬛ PostgreSQL locks: CREATE INDEX	[avg]	0	0	180.47	639
⬜ PostgreSQL locks: Locks from application	[avg]	0	0	0	0
🟨 PostgreSQL locks: Locks from application or some operation on system catalogs	[avg]	1	1	431.13	1.8 K
🟦 PostgreSQL locks: ALTER TABLE, DROP TABLE, TRUNCATE, REINDEX, CLUSTER, VACUUM FULL, LOCK TABLE	[avg]	0	0	0	0

List of useful extensions (1)

pg_stat_statements for analyzing which queries have the longest execution time.

pg_stat_kcache for finding queries consuming the most CPU system and user time.

auto_explain for finding query plans and parameters for further tuning.

pg_wait_sampling for collecting history of wait events and waits profiles.

List of useful extensions (2)

pg_profile for creating historic workload repository containing various metrics such as:

- SQL Query statistics
- DML statistics
- Schema object statistics
- Vacuum-related statistics

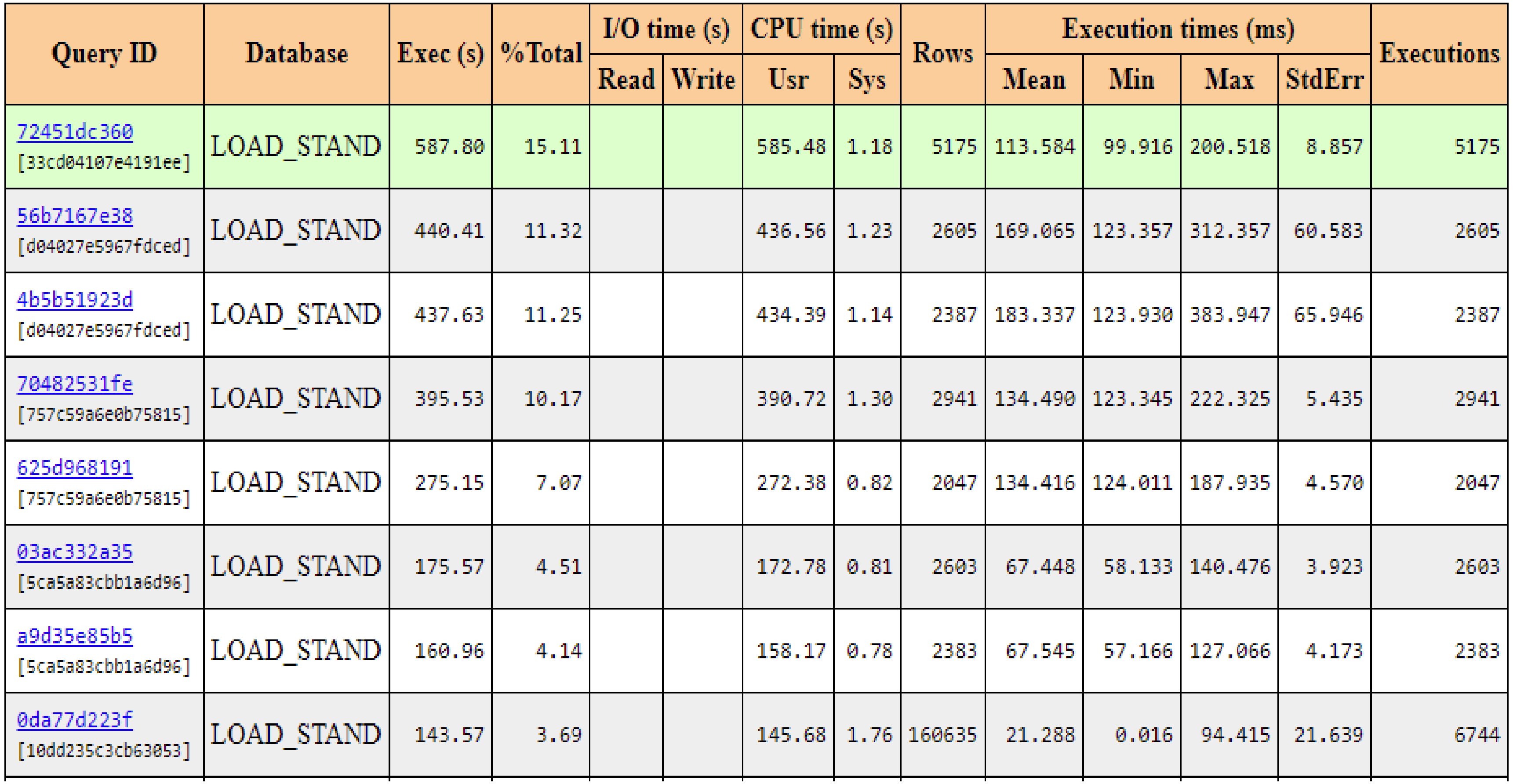
List of useful extensions (3)

