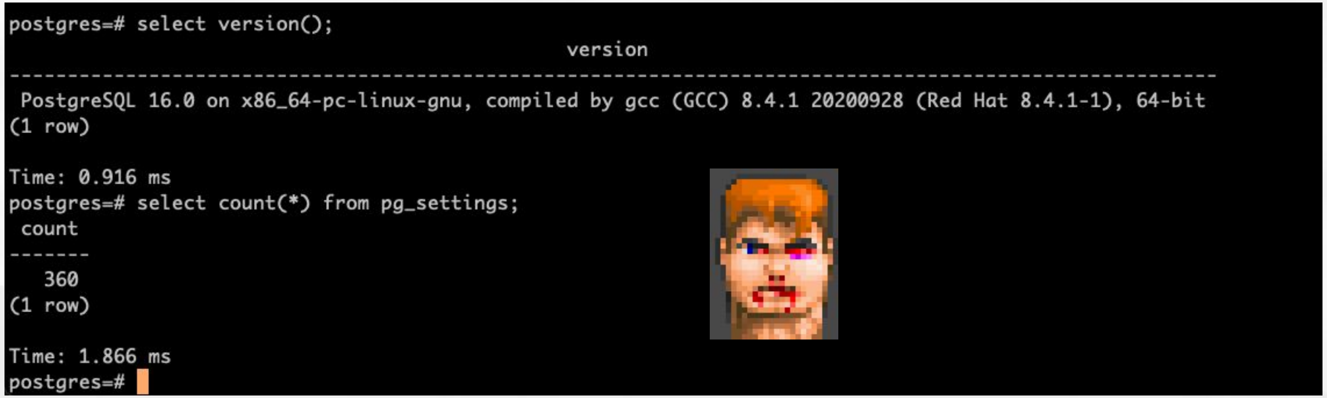
PostgreSQL Parametrization Hints for Intensive Workloads

Agustín Gallego - Support Engineer

Table of Contents



```
postgres=# select version();
                                                     version
-----------------------------------------------------------------------------------------------------------
 PostgreSQL 16.0 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 8.4.1 20200928 (Red Hat 8.4.1-1), 64-bit
(1 row)

Time: 0.916 ms
postgres=# select count(*) from pg_settings;
 count
-------
   360
(1 row)

Time: 1.866 ms
postgres=#
```

Preamble

- Is there a list of foolproof variables we should tune?
- How can we assess which candidates to pick first?
- How can we measure and quantify the improvements or regressions in performance?

- Is there a list of foolproof variables we should tune?
    - It will depend on your (application) workload and needs in particular

- How can we assess which candidates to pick first?
    - There are some helper tools that can guide us when we have doubts

- How can we measure and quantify the improvements or regressions in performance?
    - We first need to set up data gathering tools, and then be familiar with them to interpret their results

PERCONA

Incidence Statistics

Incidence Statistics

	Parameter	Incidence [%]
1	max_connections	10.62%
2	max_wal_size	9.68%
3	work_mem	8.63%
4	shared_buffers	7.85%
5	checkpoint_timeout	7.27%
6	min_wal_size	6.44%
7	wal_level	4.76%
8	wal_keep_segments/wal_keep_size	4.71%
9	huge_pages	4.55%
10	autovacuum_vacuum_cost_limit	4.19%
11	maintenance_work_mem	3.98%
12	random_page_cost	3.82%
13	autovacuum_max_workers	3.30%
14	effective_cache_size	2.88%
15	checkpoint_completion_target	2.67%

Incidence Statistics

16	statement_timeout	2.35%
17	autovacuum_vacuum_scale_factor	2.25%
18	autovacuum_vacuum_threshold	2.15%
19	idle_in_transaction_session_timeout	2.04%
20	default_statistics_target	2.04%
21	autovacuum_vacuum_cost_delay	1.78%
22	bgwriter_lru_maxpages	1.36%
23	autovacuum_naptime	1.31%
24	lock_timeout	0.84%
25	autovacuum_work_mem	0.84%
26	bgwriter_delay	0.63%

Top Candidate Configuration Variables

CONNECTIONS AND AUTHENTICATION

Connections and Authentication

- **max_connections:** Determines the maximum number of concurrent connections to the database server.

RESOURCE USAGE (except WAL)

Resource Usage (except WAL)

- **work_mem:** Sets the maximum amount of memory used by a query operation (ORDER BY, DISTINCT, merge joins, hash joins, hash-based aggregation, and so on) before writing to temporary disk files. Multiple query operations and concurrent sessions can use memory up to this limit, resulting in a higher total memory usage which might lead to OOM scenarios.
- **shared_buffers:** Determines the amount of memory used by the database server for shared memory buffers. This means the most frequently used pages.

Resource Usage (except WAL)

- **maintenance_work_mem:** Specifies the maximum memory used by maintenance operations like VACUUM, CREATE INDEX, and ALTER TABLE ADD FOREIGN KEY. Increasing this value can improve performance for vacuuming. When autovacuum_work_mem is not set, autovacuum_max_workers will use maintenance_work_mem.
- **huge_pages:** Setting controls whether huge pages are requested for the main shared memory area. It reduces memory management overhead and improves performance by reducing page table size and CPU time.
- **bgwriter_lru_maxpages:** Determines the maximum number of buffers written by the background writer in each round.

