PostgreSQL Adores a Vacuum

Quinn Weaver <quinn@fairpath.com>

http://fairpath.com/vacuum-slides.pdf

Background: MVCC

When you DELETE a row in PostgreSQL, it doesn't really go away.

It just gets marked invisible to transactions that happen after the DELETE.

PostgreSQL implements UPDATE like DELETE-plus-INSERT.

xmin	***xmax***	**id**	**product_name**	**price**
3		4	dog food	30
17		5	turtle food	15
250	500	6	cat food	20
500		6	cat food	25

xmin and xmax are hidden columns.

They determine which transactions see which row versions.

Price of dog food hasn't changed.

Price of cat food was UPDATEd recently; transactions 250 to 500 see $20, while newer transactions see $25.

MVCC: the good

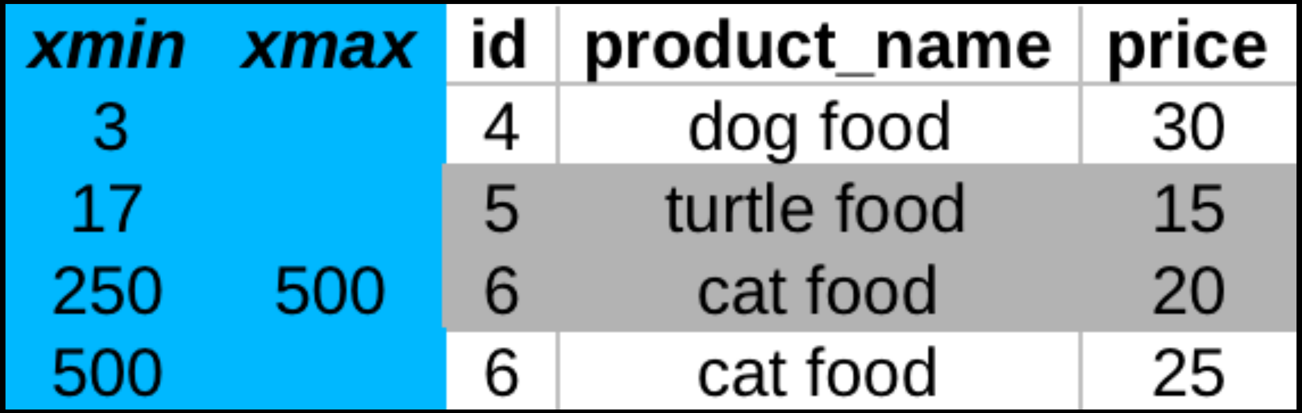
This versioning scheme is called MVCC (= multi-version concurrency control).

Efficient

No locking

Scales to high number of concurrent transactions.

MVCC: the bad



xmin	*xmax*	id	product_name	price
3		4	dog food	30
17		5	turtle food	15
250	500	6	cat food	20
500		6	cat food	25

The $20 cat food row is visible to old transactions.

Eventually it will be visible to no one.

We call this "bloat."

(auto)vacuum removes bloat. More on this soon.

MVCC: the bad

Indexes too get bloat:

leaf node t_tid's that no longer point at live rows

and inner nodes that point at them

and inner nodes that point at those inner nodes

et cetera

A btree index page has to be empty before it's removed by (auto)vacuum.

(cf. src/backend/access/nbtree/README)

Why is bloat bad?

Bloat fills up pages

… so you have to read more pages to get the data you want.

This slows things down

And eats memory

Your working set gets bigger, maybe doesn't fit in RAM

Why is bloat bad?

Bloated btree indexes take more steps to traverse

and occupy more space in memory.