plprofiler for creating performance profiles of PL/pgSQL functions and stored procedures.

pgpro_stats as a combination of pg_stat_statements, pg_stat_kcache and pg_wait_sampling (only for Postgres Pro customers)

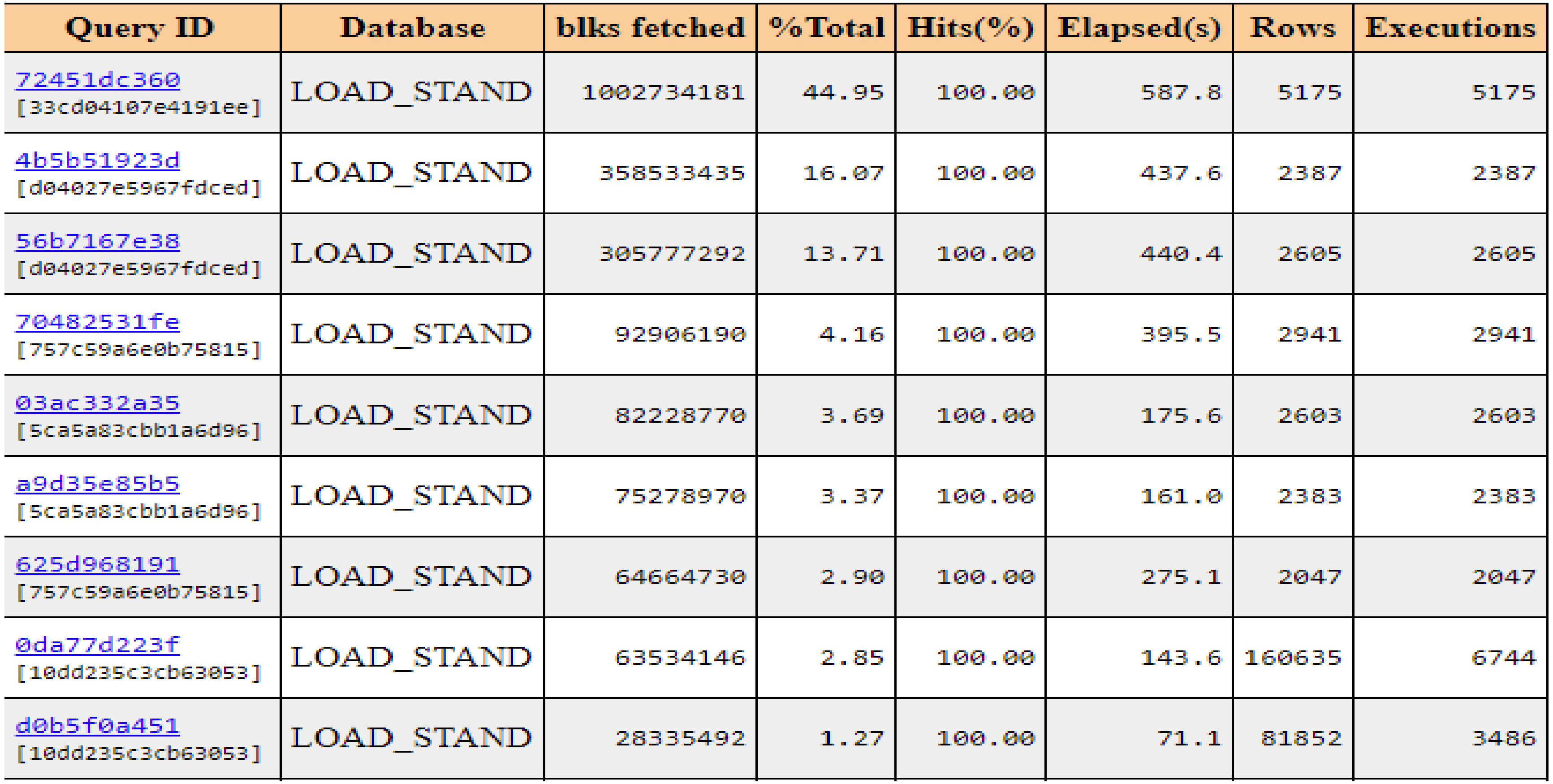
pgpro_pwr for gathering information from pgpro_stats (only for Postgres Pro customers)

