stackoverflow

Postgresql Truncation speed

We're using `Postgresql 9.1.4` as our db server. I've been trying to speed up my test suite so I've stared profiling the db a bit to see exactly what's going on. We are using database_cleaner to truncate tables at the end of tests. YES I know transactions are faster, I can't use them in certain circumstances so I'm not concerned with that.

What I AM concerned with, is why TRUNCATION takes so long (longer than using DELETE) and why it takes EVEN LONGER on my CI server.

Right now, locally (on a Macbook Air) a full test suite takes 28 minutes. Tailing the logs, each time we truncate tables... ie:

```
TRUNCATE TABLE table1, table2  -- ... etc
```

it takes over 1 second to perform the truncation. Tailing the logs on our CI server (Ubuntu 10.04 LTS), take takes a full 8 seconds to truncate the tables and a build takes 84 minutes.

When I switched over to the `:deletion` strategy, my local build took 20 minutes and the CI server went down to 44 minutes. This is a *significant* difference and I'm really blown away as to why this might be. I've tuned the DB on the CI server, it has 16gb system ram, 4gb shared_buffers... and an SSD. All the good stuff. How is it possible:

a. that it's SO much slower than my Macbook Air with 2gb of ram
b. that TRUNCATION is so much slower than DELETE when the postgresql docs state explicitly that it should be much faster.

Any thoughts?

postgresql delete truncate database-performance

asked Jul 10 '12 at 18:25
brad
8,262 ●5 ●45 ●90

Are you running the tests and database on macbook, and tests and database on CI server? Are tests and database on the same machine? – Szymon Guz Jul 10 '12 at 18:31

Btw, you are doing it wrong... you cannot clear database AFTER the test. You should do it BEFORE running the test. You cannot be sure that the database is cleared after tests. – Szymon Guz Jul 10 '12 at 18:32

What postgresql.conf parameters are in use? I'm wondering if you're running with fsync=off (ok if you don't mind losing all your data, like in testing) in which case the balance between `DELETE` and `TRUNCATE` could be different. I'd also be interested in your `shared_buffers` . – Craig Ringer Jul 10 '12 at 23:47

When you mean "using transactions" do you mean opening a transaction, doing some tests, then rolling it back? Because in my view that's only half a test. So much can happen at `COMMIT` time if you're using `SERIALIZABLE` transactions, `DEFERRABLE INITIALLY DEFERRED` constraints, etc, that committing test changes seems wise. – Craig Ringer Jul 10 '12 at 23:48

How long is "etc" btw, ie how many tables are being truncated in one go? Are they very small tables, or do they contain a bit of data? I won't be too shocked if tiny tables are quicker to `DELETE FROM` than `TRUNCATE` as `TRUNCATE` has to allocate a new backing file, write the headers, swap the old one with it, *flush the buffer caches for the table*, and fsync. I suspect the docs probably need to be updated to reflect that `TRUNCATE` is lots faster with big tables, but not necessarily with tiny/empty ones. – Craig Ringer Jul 10 '12 at 23:51

add / show **3** more comments

start a bounty

3 Answers

This has come up a few times recently, both on SO and on the PostgreSQL mailing lists.

The *TL;DR* for your last two points:

(a) The bigger shared_buffers may be why TRUNCATE is slower on the CI server. Different fsync configuration or the use of rotational media instead of SSDs could also be at fault.

(b) `TRUNCATE` has a fixed cost, but not necessarily slower than `DELETE`, plus it does more work. See the detailed explanation that follows.

UPDATE: A significant discussion on pgsql-performance arose from this post. See this thread.

UPDATE 2: Improvements have been added to 9.2beta3 that should help with this, see this post.

Detailed explanation of `TRUNCATE` vs `DELETE FROM`:

While not an expert on the topic, my understanding is that `TRUNCATE` has a nearly fixed cost per table, while `DELETE` is at least O(n) for n rows; worse if there are any foreign keys referencing the table being deleted.

