Your remote PostgreSQL DBA Team

data egret

My "default" postgresql.conf file, Step by Step

Ilya Kosmodemiansky

ik@dataegret.com

Before we start...

- 269 settings in version 10
- 365 setting in version 14
- Settings in **postgresql.conf** are to be changed manually
- **postgresql.auto.conf** can be updated by **ALTER SYSTEM**
- **pg_settings** combines everything together

pg_settings

```
postgres=# \x
Expanded display is on.
postgres=# select * from pg_settings where name ~ 'checkpoint_timeout';
-[ RECORD 1 ]---+-------------------------------------------------------------
name            | checkpoint_timeout
setting         | 3600
unit            | s
category        | Write-Ahead Log / Checkpoints
short_desc      | Sets the maximum time between automatic WAL checkpoints.
extra_desc      |
context         | sighup
vartype         | integer
source          | configuration file
min_val         | 30
max_val         | 86400
enumvals        |
boot_val        | 300
reset_val       | 3600
sourcefile      | /etc/postgresql/10/main/postgresql.conf
sourceline      | 208
pending_restart | f
```

pg_settings category

```
postgres=# select distinct(category) from pg_settings ;
                             category
-------------------------------------------------------------------
 Write-Ahead Log / Settings
 Client Connection Defaults / Locale and Formatting
 Version and Platform Compatibility / Other Platforms and Clients
 Replication
 Query Tuning / Genetic Query Optimizer
 Write-Ahead Log / Archiving
 Resource Usage / Memory
 Statistics / Monitoring
 Reporting and Logging / Where to Log
 Resource Usage / Kernel Resources
 Preset Options
 Error Handling
 Replication / Sending Servers
 Reporting and Logging / What to Log
 Lock Management
 Connections and Authentication / Security and Authentication
 Process Title
 Resource Usage / Disk
 Replication / Standby Servers
 Autovacuum
 Write-Ahead Log / Checkpoints
 Client Connection Defaults / Shared Library Preloading
 Connections and Authentication / Connection Settings
 Query Tuning / Planner Method Configuration
 Replication / Master Server
 Statistics / Query and Index Statistics Collector
 Developer Options
 Resource Usage / Background Writer
 Resource Usage / Asynchronous Behavior
 Query Tuning / Other Planner Options
 File Locations
 Client Connection Defaults / Statement Behavior
 Reporting and Logging / When to Log
 Resource Usage / Cost-Based Vacuum Delay
 Query Tuning / Planner Cost Constants
 Client Connection Defaults / Other Defaults
 Version and Platform Compatibility / Previous PostgreSQL Versions
(37 rows)
```

pg_settings context

```
postgres=# select distinct(context) from  pg_settings ;
      context
-------------------
 postmaster
 superuser-backend
 user
 internal
 backend
 sighup
 superuser
(7 rows)
```

postgresql.conf - main config file

- We usually advise not to change the order of the settings when you edit them manually
- **postgresql.conf** supports includes
- Always check **pg_settings** if you doubt...
- And off we go

listen_addresses

- We usually use `*` or `127.0.0.1`. One can use `127.0.0.1` if Postgres works together with pgbouncer.
- If Postgres listens on external IP address, this IP address must be protected by a firewall.
- There are *arguments* that using UNIX-socket could bring more performance, but generally using TCP is more convenient because of keepalives.

max_connections

- Client connection cause Postgres to spawn a "heavy" Unix-Process
- Thats why things like `max_connections = 1000` would never work
- A much better idea: `max_connections = 100` or `200` and really small pool sizes in pgbouncer or another connection pooler

superuser_reserved_connections

- When all of `max_connections` are utilized, DBA needs to connect to a database server in order to troubleshoot such situation
- Should be at least 5, better 10

Don't forget

Postgres already have some processes:

```
1295  0:00 /usr/lib/postgresql/14/bin/postgres -D /var/lib/postgresql/14/main -c config_file=/etc/postgresql/14/main/postgresql.conf
1315  0:00 postgres: 14/main: checkpointer
1316  0:00 postgres: 14/main: background writer
1318  0:00 postgres: 14/main: walwriter
1320  0:00 postgres: 14/main: autovacuum launcher
1321  0:00 postgres: 14/main: stats collector
1322  0:00 postgres: 14/main: logical replication launcher
```