Top SQL by execution time collected by pg_profile module



Query ID	Database	Exec (s)	%Total	I/O time (s)		CPU time (s)		Rows	Execution times (ms)				Executions
				Read	Write	Usr	Sys		Mean	Min	Max	StdErr	
72451dc360 [33cd04107e4191ee]	LOAD_STAND	587.80	15.11			585.48	1.18	5175	113.584	99.916	200.518	8.857	5175
56b7167e38 [d04027e5967fdced]	LOAD_STAND	440.41	11.32			436.56	1.23	2605	169.065	123.357	312.357	60.583	2605
4b5b51923d [d04027e5967fdced]	LOAD_STAND	437.63	11.25			434.39	1.14	2387	183.337	123.930	383.947	65.946	2387
70482531fe [757c59a6e0b75815]	LOAD_STAND	395.53	10.17			390.72	1.30	2941	134.490	123.345	222.325	5.435	2941
625d968191 [757c59a6e0b75815]	LOAD_STAND	275.15	7.07			272.38	0.82	2047	134.416	124.011	187.935	4.570	2047
03ac332a35 [5ca5a83cbb1a6d96]	LOAD_STAND	175.57	4.51			172.78	0.81	2603	67.448	58.133	140.476	3.923	2603
a9d35e85b5 [5ca5a83cbb1a6d96]	LOAD_STAND	160.96	4.14			158.17	0.78	2383	67.545	57.166	127.066	4.173	2383
0da77d223f [10dd235c3cb63053]	LOAD_STAND	143.57	3.69			145.68	1.76	160635	21.288	0.016	94.415	21.639	6744

Top SQL by shared blocks fetched by pg_profile



Query ID	Database	blks fetched	%Total	Hits(%)	Elapsed(s)	Rows	Executions
72451dc360 [33cd04107e4191ee]	LOAD_STAND	1002734181	44.95	100.00	587.8	5175	5175
4b5b51923d [d04027e5967fdced]	LOAD_STAND	358533435	16.07	100.00	437.6	2387	2387
56b7167e38 [d04027e5967fdced]	LOAD_STAND	305777292	13.71	100.00	440.4	2605	2605
70482531fe [757c59a6e0b75815]	LOAD_STAND	92906190	4.16	100.00	395.5	2941	2941
03ac332a35 [5ca5a83cbb1a6d96]	LOAD_STAND	82228770	3.69	100.00	175.6	2603	2603
a9d35e85b5 [5ca5a83cbb1a6d96]	LOAD_STAND	75278970	3.37	100.00	161.0	2383	2383
625d968191 [757c59a6e0b75815]	LOAD_STAND	64664730	2.90	100.00	275.1	2047	2047
0da77d223f [10dd235c3cb63053]	LOAD_STAND	63534146	2.85	100.00	143.6	160635	6744
d0b5f0a451 [10dd235c3cb63053]	LOAD_STAND	28335492	1.27	100.00	71.1	81852	3486

huge_pages activation (1)

If PostgreSQL **shared_buffers** >= 20GB, it is highly recommended to use **huge pages** to reduce overhead while working with large and continuous regions of memory. For activating it you should take the following steps.

huge_pages activation (2)

1. Determine postmaster pid by watching contents of $PGDATA/postmaster.pid.
2. Determine VmPeak by watching contents of **/proc/postmaster_pid/status**.
3. Determine HugePageSize from **/proc/meminfo**
4. Divide VmPeak by HugePageSize and save the calculated value in **/etc/sysctl.conf** file as **vm.nr_hugepages = value**

transparent_huge_pages deactivation

Disable **transparent huge pages** by executing following commands as root user:

- echo never > /sys/kernel/mm/transparent_hugepage/enabled
- echo never > /sys/kernel/mm/transparent_hugepage/defrag

However, some changes must be made to grub config to preserve settings even after the server's reboot.

