How Autovacuum Goes Wrong

And can we please make it stop doing that?

Robert Haas
VP, Chief Database Scientist
EDB

EDB™

Agenda

- Overview of Failure Modes
- Cost Limits
- `VACUUM` Slow or Stuck
- `VACUUM` Spinning
- `VACUUM` Skipped or Starvation
- Conclusion

Overview of Failure Modes

- *Slow*. Making forward progress, but not quickly enough.
- *Stuck*. Making no forward progress.
- *Spinning*. Running repeatedly on the same table but not accomplish anything useful.
- *Skipped*. autovacuum thinks that vacuuming is not needed, even though it is.
- *Starvation*. autovacuum can't vacuum everything that needs vacuuming.

Cost Limits

Cost Limits: The Problem

- `autovacuum_vacuum_cost_limit` and related settings are intended to prevent autovacuum from consuming too many resources.

- However, these resource limits can prevent autovacuum from getting enough vacuuming done.

- The amount of vacuum work that needs to be done depends on the size of the database, the rate at which dead tuples are being created, and the rates of XID and MXID consumption - but the default is a constant.

Cost Limits: The Consequences

- Cost limit problems are the single most frequent cause of “vacuum slow” problems and of “vacuum starvation” problems.
- Users often mistakenly believe that raising `autovacuum_max_workers` will fix things. It often does the opposite - and it’s never the first thing to change.
- In v12, we reduced the cost delay by 10x, which is equivalent to increasing the cost limit by 10x. This helped a lot.
- “Emergency mode” disregards the cost limit, but too late.

Cost Limits: Intuition

- Consider the “backlog” of tables which autovacuum would think need to be vacuumed if it looked at them, but it hasn’t looked at them yet.
- If the backlog is growing over time, the cost limit isn’t high enough, and possibly we also need more workers.
- We don’t really want to wait for the backlog to start growing: we want to get ahead of the problem.

Cost Limits: Solution Sketch

- Have a set of autovacuum workers that connect to each database and gather information:
  - Keep track of which tables need vacuuming now.
  - Estimate when the rest of the tables will need vacuuming in the future.
  - Estimate how long each table will take to vacuum when we process it.
- If the backlog is large, raised the cost limit or the worker count to try to bring the situation under control.
- If the backlog is small now but will grow sharply in the near future, get a head start.

Prioritization

- The same information that we use to track the vacuum backlog could also be used for prioritization, so that we try to do more important vacuums first.
- XID wraparound > MXID wraparound > bloat, but only in the limit. Being slightly beyond some XID cutoff is not more important than unlimited bloat.
- Prioritization seems significantly less important than cost limit adjustment, because we really need to do all the vacuuming.
  - If the table doesn't need to be vacuumed right now for good system performance, we shouldn't vacuum it at all until that changes.

VACUUM Slow or Stuck

VACUUM Slow or Stuck: Causes

- Everything is fine, just be patient.
- Cost limit too low.
- Waiting for some other PostgreSQL process.
- Slow disk.
- Corrupted indexes.

VACUUM Slow or Stuck: Fixes

- Waiting for some other PostgreSQL process.
  - Kill the other process.
  - Doing this automatically might be possible in some cases, but sounds a bit scary.
- Slow disk.
  - Get a new disk.
  - Not much to do in software.
- Corrupted indexes.
  - DROP or REINDEX.
  - Even if we detect this automatically and error out, it would just turn this into the "spinning" case, unless we had some mechanism to delay retrying.

VACUUM Spinning

VACUUM Spinning: Causes

- Everything is fine, you just have a lot of write activity in that table.
- `VACUUM` is failing with an `ERROR` and autovacuum keeps retrying it.
- Autovacuum keeps thinking the table needs vacuuming, but when `VACUUM` actually processes it, nothing gets any better.

VACUUM Spinning: Solutions

- `VACUUM` is failing with an `ERROR` and autovacuum keeps retrying it.
  - Fix whatever is causing the error.
  - Can’t really do anything in the code.

- Autovacuum keeps thinking the table needs vacuuming, but when `VACUUM` actually processes it, nothing gets any better (“useless vacuuming”).
  - By far the most common cause of this problem.
  - Generally happens because something is holding back xmin, meaning that the old tuples are not yet dead, and thus not removable.

VACUUM Spinning: What’s holding back xmin?

- A long-running transaction (check `pg_stat_activity`) that needs to be killed.
- An unused replication slot (check `pg_replication_slots`) that needs to be dropped or reconnected.
- An old prepared transaction (check `pg_prepared_xacts`) that needs to be committed or aborted.

Useless Vacuuming: Dumbest Possible Fix

- If vacuuming the table does nothing, don't do it again until `xmin` advances.
- We'll still do at least one useless vacuum.
- Even if `xmin` advances, vacuuming might still be useless.
- Even if `xmin` advances and vacuuming is no longer useless, it might be *nearly* useless.

Useless Vacuuming: Smarter Fix?

- Find a way to answer this question: What is the next currently non-vacuumable XID such that, when it becomes vacuumable, we’ll be able to clean up a reasonable number of tuples?

- If we count the number of recently-dead tuples in each range of XIDs, we can answer this question well *provided that* we pick the right ranges.

- e.g. If we count recently dead tuples with XIDs 1001-2000, it might be that all the tuples in this table in this range have xmax = 1001, but we’ll wait for XID 2000 to be vacuumable before doing anything.

VACUUM Starved or Skipped

Autovacuum Is Not Running VACUUM on My Table - Why?

- *Starved*: All autovacuum worker processes have been busy doing other things, and therefore your table is not being checked.

- *Skipped*: Autovacuum decides the table doesn't need vacuuming so it doesn't do anything.

- It's usually easy to tell the difference by looking at the number of autovacuum workers in `pg_stat_activity` compared to `autovacuum_max_workers`, and how long they've been running.

VACUUM Starved: Causes

- Cost limit too low.
- Need more workers.

VACUUM Skipped: Causes

- Misconfiguration, e.g. `autovacuum_enabled=false`.
- Missing statistics, e.g. `pg_stat_reset()` or corrupted stats file.
- Large table with default scale factor.
  - The new TID store may help, by preventing multiple index passes, which are very bad.
  - However, waiting until we have terabytes of bloat in the table is probably not right.
  - Possible code fix: Cap the result of the scale factor calculation, or grow sublinearly.

Conclusion

Triggers vs. Goals

- Autovacuum doesn't try to ensure that operations finish at the right time, only that they start at the right time.
- Forces a very conservative value of `autovacuum_freeze_max_age`.
- May be part of why `autovacuum_vacuum_scale_factor` doesn't work well on very large tables.
- Makes it hard to do proper prioritization.

Summary of Improvement Ideas

- Automatically tune the cost limit - and worker count - in response to pressure.
- Prioritization, maybe.
- Avoid useless vacuuming when something is holding back xmin.
- Rethink scale factor mechanism for large tables.
- Goal-based rather than trigger-based.

Thank You!

Any questions?

EDB™