I always assumed that the fixed cost of a `TRUNCATE` was lower than the cost of a `DELETE` on a near-empty table, but this isn't true at all.

`TRUNCATE table;` does more than `DELETE FROM table;`

The state of the database after a `TRUNCATE table` is much the same as if you'd instead run:

- `DELETE FROM table;`
- `VACCUUM (FULL, ANALYZE) table;` (9.0+ only, see footnote)

... though of course `TRUNCATE` doesn't actually achieve its effects with a `DELETE` and a `VACUUM`.

The point is that `DELETE` and `TRUNCATE` do different things, so you're not just comparing two commands with identical outcomes.

A `DELETE FROM table;` allows dead rows and bloat to remain, allows the indexes to carry dead entries, doesn't update the table statistics used by the query planner, etc.

A `TRUNCATE` gives you a completely new table and indexes as if they were just `CREATE`ed. It's like you deleted all the records, reindexed the table and did a vacuum full.

If you don't care if there's crud left in the table because you're about to go and fill it up again, you may be better off using `DELETE FROM table;`.

Because you aren't running `VACCUM` you will find that dead rows and index entries accumulate as bloat that must be scanned then ignored; this slows all your queries down. If your tests don't actually create and delete all that much data you may not notice or care, and you can always do a `VACCUM` or two part-way through your test run if you do. Better, let aggressive autovaccum settings ensure that autovaccum does it for you in the background.

You can still `TRUNCATE` all your tables after the *whole* test suite runs to make sure no effects build up across many runs. On 9.0 and newer, `VACUUM (FULL, ANALYZE);` globally on the table is at least as good if not better, and it's a whole lot easier.

IIRC Pg has a few optimisations that mean it might notice when your transaction is the only one that can see the table and immediately mark the blocks as free anyway. In testing, when I've wanted to create bloat I've had to have more than one concurrent connection to do it. I wouldn't rely on this, though.

`DELETE FROM table;` is very cheap for small tables with no f/k refs

To `DELETE` all records from a table with no foreign key references to it, all Pg has to do a sequential table scan and set the `xmax` of the tuples encountered. This is a very cheap operation - basically a linear read and a semi-linear write. AFAIK it doesn't have to touch the indexes; they continue to point to the dead tuples until they're cleaned up by a later `VACCUM` that also marks blocks in the table containing only dead tuples as free.

`DELETE` only gets expensive if there are *lots* of records, if there are lots of foreign key references that must be checked, or if you count the subsequent `VACUUM (FULL, ANALYZE) table;` needed to match `TRUNCATE` 's effects within the cost of your `DELETE` .

In my tests here, a `DELETE FROM table;` was typically 4x faster than `TRUNCATE` at 0.5ms vs 2ms. That's a test DB on an SSD, running with `fsync=off` because I don't care if I lose all this data. Of course, `DELETE FROM table;` isn't doing all the same work, and if I follow up with a `VACCUM (FULL, ANALYZE) table;` it's a much more expensive 21ms, so the `DELETE` is only a win if I don't actually need the table pristene.

`TRUNCATE table;` does a lot more fixed-cost work and housekeeping than `DELETE`

By contrast, a `TRUNCATE` has to do a lot of work. It must allocate new files for the table, its TOAST table if any, and every index the table has. Headers must be written into those files and the system catalogs may need updating too (not sure on that point, haven't checked). It then has to replace the old files with the new ones or remove the old ones, and has to ensure the file system has caught up with the changes with a synchronization operation - fsync() or similar - that usually flushes all buffers to the disk. I'm not sure whether the the sync is skipped if you're running with the (data-eating) option `fsync=off` .

I learned recently that `TRUNCATE` must also flush all PostgreSQL's buffers related to the old table. This can take a non-trivial amount of time with huge `shared_buffers`. I suspect this is why it's slower on your CI server.