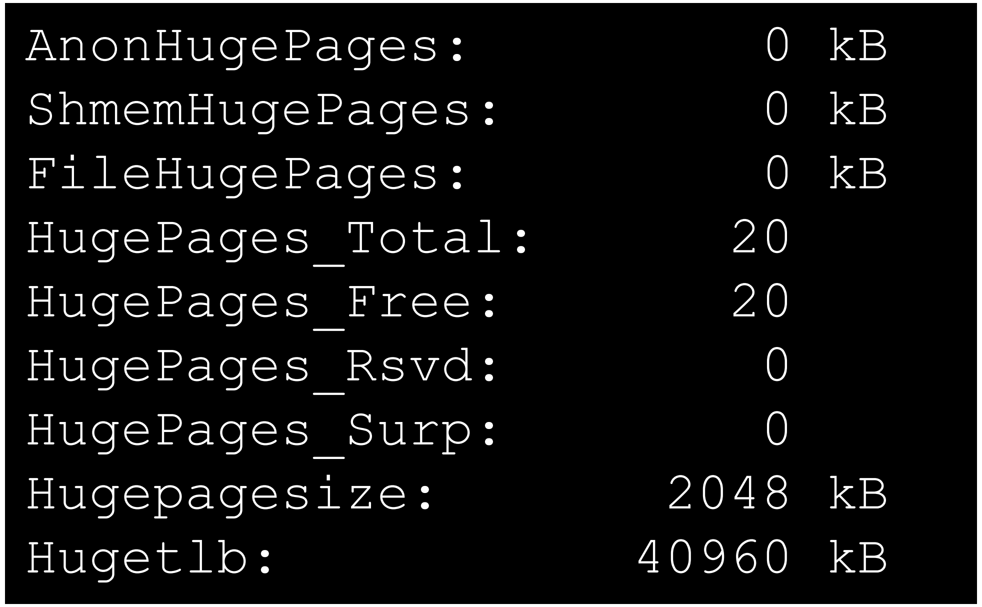
Making changes to grub configuration file

1. Install grub2-common package.
2. Add **hugepage=value** at the end of GRUB_CMDLINE_LINUX_DEFAULT in /etc/default/grub file.
3. Add transparent_hugepage=never at the end of GRUB_CMDLINE_LINUX_DEFAULT in /etc/default/grub file.
4. Run **update-grub** to apply the config to grub and reboot the system.

Checking values of the performance-related parameters

After rebooting run command **`grep Huge /proc/meminfo`**.

If HugePages_Total > 0 and AnonHugePages = 0kB then settings have applied correctly.

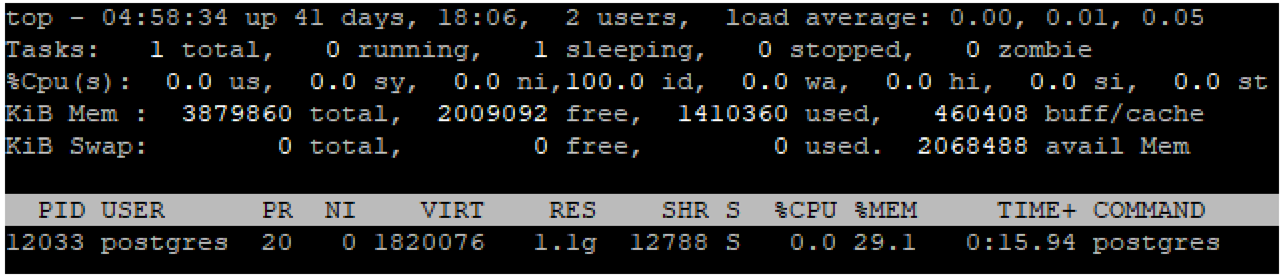


```
AnonHugePages:          0 kB
ShmemHugePages:         0 kB
FileHugePages:          0 kB
HugePages_Total:       20
HugePages_Free:        20
HugePages_Rsvd:         0
HugePages_Surp:         0
Hugepagesize:        2048 kB
Hugetlb:            40960 kB
```