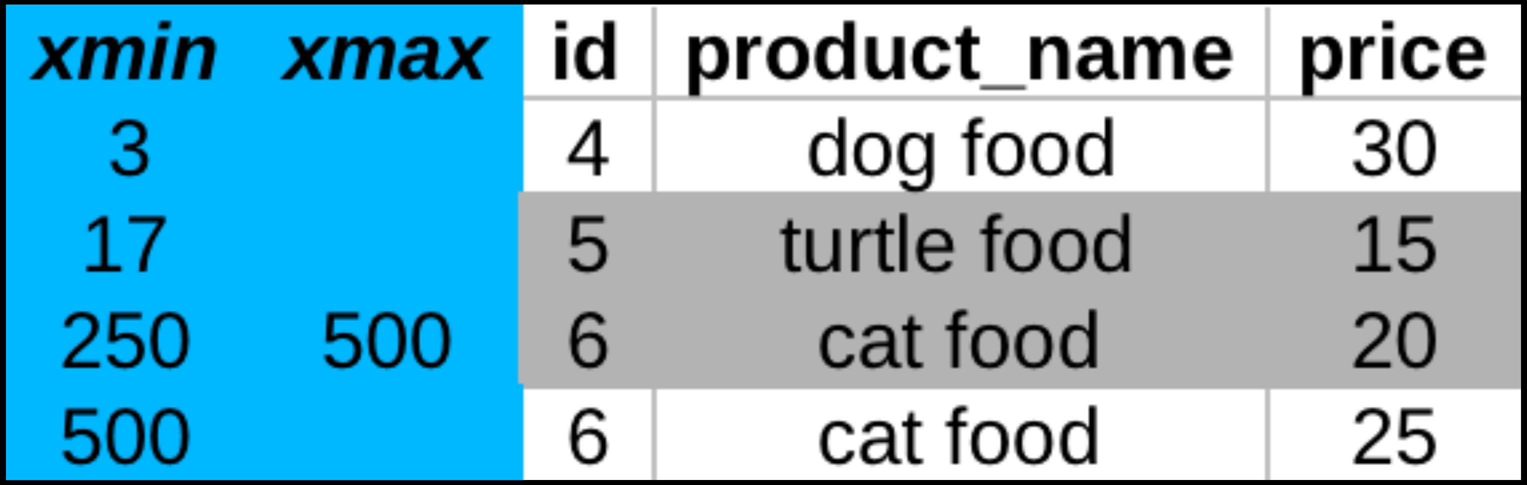
Indexes can get really big.

Remember, autovacuum doesn't help much.

Why is bloat bad?

And, of course, bloat eats up disk space.

Bloat: what to do about it?



xmin	*xmax*	id	product_name	price
3		4	dog food	30
17		5	turtle food	15
250	500	6	cat food	20
500		6	cat food	25

Someone needs to garbage-collect "$20 cat food."

= the autovacuum daemon.

Works out of the box since PostgreSQL 8.1 (2005).

Usually does a pretty good job.

The autovacuum daemon

But for high-traffic sites, the defaults aren't good enough.

autovacuum can't keep up.

Bloat grows out of control.

How do you know if this is happening to you?

Measuring bloat

git clone https://github.com/pgexperts/pgx_scripts

We wrote some queries.

PostgreSQL-Licensed

Measuring bloat: table bloat

```
 databasename | schemaname |  tablename  | can_estimate | est_rows  | pct_bloat | mb_bloat | table_mb
--------------+------------+-------------+--------------+-----------+-----------+----------+-----------
 my_database  | public     | huge_table  | t            | 180395000 |        64 | 45848.25 | 71777.266
 my_database  | public     | big_table_1 | t            |  70566100 |        37 |  9357.82 | 25277.641
 my_database  | public     | big_table_2 | t            |  78034400 |        34 |  9214.13 | 26818.281
(3 rows)
```

This shows potential problem tables only, not all tables.

Note huge_table has very high absolute bloat

But low percentage bloat.

That's probably OK!

Measuring bloat

It's normal to have have some bloat!

UPDATEs and DELETEs happen all the time.

autovacuum marks old rows as bloat.

Then their space get reused for new rows

(in the same table).

Measuring bloat

As much as 75% bloat can be normal, but It Depends.

The important thing is rate of growth.

Use a cron job (weekly?)