The balance

Anyway, you can see that a `TRUNCATE` of a table that has an associated TOAST table (most do) and several indexes could take a few moments. Not long, but longer than a `DELETE` from a near-empty table.

Consequently, you might be better off doing a `DELETE FROM table;`.

--

Note: on DBs before 9.0, `CLUSTER table_id_seq ON table; ANALYZE table;` or `VACCUM FULL ANALYZE table; REINDEX table;` would be a closer equivalent to `TRUNCATE`. The `VACUUM FULL` impl changed to a much better one in 9.0.

edited Apr 29 at 23:56 answered Jul 11 '12 at 0:24

Craig Ringer
24.8k ●4 ●19 ●59

2 And they have different types of locks als well: table lock vs row lock. – Frank Heikens Jul 11 '12 at 4:57

Thanks for the comprehensive answer! According to the docs >> It [TRUNCATE] has the same effect as an unqualified DELETE on each table, but since it does not actually scan the tables it is faster. Furthermore, it reclaims disk space immediately, rather than requiring a subsequent VACUUM operation. >> So I don't think it actually vacuums after a truncate. Are you suggesting also that the fact taht I have 4GB shared_buffers is actually a detriment to performance? – brad Jul 11 '12 at 18:19

@brad For the specific case of `TRUNCATE`, yes, I'm saying that my understanding is that big `shared_buffers` may slow things down. I haven't tested this myself but that's how it sounds from ML discussion. And no, there is no `VACCUM` done after a truncate - while truncate *has the effect of* a `DELETE FROM` followed by a `VACUUM FULL ANALYZE;`, it doesn't actually work that way or perform those steps. – Craig Ringer Jul 12 '12 at 1:46

ah ok, thanks for the clarification. – brad Jul 12 '12 at 13:54

thanks for the link to that ML btw... it's great to see the conversations that surround topics such as this – brad Jul 12 '12 at 13:56

add / show **3** more comments

A couple of alternate approaches to consider:

- Create a empty database with static "fixture" data in it, and run the tests in that. When you are done, just just drop the database, which should be fast.
- Create a new table called "test_ids_to_delete" that contains columns for table names and primary key ids. Update your deletion logic to insert the ids/table names in this table instead, which will be much faster than running deletes. Then, write a script to run "offline" to actually delete the data, either after a entire test run has finished, or overnight.

The former is a "clean room" approach, while latter means there will be some test data will persist in database for longer. The "dirty" approach with offline deletes is what I'm using for a test suite with about 20,000 tests. Yes, there are sometimes problems due to having "extra" test data in the dev database but at times. But sometimes this "dirtiness" has helped us find and fixed bug because the "messiness" better simulated a real-world situation, in a way that clean-room approach never will.

answered Jul 10 '12 at 18:39

Mark Stosberg
1,926 ●8 ●21

add comment

Brad, just to let you know. I've looked fairly deeply into a very similar question.

Related question: 30 tables with few rows - TRUNCATE the fastest way to empty them and reset attached sequences?

Please also look at this issue and this pull request:

https://github.com/bmabey/database_cleaner/issues/126

https://github.com/bmabey/database_cleaner/pull/127

Also this thread: http://archives.postgresql.org/pgsql-performance/2012-07/msg00047.php

I am sorry for writing this as an answer, but I didn't find any comment links, maybe because there are too much comments already there.

edited Sep 19 '12 at 2:43

answered Jul 14 '12 at 10:50
Stanislaw
317 ●1 ●6 ●23

hey thanks stanislaw. I actually saw those posts which prompted me to upgrade db cleaner to use the mass truncation. That, however, did little to help me. Still on PG it seems the deletion strategy is significantly faster, which is what I've ended up using. – brad Jul 16 '12 at 1:10

add comment

Not the answer you're looking for? Browse other questions tagged postgresql delete truncate database-performance or ask your own question.