Memory leak investigation

One of our customers noticed that some PostgreSQL process was consuming large amount of memory, 1.1GB and asked us to help them in resolving the problem.

We need to know function call hierarchy to understand the problem's origin. Let's see what it looks like by default without installing any additional packages.



```
top - 04:58:34 up 41 days, 18:06,  2 users,  load average: 0.00, 0.01, 0.05
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3879860 total,  2009092 free,  1410360 used,   460408 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  2068488 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
12033 postgres  20   0 1820076   1.1g  12788 S   0.0 29.1   0:15.94 postgres
```

Stack trace without installing any additional packages

We can see incomplete function call hierarchy which doesn’t help to detect the problem’s origin. To solve this issue, additional packages with debug symbols must be installed.

```
#0  0x0000000000835440 in GetCachedPlan ()
#1  0x000000000063578d in SPI_plan_get_cached_plan
()
#2  0x00007f1c6b7528d2 in ?? () from /opt/pgsql/ver-
10/lib/plpgsql.so
#3  0x00007f1c6b753b4a in ?? () from /opt/pgsql/ver-
10/lib/plpgsql.so
```

Debug symbols installation for PostgreSQL (1)

Debug symbols allow us to get the names of variables, functions and functions calling hierarchy.

The debug symbols package’s version must match the server version with minor precision. For example, for PostgreSQL 13.2 the following packages should be installed:

- postgresql-client-13-dbsym
- postgresql-13-dbgsym
- postgresql-plperl-13-dbgsym (in case of using plperl)
- postgresql-plpython3-13-dbgsym (in case of using plpython3)

Debug symbols installation for PostgreSQL (2)

Some extensions and their debug symbols should be installed separately.

Let’s consider pg_stat_kcache extension:

- postgresql-13-pg-stat-kcache
- postgresql-13-pg-stat-kcache-dbgsym

Debug symbols installation for OS (1)

Also debug packages for OS should be installed which can be done the following way:

```
echo "deb http://ddebs.ubuntu.com $(lsb_release -cs) main
restricted universe multiverse

      deb http://ddebs.ubuntu.com $(lsb_release -cs)-updates main
restricted universe multiverse

      deb http://ddebs.ubuntu.com $(lsb_release -cs)-proposed
main restricted universe multiverse" | \

      sudo tee -a /etc/apt/sources.list.d/ddebs.list

wget --quiet -O - http://ddebs.ubuntu.com/dbgsym-release-key.asc
| sudo apt-key add -

sudo apt-get update && sudo apt-get install gdb
```

Debug symbols installation for OS (2)

Connect to an idle PostgreSQL backend by using

```
sudo gdb -p pid
```

Then gdb will display a list of debug symbols packages that need to be installed.

In the case of clean installation of Ubuntu 20.04 the command will looks something like that.

Debug symbols installation for OS (3)

```
apt-get install libxml2-dbgsym libssl1.1-dbgsym
libcrypto++6-dbg libicu66-dbgsym libc6-dbg
libaudit1-dbgsym libkrb5-dbg libldap-2.4-2-dbgsym
libsasl2-modules-dbgsym libstdc++6-10-dbg liblz4-1-
dbgsym libcrypt1-dbgsym libcap-ng0-dbgsym
libkeyutils1-dbgsym libheimntlm0-heimdal-dbgsym
libasn1-8-heimdal-dbgsym libhcrypto4-heimdal-dbgsym
libidn2-0-dbgsym libunistring2-dbgsym libtasn1-6-
dbgsym libnettle7-dbgsym libhogweed5-dbgsym
libgmp10-dbgsym libgpg-error0-dbgsym libwind0-
heimdal-dbgsym libheimbase1-heimdal-dbgsym libhx509-
5-heimdal-dbgsym libffi7-dbgsym liblzma5-dbgsym
```