Immediately after autovacuum, bloat will be the same

Again, vacuum merely marks space as reusable.

Measuring bloat: index bloat

```
$ git clone https://github.com/pgexperts/pgx_scripts
$ psql -f index_bloat_check.sql my_database

 database_name  | schema_name | table_name | index_name | bloat_pct | bloat_mb | index_mb | table_mb  | index_scans
----------------+-------------+------------+------------+-----------+----------+----------+-----------+-------------
 my_database    | public      | foo        | foo_idx1   |        76 |     6053 | 7935.242 | 17479.820 |   403646039
 my_database    | public      | foo        | foo_idx2   |        74 |     5357 | 7239.094 | 17479.820 |   129716832
 my_database    | public      | bar        | bar_idx    |        56 |     1631 | 2915.008 |  5029.945 |     8951148
 my_database    | public      | batz       | batz_idx   |        92 |       75 |   81.063 |    27.734 |      179949
 my_database    | public      | quux       | quux_pkey  |        60 |       19 |   31.906 |    39.063 |    30449474
(5 rows)
```

batz_idx is small, but…

has high percent bloat.

It is much larger than the table data.

foo_idx1, foo_idx2: high absolute bloat

Fixing bloat

So what if bloat keeps growing?

That means autovacuum is not keeping up

(with UPDATEs and DELETEs).

Let's fix it!

Fixing past bloat

Bloat = technical debt.

Pay it down.

If you have a lot of bloat, your best bet is pg_repack.

pg_repack

https://github.com/reorg/pg_repack

How it works:

pg_repack makes a new, non-bloated table

containing only the non-bloat part of your table (including indexes).

Then it fiddles with the system catalogs to swap in the new table for the old one.

So much better than VACUUM FULL (which AccessExclusiveLocks the table the whole time).

pg_repack caveats

pg_repack does take a brief AccessExclusiveLock when it finishes.

If it can't acquire that lock, eventually it will cancel competing queries

(or even kill backends!)

Unpredictable: you can't control when it will finish.

pg_repack caveats

Requires free space = 2 * size of table you're repacking.

This could be a problem if you're repacking to regain scarce space!

Lots of I/O, since it's reading/rewriting the whole table.

Requires installing an extension, so it doesn't work on RDS.

Flexible Freeze

git clone https://github.com/pgexperts/flexible-freeze

Takes a different approach from pg_repack:

An extra-aggressive manual VACUUM.

Marks rows reusable; doesn't repack.

Run as a cron job nightly (or whenever traffic is low).

Flexible Freeze

Good practice: cron job =

  Bloat query before

  flexible_freeze.py [OPTIONS]

You can give FF a "hard" cutoff (stop vacuuming mid-table) or soft (finish current table).

More on FF soon…

Indexes are special

Next we'll talk about preventing future bloat.

This mostly means tuning autovacuum.

But we've noted (auto)vacuum works poorly for indexes.

What to do?

De-bloating indexes: pg_repack

The easy way: just pg_repack the index's table.

A good idea if the table is bloated

Or if there are a bunch of bloated indexes on one table

(these things tend to go together).

De-bloating indexes: CREATE INDEX CONCURRENTLY

The hard, targeted way: build an identical index:

1. `CREATE INDEX CONCURRENTLY my_dup_index ON my_table(column_1, column_2);`

2. [Make sure the index built OK!]

3. `DROP INDEX my_original_index;`

De-bloating indexes: CREATE INDEX CONCURRENTLY

How to check the index built OK:

```
\d+ my_table
```

At the bottom, you'll see the index:

```
  "my_dup_index" UNIQUE, btree (column_1, column_2) INVALID
```

If your new index is marked as INVALID, you need to DROP it and try again.

http://www.postgresql.org/docs/current/static/sql-createindex.html#SQL-CREATEINDEX-CONCURRENTLY

De-bloating indexes: CREATE INDEX CONCURRENTLY

SQL for finding INVALID indexes:

```
SELECT pg_namespace.nspname AS namespace, pg_class.relname AS bad_index
  FROM pg_class
  JOIN pg_index ON pg_class.oid = pg_index.indexrelid
  JOIN pg_namespace ON pg_class.relnamespace = pg_namespace.oid
 WHERE (pg_index.indisvalid = false OR pg_index.indisready = false)
;
```

De-bloating indexes: CREATE INDEX CONCURRENTLY

caveat: the index swap method doesn't work so well for primary key indexes.

