Postgres v15 - a billion transactions later

Posted on November 9, 2022

As the last Postgres release, carrying the number 15, seemed kind of an “anniversary” release to me, I thought it would be fun / useful to run a series of tests to try to gauge the scale of improvements over the last 5 years, i.e. putting v10 and v15 in the ring. It’s of course a very hard task to do in a fair and also meaningful way (see my last try with mixed results here) as so much has changed in between and there are too many use cases out there and not enough time. But as always - it’s super easy to fire off “pgbench”, let it churn for a while and then try to analyze the numbers - did something change and what might be the reason? Fun stuff!

Usually my such test sessions run for some hours or so…but this time, to pay proper “anniversary” respects and given the fact that there are actually very few long-term Postgres perf test pieces out there, I thought what the heck - let’s do a round **1B pgbench transactions**! And see firstly how long it takes, and in what state are the tables and indexes after that for respective Postgres versions, and if there are visible performance degradations!

TLDR; - not much difference from query runtime side at all 😏

Test setup

Hardware: 2x on-prem (under my desk) old workstations, 4 CPU (Intel i5 and Xeon E3 both @ 3.30GHz, no hyperthreading), 16GB RAM, SATA SSD

OS: Ubuntu 20.04 Server

Postgres: latest v10 and v15 (10.22, 15.0) from the official PGDG repos

Working set size: pgbench scale 5000 ~ 73 GB DB size, i.e. an active set of 5x RAM

Measuring method: Postgres built-in “pg_stat_statements” extension stats aggregation and custom SQL

Level of parallelism: decided to go for a full load test this time with client count of 8 (2x CPU), that fully utilized the systems (a few % idle, ~40% iowait on avg)

Pgbench test mode: *--skip-some-updates* e.g. remove updates on the tiny tables that suffer from lock contention + **1 extra index on pgbench_accounts** (bid, abalance). Prepared statements mode was used.

Test duration: 1B TX, i.e. 125M for each pgbench client

Postgres config: Pgtune OLTP + a few custom changes:

```
# max_parallel_maintenance_workers = 2 # Not supported on v10
unix_socket_directories='/tmp' # To run under any OS user conveniently
shared_preload_libraries='pg_stat_statements'
wal_compression=on # Should be enabled on all instances basically
track_io_timing=on # To get IO call duration in pg_stat_statements
synchronous_commit=off # Don't want to test disk COMMIT throughput
```

The full test script can be found here

Results

So after some 11 days I got a ping in my inbox for the slower machine, checked the logs for errors…and without further ado, here's the stuff extracted from pg_stat_statements and some ad-hoc SQLs executed after the testing (averaged from the runs on two different servers).

Metric	Change (%)
Avg query runtime change over 3 pgbench queries	+0.8%
DB size growth change after 1B TX	101 GB vs 69 GB (-46%)
WAL generated change	6.3 TB vs 7.7 TB (+18%)
Shared buffers hit pct change	92.7 vs 93.2 (+5.4%)
Tuple percent (pgstattuple) on pgbench_accounts	87.2 vs 88.6 (+1.6%)
Autovacuum count on pgbench_accounts	5 vs 9 (+80%)
Pgbench_accounts table size	65 GB vs 64 GB (-1.5%)
Pgbench_accounts indexes size	49 GB vs 20 GB (-59.1%)

Detailed SQL exec stats from pg_stat_statements

Query	Mean exec time for v15 (ms)	Exec time change (%)	Stddev change (%)
SELECT abalance FROM pgbench_accounts WHERE aid = $1	0.00932	+3.0 %	+20.6 %
UPDATE pgbench_accounts SET abalance = abalance + $1 WHERE aid = $2	2.91	+0.5 %	+59.5 %
INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES ($1, $2, $3, $4, CURRENT_TIMESTAMP)	0.0541	-1.1 %	-0.77 %

Note here that the faster INSERT and SELECT queries are in microseconds range so things are already nearing the timing / measurement error levels and we could also basically just ignore these.

PS A full dump of my run's *pg_stat_statements* data can be found here

Key learnings

I hope the above numbers illustrate a bit what happens to your DB after a billion simple OLTP transactions. From the learnings side though I guess we can only say a few things with definite certainty:

- Postgres managed to chew through 1B TX quite fast (~10d) on my lowly test hardware (which actually represents an average Postgres cloud instance pretty well looking at my current company's inventory), thus I suspect transaction wraparound (2B counter) is still a topic for high-churn Postgres instances and DBAs should watch out.
- Execution times for low-rowcount indexed OLTP access haven't basically changed at all in recent years / versions - similar to what I found out in my previous test. Probably good news for those who can't upgrade too easily though - you don't have to sweat too much, Postgres was very good in that area already 5 years ago and has your back!
- Hardware (disks) matter - although test boxes are very similarly specced (on purpose), one box did had an older slower SSD (as it later appeared after googling) and total test runtime was due to that a good third longer.
- v15 is definitely better on the disk consumption side - a relatively big 46% reduction for such a short period is definitely the biggest surprise for me in this test. Thus if you have slow storage and the working set has slipped heavily out of memory then one should definitely upgrade ASAP.
    - The disk reduction by the way comes from recent indexing improvements like de-duplication and bottom-up deletion as bloat levels were more or less same.
- v15 had a bit more randomness ("pg_stat_statements.stddev_exec_time") at play - this is probably due to more aggressive background Autovacuum settings and also maybe due to more complex index handling algorithms? A bit weird though, given general cache hit ratio was improved - but that matches actually with general finding from my previous test also, so must be something there.
- Throwing in one key mystery also - the considerably bigger WAL "consumption" for v15 is a bit hard to explain without storing some of it and running "pg_waldump", but most probably it's again due to more aggressive autovacuum / freeze default configuration as "wal_level" was at default "replica" for both. I hope 🤔

In the end, as usual, some kind of a disclaimer - remember that ~10 days is still a very narrow time frame considering the lifecycle of a typical application, and this was a very simple OLTP test with only 3 statements - in real-world cases one would have many other queries, some "annoying" Cronjobs or backup jobs that mess up your cache etc, so as always - for critical growth-prone applications one should run a load test tuned to specific schema and usage patterns.

Also if you plan to run your own *pgbench*, but skipped the above test setup part - note that in addition to the standard *pgbench* schema, I added an index to the *pgbench_accounts* table to enable hypothetical quick "top X accounts for branch Y" queries (to try to sell them crypto investment products 🤑), as the default only-one-index is a very unrealistic use case for OLTP, catering for too many HOT updates.

Notes for the next run - more monitoring

Wish I had tracked TPS (transactions per second) better, to see how or if it changed over time - currently just need to be happy with an approximate average of 1.3K. And also something for "relfrozenxid" age movements and background writing (*pg_stat_bgwriter*) stats.

Tags: postgres performance pgbench

← PREVIOUS POST