Resource Usage (except WAL)

- **bgwriter_delay:** Specifies the time interval between activity rounds for the background writer.
- **autovacuum_work_mem:** Determines the maximum memory used by each autovacuum worker process. Autovacuum can utilize a maximum of 1GB of memory. Values higher than 1GB will not improve the number of dead tuples autovacuum can collect during table scanning.

WRITE-AHEAD LOG

Write-ahead Log

- **max_wal_size:** Sets the maximum (soft limit) size allowed for the Write-Ahead Log (WAL) during automatic checkpoints in PostgreSQL.
- **checkpoint_timeout:** Sets the maximum interval between automatic Write-Ahead Log (WAL) checkpoints. Be careful when crash recovery time is a priority.
- **min_wal_size:** Determines the threshold for recycling old Write-Ahead Log (WAL) files during checkpoints instead of removing them.
- **checkpoint_completion_target:** Determines the target duration of checkpoint completion as a fraction of the total time between checkpoints.

REPLICATION

Replication

- **wal_keep_segments(v12-):** Specifies the minimum number of past log file segments that are retained in the pg_wal directory. Segments are hold in case a Standby requires them for streaming replication purposes.
- **wal_keep_size(v13+):** Specifies the minimum size of past log file segments that are retained in the pg_wal directory in case a Standby requires them for streaming replication purposes.

QUERY TUNING

Query Tuning

- **random_page_cost:** Determines the planner's estimation of the cost of accessing a disk page randomly. Increasing the value makes the optimizer prone to produce plans based on sequential scans while reducing it makes it prone to produce plans based on index scans.
- **default_statistics_target:** Determines the default statistics target for table columns. Increasing the value improves the planner's estimatimations but also increases the time and HW resources required for ANALYZE operations.

Query Tuning

- **effective_cache_size:** Represents the planner's estimation of the available disk cache size for a single query. It influences the cost estimation for using an index. Higher values will favor index scans. This parameter does not affect the size of shared memory allocated and does not reserve kernel disk cache; it is used for estimation purposes.

AUTOVACUUM

Autovacuum

- **autovacuum_vacuum_cost_limit:** Specifies the cost limit used for automatic VACUUM operations. Cost refers to an internal effort measure. The cost is distributed across all concurrent autovacuum workers.
- **autovacuum_max_workers:** Determines the maximum number of autovacuum processes that can run concurrently.
- **autovacuum_vacuum_cost_delay:** Determines the delay between automatic VACUUM operations. A throttle mechanism between vacuum operations. Higher values reduce the DB load and slow down the autovacuum executions. Increasing it speeds up the relation clean up, increasing the load.

Autovacuum

- **autovacuum_naptime:** Determines the minimum delay between autovacuum.
- **autovacuum_vacuum_scale_factor:**
- **autovacuum_vacuum_threshold:**
-

CLIENT CONNECTION DEFAULTS

Client Connections

- **statement_timeout:** Abort any statement that takes more than the specified amount of time.
- **lock_timeout:** Abort any statement that waits longer than the specified amount of time while attempting to acquire a lock on a table, index, row, or other database object.
- **idle_in_transaction_session_timeout:** Terminate any session that has been idle (that is, waiting for a client query) within an open transaction for longer than the specified amount of time.

REPORTING AND LOGGING

Reporting and Logging

- **log_checkpoints:** Enables or disables logging of checkpoints and restartpoints in the server log. The messages include statistics such as the number of buffers written and the time taken for the write operation.
- **log_connections/log_disconnections:** When enabled, it logs every attempt of connection and session termination. Connection attempts will be logged when successful or fail.
- **log_temp_files:** Determines the logging behavior for temporary file names and sizes.

Reporting and Logging

- **log_autovacuum_min_duration:** Sets the autovacuum execution time threshold from which such actions will be logged.
- **log_min_duration_statement:** Enables or disables logging the duration of *completed* SQL statements that meet or exceed a specified execution time threshold.

Monitoring Tools

Monitoring and Data Gathering Tools

- We need to be able to measure performance, to tell if changes are positively or negatively impacting it
- Change one configuration/setting at a time
- Collect data at low resolutions to have more introspection

Some tools we use in Support are:

- Percona Monitoring and Management
  - https://docs.percona.com/percona-monitoring-and-management/index.html
- pg_gather
  - https://github.com/percona/support-snippets/tree/master/postgresql/pg_gather
- pgBadger
  - https://github.com/darold/pgbadger

Tuning Tools

Tuning Tools

- Be careful, this is a double-edged sword
- Projects can be abandoned or not maintained for a long time

Some examples:

- pg_auto_tune
  - https://github.com/codeforall/pg_auto_tune
- postgresqltuner.pl
  - Outdated
  - https://github.com/jfcoz/postgresqltuner
- PGTune
  - Very basic
  - https://pgtune.leopard.in.ua/

¡Gracias!

¿Más preguntas?
Nuestro email:

Pablo Svampa
pablo.svampa@percona.com

Agustin Gallego
agustin.gallego@percona.com