Stack
trace after installing packages with debug symbols (1)

After installing packages with debug symbols, we get a more accurate function call tree.

```
#0  GetCachedPlan (plansource=0x2c4d668,
boundParams=boundParams@entry=0x0, useResOwner=1
'\001', queryEnv=0x0) at plancache.c:1308
#1  0x000000000063578d in SPI_plan_get_cached_plan
(plan=<optimized out>) at spi.c:1669
#2  0x00007f1c6b7528d2 in exec_simple_check_plan
(estate=0x7ffd7f136a00, expr=0x2d42ad0) at
pl_exec.c:6954
#3  exec_prepare_plan (estate=0x7ffd7f136a00,
expr=0x2d42ad0, cursorOptions=<optimized out>) at
pl_exec.c:3743
```

Setting options for gathering core dump (1)

segmentation fault is a failure condition associated with memory access violation. The process stops working and generates a core dump file.

core dump file is a state of a working memory of a computer program at a specific time of crashing.

core_pattern is a template for core dump file’s name.

sudo sysctl kernel.core_pattern

- kernel.core_pattern = |/usr/share/apport/apport %p %s %c %d %P %E
- kernel.core_pattern = |/lib/systemd/systemd-coredump %P %u %g %s %t 9223372036854775808 %h

Setting options for gathering core dump (2)

You can change kernel.core_pattern setting as follows:

```
sudo sysctl ‘kernel.core_pattern=/tmp/core-%e-%s-%u-%g-%p-%t’
```

%e – executable filename

%s – signal number, which caused core dump generation

%u – user identifier of process owner

%g – group identifier of process owner

%p – terminated process identifier

%t – UNIX-time of a dump

Limit settings for PostgreSQL and its client applications (1)

For client applications like pg_dump, psql and pg_restore limits for maximum file and core dump size should be written in **/etc/security/limits.conf** as shown below:

```
postgres hard core unlimited

postgres soft core unlimited

postgres hard fsize unlimited

postgres soft fsize unlimited
```

Limit settings for PostgreSQL and its client applications (2)

In case of running PostgreSQL as a service by systemd limits can be defined, for example, in a system unit file. For more information, please, consult the following manual page

man 5 systemd.exec

Useful links (1)

- Debug Symbol Packages. https://wiki.ubuntu.com/Debug%20Symbol%20Packages
- Linux kernel documentation. https://www.kernel.org/doc/html/latest/admin-guide/sysctl/kernel.html
- Apport. https://wiki.ubuntu.com/Apport
- systemd-coredump. https://man7.org/linux/man-pages/man8/systemd-coredump.8.html
- Logging in PostgreSQL. https://www.postgresql.org/docs/current/runtime-config-logging.html
- Planner options in PostgreSQL. https://www.postgresql.org/docs/13/runtime-config-query.html

Useful links (2)

- pg_checksums for PostgreSQL 12 and higher. https://www.postgresql.org/docs/13/app-pgchecksums.html
- pg_checksums for PostgreSQL version lower than 12. https://github.com/credativ/pg_checksums
- pg_stat_statements module. https://www.postgresql.org/docs/13/pgstatstatements.html
- pg_stat_kcache module. https://github.com/powa-team/pg_stat_kcache
- pg_wait_sampling module. https://github.com/postgrespro/pg_wait_sampling

Useful links (3)

- auto_explain module. https://www.postgresql.org/docs/13/auto-explain.html
- pgpro_stats module. https://postgrespro.com/docs/enterprise/12/pgpro-stats
- pg_profile module. https://github.com/zubkov-andrei/pg_profile
- pgpro_pwr module. https://postgrespro.com/docs/enterprise/12/pgpro-pwr
- mamonsu. https://github.com/postgrespro/mamonsu
- pg_probackup. https://github.com/postgrespro/pg_probackup

Postgres Professional

http://postgrespro.com/

info@postgrespro.com

postgrespro.com