You can make a UNIQUE index identical to a PK.

But you need an AccessExclusiveLock to make it the PK/fix FKs.

= Downtime

Better just pg_repack.

Preventing future bloat

OK, that takes care of old bloat. How to keep it from recurring?

Tune autovacuum.

There are a lot of knobs to turn.

```
select name, setting, unit, context from pg_settings where name like '%vacuum%' or
name like '%analyze%';
```

How much to vacuum

autovacuum_max_workers

= max number of concurrent autovacuum process

= number of tables that will be autovacuumed at once

default: 3

5-6 is OK

How much to vacuum

maintenance_work_mem

= the max mem allowed per autovacuum process

16MB is the default; this is *way* too low.

normally set to 1/16 of RAM

(but consider lowering if you raise autovacuum_max_workers)

How often to vacuum

How often does autovacuum vacuum a table?

Every n rows written, where

n = autovacuum_vacuum_scale_factor * [CURRENT NUMBER OF ROWS] + autovacuum_vacuum_threshold

"Current number of rows" = pg_class.reltuples = an approximation from the stats collector.

How often to vacuum

n = autovacuum_vacuum_scale_factor * [CURRENT NUMBER OF ROWS] + autovacuum_vacuum_threshold

autovacuum_scale_factor = .1 by default

so 10% of table rows

That's too high for large tables;

autovacuum happens too seldom.

How often to vacuum

n = autovacuum_vacuum_scale_factor * [CURRENT NUMBER OF ROWS] + autovacuum_vacuum_threshold

For big tables, try a per-table "storage parameter" setting:

```
ALTER TABLE my_table SET (autovacuum_vacuum_scale_factor
= 0.02, autovacuum_vacuum_threshold = 50000);
```

Here we increase autovacuum_vacuum_threshold so autovacuum doesn't overdo it.

Now time between vacuum is mostly constant

(less table-size-dependent).

How fast to vacuum

autovacuum tries not to hammer your system.

It takes many small breaks while working, each autovacuum_vacuum_cost_delay ms long.

It counts I/O "cost units" up to autovacuum_vacuum_cost_limit, then takes a break for autovacuum_cost_delay ms.

After finishing one table, it sleeps for autovacuum_naptime seconds.

So much for theory. You almost never need to hack these.

If you do, raise autovacuum_vacuum_cost_limit and lower autovacuum_vacuum_cost_delay to trade higher disk and CPU utilization for faster vacuuming.

Side note: autoanalyze

The autovacuum daemon has a cousin: the autoanalyze daemon

It updates stats periodically.

Many of these GUCS have _analyze_ versions: autovacuum_analyze_scale_factor, autovacuum_analyze_threshold, …

These are much cheaper to raise, but don't bother…

… without specific evidence that bad stats are causing bad query plans.

The story so far

So far, we've talked about bloat

What it is

Why it's bad

How to measure it

How to get rid of it

How to prevent it

XID overflow

But there's another danger.

Like bloat, it's a downside of MVCC.

It is XID overflow.

XID overflow

xmin	*xmax*	id	product_name	price
3		4	dog food	30
17		5	turtle food	15
250	500	6	cat food	20
500		6	cat food	25

Review:

xmin and xmax are hidden columns.

They track which row versions are visible to which transactions.

XID is actually a ring buffer.

XID overflow

An XID is a 32-bit int

2^31 - 1 transactions before it overflows; math here: http://www.depesz.com/2013/12/06/what-does-fix-vacuums-tests-to-see-whether-it-can-update-relfrozenxid-really-mean/

Then PostgreSQL loses track, can't tell which table rows are in the past and which are in the future.

