Introducing Crunchy Data Warehouse: A next-generation Postgres-native data warehouse.
Learn more →

Production Postgres

Managing Transaction ID Exhaustion (Wraparound) in PostgreSQL

Keith Fiske
Jan 31, 2019 · 9 min read

One of the most critical topics to understand when administering a PostgreSQL database is the concept of *transaction IDs* (TXID) and that they can be exhausted if not monitored properly. However, this blog post isn't going to go into the details of what it TXID exhaustion actually is. The **Routine Vacuuming** section of the documentation is probably one of the most important to read and understand so I will refer you there. What this blog post is going to cover is an easy way to monitor for it and what can be done to prevent it ever being a problem.

Monitoring

Most people initially aware of the issue consider the TXID wraparound itself to be the problem that they're monitoring for, but it's technically the exhaustion of the TXIDs that's the real issue. PostgreSQL is technically capable of handling things just fine if the TXID value wraps around. However, if the wraparound point is reached, and the TXIDs are close to being used up, that is why wraparound itself is such a cause for concern.

The following query is the one that we use at **Crunchy Data** in our **PGmonitor** tool to provide very simple data points to trend/alert on.

```
WITH max_age AS (
    SELECT 2000000000 as max_old_xid
        , setting AS autovacuum_freeze_max_age
        FROM pg_catalog.pg_settings
        WHERE name = 'autovacuum_freeze_max_age' )
, per_database_stats AS (
    SELECT datname
        , m.max_old_xid::int
```

```
    JOIN max_age m ON (true)
    WHERE d.datallowconn )
SELECT max(oldest_current_xid) AS oldest_current_xid
    , max(ROUND(100*(oldest_current_xid/max_old_xid::float))) AS percent_towards_wraparound
    , max(ROUND(100*(oldest_current_xid/autovacuum_freeze_max_age::float))) AS percent_towards_emergency_autc
FROM per_database_stats
```

The ***percent_towards_wraparound*** metric is the one that is really critical that an alert be set up for. Since it is using the *age()* function to determine the TXID value, it is taking into account if they are actually at the point of exhaustion to see if the wraparound is a real concern. If exhaustion is ever reached, the database will be forced to shut down and could cause an indeterminate amount of downtime in order to be fixed. This query has a little bit of a buffer in it since the upper boundary it checks (2 billion exactly) is less than the actual max integer value that causes the exhaustion. But it's close enough that this alert hitting 100% should be acted upon immediately.

The ***percent_towards_emergency_autovac*** metric is the additional value we recommend monitoring for, especially for systems that have never had this monitored for before (see notes on *Recent Freezing Benefits* below about when this alert priority can be lowered or removed). This watches for the database's highest TXID value reaching ***autovacuum_freeze_max_age***. This is a user-tunable value that has a default value of 200 million and when any table's highest TXID value reaches it, a higher priority autovacuum kicks in on that table. You'll recognize this special vacuum session because in pg*stat_activity it will be labelled ***(to prevent wraparound)_**. It is higher priority in the sense that it will run even if autovacuum is disabled and if that vacuum is manually cancelled, it will almost immediately restart again. It also takes some different internal, low-level locks, so it could cause slightly higher contention on those tables depending on how they're being used during the emergency vacuum. If you do run into contention/locking issues and they can be narrowed down to the emergency vacuum being the cause, it's perfectly safe to cancel it to allow your other transactions to finish. Just be aware that it will keep restarting until either that wraparound vacuum is able to be completed successfully or a manual vacuum is run

For databases with a high transaction rate, it could be beneficial to increase *autovacuum_freeze_max_age* to avoid that emergency vacuum period coming around quite so often. The main concern with increasing this is that it can increase the storage requirements in the *pg_xact* and *pg_commit_ts* folders in the data directory. Again, please read the **Routine Vacuuming** documentation linked above for what these storage requirements are when you adjust this setting. I've often set this value to 1 billion without much issue, but only when I'm sure wraparound is being monitored for and the disk space is available.

So when either of these alerts go off, how do you fix it?

Easy Fix

The easiest (but not necessarily the quickest) way to get the highest TXID age value back down is to force a vacuum on

```
vacuumdb --all --freeze --jobs=2 --echo --analyze
```

The *--all* option ensures all databases are vacuumed since the TXID is a global value. The *--freeze* option ensures a more aggressive vacuum is run to ensure as many tuples as possible are frozen in that table (see **Routine Vacuuming** for details on what freezing is). *--jobs=2* allows multiple vacuums to be run in parallel. This should be set as high as your system can handle to speed things up, but be careful setting it too high since it causes additional IO and faster WAL generation (increased disk usage). *--echo* just provides some minimal feedback so you can see some progression. *--analyze* ensures that statistics are updated. If time is of concern to get this vacuum run finished, this can be left off and run as a separate step later using the *--analyze-only* option.

Recent Freezing Benefits

Another benefit of the *--freeze* option I'll mention here can be a huge reduction in IO & WAL generation during future vacuum operations. PostgreSQL 9.6 introduced a feature that allows vacuum to be able to skip over a page if all tuples inside of it are marked as frozen. And PG11 improved on this even more for indexes. So if you have a lot of old tables that don't get writes anymore, this makes it so that when they do need to be vacuumed for any reason, it's a much, much less expensive operation. It also makes the ***percent_towards_emergency_autovac*** alert less of a concern since it won't be quite as much of an unexpected activity spike. So once you have things tuned well, you could consider this alert a low priority warning or maybe even remove it and just worry about monitoring for wraparound itself.