Keepalives

- `tcp_keepalives_idle = 5` If network is unstable, 5 seconds can really help
- `tcp_keepalives_interval = 1`
- `tcp_keepalives_count = 5`
- Even if you have a very good network quality between your app and database, it could became *suddenly* unstable

shared_buffers

- Rule of Thumb: 25% of RAM
- But to use 16/32/64Gb of `shared_buffers` efficiently, fast discs are required
- If the database is definitely smaller than RAM, 75% of RAM for `shared_buffers` can also work

huge_pages

- Rule of thumb: when there are at least 8-16Gb `shared_buffers`, using of Huge Pages is recommended
- `huge_pages = on` (and not `try`)
- Huge Pages should be first enabled in kernel
- `vm.nr_overcommit_hugepages` and `vm.nr_hugepages`

temp_file_limit

- Temporary file limit in Kb per session
- good starting point is **number of sessions * temp_file_limit < 10% of your disks**
- but it is very individual for particular server and application

work_mem

- RAM per process, Postgres workers use this RAM for sorting, hash joins etc.
- 128Mb is a good starting point
- To high setting could cause OOM
- Could be individually configured for each session

maintenance_work_mem

- Same as `work_mem` but for superuser connections
- 256-512Mb, if there is enough RAM
- Could be quite helpful for `CREATE INDEX CONCURRENTLY`
- `autovacuum_work_mem` is a part of `maintenance_work_mem`, can be smaller

Write Ahead Log

- `wal_level = replica` unless zou use logical replication
- `checkpoint_timeout = 60min`, if it is by given recovery target acceptable, could gain performance improvement
- `max_wal_size = 32GB`
- `checkpoint_completion_target = 0.9`

bgwriter

- Background Writer helps Checkpointer to send unused dirty pages to disk
- Regret to say, it is not the best part of PostgreSQL codebase
    - All settings to maximum:
        - `bgwriter_delay = 10ms`
        - `bgwriter_lru_maxpages = 1000`
        - `bgwriter_lru_multiplier = 10.0`
    - It might help if your disks are not the best

Must have optimizer settings

- `effective_cache_size = 2 * shared_buffers` or less
- `default_statistics_target = 100`

Autovacuum

- There is no practical use case, where autovacuum should be disabled
- Deserves a separate talk (or maybe a separate tutorial)
- We try to provide some good starting points

Autovacuum

- Autovacuum checks tables one by another, to figure out if they need to be vacuumed
- It starts vacuuming if either `autovacuum_vacuum_threshold` or `autovacuum_vacuum_scale_factor` achieved (whatever comes first)
- `autovacuum_vacuum_threshold = 50` 50 rows are modified
- `autovacuum_vacuum_scale_factor = 0.05` 5% of rows are modified

Autovacuum

- If there is a lot of write acivity in your database
- default `autovacuum_max_workers = 3` might be not enough
- Consider `autovacuum_max_workers = 10` but beware of IO

Autovacuum

- `autovacuum_freeze_min_age = 20000000` # 9.6 and older - default is most likely enough, older versions often require up to 1B
- `autovacuum_freeze_table_age = 15000000`
- `idle_in_transaction_session_timeout = '6h'`

Autovacuum also collects statistics

- `autovacuum_analyze_threshold = 50`
- `autovacuum_analyze_scale_factor = 0.05`

Autovacuum

- `autovacuum_naptime = 1s`
- `autovacuum_vacuum_cost_delay = 5ms`

Those settings were designed to reduced impact of vacuuming on the system, but that doesn't work, so minimize them.

Don't forget manual vacuum

- There are cases when you need it
- When an autovacuum parameter is not set, Postgres falls back to vacuum setting
- `vacuum_cost_delay = 0 # 0-100 milliseconds`
- `vacuum_cost_page_hit = 0 # 0-10000 credits`
- `vacuum_cost_page_miss = 1 # 0-10000 credits`
- `vacuum_cost_page_dirty = 10 # 0-10000 credits`
- `vacuum_cost_limit = 100 # 1-10000 credits`

Logging

```
log_directory = "/var/log/postgresql"
log_filename = "postgresql-%Y-%m-%d.log"
log_rotation_age = 1d
log_rotation_size = 0
log_min_error_statement = error
log_min_duration_statement = 1000 # -1 disabled, 0 -all, in ms
log_checkpoints = on
log_line_prefix = "%m %p %u@%d from %h [vxid:%v txid:%x] [%i] "
log_lock_waits = on
log_statement = "none"
log_replication_commands = on
log_temp_files = 0
log_timezone = "Europe/Berlin"
```

Logging

```
track_activities = on
track_counts = on
track_io_timing = on
track_functions = pl
track_activity_query_size = 8192
log_autovacuum_min_duration = 1000
```

pg_stat_statements

```
shared_preload_libraries = 'pg_stat_statements'
```

Do not forget to create extension in postgres database:

```
psql -d postgres -c "create extension pg_stat_statements"
```

```
pg_stat_statements.max = 10000
pg_stat_statements.track = top
pg_stat_statements.track_utility = false
pg_stat_statements.save = false
```

Replication

- `max_wal_senders = 3`
- `hot_standby = on`
- `max_standby_streaming_delay = 120s # max delay before canceling queries`
- `hot_standby_feedback = on`
    - `ERROR: canceling statement due to conflict with recovery`
    - you need it only if you have intensive OLTP on primary
    - do not switch it on until your replica caught it up!

Questions?

ik@dataegret.com