This is XID overflow. It's a disaster.

XID overflow

XID overflow is so bad

postgres will shut down to prevent it.

And won't come up till you fix it.

Moral: don't turn autovacuum off.

Defense against XID wraparound

"autovacuum freeze" = a special (auto)vacuum

It finds rows so old that every transaction can see them.

Then it marks them as such by replacing xmin with a magic value (RelFrozenXID == 2).

This can happen opportunistically during "normal" (anti-bloat) vacuuming.

Defense against XID wraparound

Another line of defense:

```
autovacuum_freeze_max_age = 2000000 # 2 million
```

Hit it and a special autovacuum freeze starts.

If you cancel it, it will just start over again.

This might happen at a time when you have high traffic, hosing I/O.

So don't let that happen!

Measuring XID usage

How do you know you're nearing wraparound?

We wrote some queries:

git clone https://github.com/pgexperts/pgx_scripts

Measuring XID usage

```
$ git clone https://github.com/pgexperts/pgx_scripts
$ psql -f pgx_scripts/vacuum/database_xid_age.sql my_database

         datname          |  xid_age  | av_wrap_pct | shutdown_pct |  gb_size
--------------------------+-----------+-------------+--------------+-----------
 postgres                 | 193237368 |        96.6 |          8.8 |      50.8
 my_database              | 177012965 |        89.2 |          8.1 | 6520866.3
(2 rows)

psql -c 'show autovacuum_freeze_max_age'

200000000
```

So a forced freeze kicks in at 200M

And my_database has at least one table at 177M.

Which table(s)?

Measuring XID usage

```
$ git clone https://github.com/pgexperts/pgx_scripts
$ psql -f pgx_scripts/vacuum/table_xid_age.sql my_database

                relname                |  xid_age  | av_wrap_pct | shutdown_pct | mb_size
---------------------------------------+-----------+-------------+--------------+---------
 small_table                           | 177012965 |        84.9 |          7.7 |   2596.0
 large_table                           | 169849047 |        88.5 |          8.0 | 226995.5
(2 rows)
```

small_table is the one that database_xid_age.sql "noticed"

but large_table is of far greater concern, because of its size.

Fixing XID problems

Explicit VACUUM FREEZE.

Scans the whole table; replaces xmin with RelFrozenXid where appropriate.

must run on the whole table at once

(if you cancel, it has to start from Square 1).

Fixing XID problems

Flexible Freeze

git clone https://github.com/pgexperts/flexible-freeze

Run as a cron job during low-traffic hours.

As with bloat, log XID query result before, after.

Preventing XID problems

Reduce vacuum_freeze_min_age.

Rows older than this will be frozen as autovacuum does its de-bloat work

(i.e., opportunistically)

Warning: Lower setting = greater CPU utilization, more time to vacuum table

Takes some tricky estimation; see See http://www.databasesoup.com/2012/10/freezing-your-tuples-off-part-2.html

Links

These slides: http://fairpath.com/vacuum-slides.pdf

Bloat queries, XID queries, other goodies: https://github.com/pgexperts/pgx_scripts

Flexible Freeze: https://github.com/pgexperts/flexible-freeze

pg_repack: http://reorg.github.io/pg_repack/ and https://github.com/reorg/pg_repack

CREATE INDEX CONCURRENTLY caveats: http://www.postgresql.org/docs/current/static/sql-createindex.html#SQL-CREATEINDEX-CONCURRENTLY

Further reading: depesz on XID math: http://www.depesz.com/2013/12/06/what-does-fix-vacuums-tests-to-see-whether-it-can-update-relfrozenxid-really-mean/

Further reading: Gabrielle Roth slides: https://wiki.postgresql.org/images/b/b5/Groth_scale12x_autovacuum.pdf

Further reading: Josh Berkus article on bloat: http://www.databasesoup.com/2012/09/freezing-your-tuples-off-part-1.html