Per-Table Fix

If you cannot afford to do a cluster-wide vacuum and want to just get the TXID age under control as quickly as possible, it's certainly possible, but just involves more steps than just calling a single binary command. This is definitely the case if you've reached wraparound or are getting close.

```
SELECT datname
    , age(datfrozenxid)
    , current_setting('autovacuum_freeze_max_age')
FROM pg_database
ORDER BY 2 DESC;

  datname  |    age    | current_setting
-----------+-----------+-----------------
 postgres  | 170604895 | 200000000
 mydb      | 169088197 | 200000000
 template0 | 142915024 | 200000000
 template1 | 142914999 | 200000000
```

```
postgres=# SELECT c.oid::regclass
    , age(c.relfrozenxid)
    , pg_size_pretty(pg_total_relation_size(c.oid))
FROM pg_class c
JOIN pg_namespace n on c.relnamespace = n.oid
WHERE relkind IN ('r', 't', 'm')
AND n.nspname NOT IN ('pg_toast')
ORDER BY 2 DESC LIMIT 100;

                    oid                     |    age    | pg_size_pretty
--------------------------------------------+-----------+----------------
 pg_proc                                    | 170606208 | 936 kB
 pg_description                             | 170606208 | 480 kB
 pg_depend                                  | 109192353 | 1336 kB
 pg_attribute                               | 109192353 | 728 kB
 pg_shdepend                                |  89774141 | 2976 MB
 pg_db_role_setting                         |  77594841 | 16 kB
 pg_tablespace                              |  77594841 | 72 kB
 pg_pltemplate                              |  77594841 | 56 kB
 pg_auth_members                            |  77594841 | 16 kB
 pg_authid                                  |  77594841 | 72 kB
[...]
```

Here you can see there's only a few tables that have a high age, but in a more realistic table example, you'd likely see results closer to the 100 row limit with a higher age. Next we'll want to vacuum just these specific tables. If it's just a few tables, manually typing out the VACUUM commands isn't a big deal. But if you have 100+, that could be tedious and prone to typos. So we can use some string concatenation along with some psql commands to generate some statements and place them in a file to then be run automatically for us.

```
\t \o /tmp/vacuum.sql select 'vacuum freeze analyze verbose ' || oid::regclass || ';' from pg_class where rel
```

I highly recommend **checking out the psql documentation** for what all these slash commands do, but basically what it does is:

- Turn off column headers
- Send all output after this to the /tmp/vacuum.sql file
- Generate at most 100 VACUUM statement for each table that returns from the query
- Turn off file output and turn column headers back on
- Output each command run after this

Even if there's more than 100 tables to clean up, I usually do it in batches of 100 just to give IO and WAL generation some time to calm down. I'll usually try and get the max XID for any table at least down to 50% of *autovacuum_freeze_max_age*, usually even down to 30-40% if it's not going to be too much trouble. And also note that you will have to log into each database to run these VACUUM commands. You can log into *template1* to fix it if you want, but you won't be able to log into *template0*. *template0* can safely be ignored and let run to *autovacuum_freeze_max_age*since it's extremely small and will finish nearly instantaneously. If you're curious what these template databases are, **check the documentation**.

Conclusion

An earlier blog post of mine on **Per-Table Auto-Vacuuming Tuning** is another way to help prevent needing to do manual fixes like this for frequently written tables. And as I said earlier, if you're on PG9.6 and later, you'll likely only have to be concerned about doing this sort of manual intervention rarely and hopefully only ever once. So once you've gotten autovacuum tuned properly and your old, static data marked frozen, reaching *autovacuum_freeze_max_age* is not really much of a concern anymore since:

- Your frequently used tables are being vacuumed when they're supposed to be getting vacuumed
- Your static tables have been marked frozen and vacuum can skip over them rather quickly

That means all you have to worry about monitoring for is the actual exhaustion. It won't likely be an issue anymore unless the usage patterns of the database change, but it should **ALWAYS** be monitored for no matter what since reaching it is a guaranteed outage that you may not be ready for.

Enjoy this article?

You will love our newsletter!

Enter your email

Join The List

PRODUCTS

Crunchy Postgres

Crunchy Postgres for Kubernetes

Crunchy Bridge

Crunchy Certified PostgreSQL

Crunchy PostgreSQL for Cloud Foundry

Crunchy MLS PostgreSQL

Crunchy Spatial

RESOURCES

Customer Portal

Software Documentation

Postgres Tutorials

Crunchy Bridge Walkthrough

Postgres Operator Walkthrough

Blog

Events

SERVICES & SUPPORT

Enterprise PostgreSQL Support

Migrate from Heroku

Ansible

Red Hat Partner

Trusted PostgreSQL

Crunchy Data Subscription

COMPANY

About Crunchy Data

Team

News

Careers

Contact Us

Newsletter

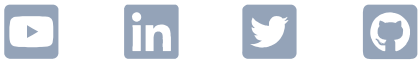
Branding

CRUNCHY DATA NEWSLETTER

Subscribe to the Crunchy Data Newsletter to receive Postgres content every month.

Enter your email

Join The List



© 2018-2025 Crunchy Data Solutions, Inc.