Transaction ID wraparound and avoiding the performance penalties from autovacuum tuple freezing

Martín Marqués

2ndQuadrant

2 de agosto de 2019

2ndQuadrant®
PostgreSQL

Contents

1. Introduction
2. Freezing
3. Autovacuum
4. Preventive vacuuming
5. Emergency shutdown
6. Conclusion

MVCC

- Tuple visibility is identified by create and delete txid
- Queries can't see uncommitted tuples
- Stricter isolation levels will only see data committed before the transaction started
- Hidden columns xmin and xmax, and tuple header
- Use modular 2^{31} arithmetic and tuple header to check visibility

Tuple identification

```
martin=> begin;
BEGIN
martin=> insert into test_id select s, clock_timestamp()
         from generate_series(1,10) s;
INSERT 0 10
martin=> select txid_current();
 txid_current
--------------
          576
```

Tuple identification

```
martin=> select xmin,xmax, * from test_id ;
 xmin | xmax | id |             ts
------+------+----+----------------------------
  576 |    0 |  1 | 2019-07-31 06:37:37.661925
  576 |    0 |  2 | 2019-07-31 06:37:37.662361
  576 |    0 |  3 | 2019-07-31 06:37:37.662384
  576 |    0 |  4 | 2019-07-31 06:37:37.662391
  576 |    0 |  5 | 2019-07-31 06:37:37.662397
  576 |    0 |  6 | 2019-07-31 06:37:37.662404
  576 |    0 |  7 | 2019-07-31 06:37:37.662409
  576 |    0 |  8 | 2019-07-31 06:37:37.662415
  576 |    0 |  9 | 2019-07-31 06:37:37.662421
  576 |    0 | 10 | 2019-07-31 06:37:37.662428
```

Deleted tuples

Session 1

```
martin=> begin;
BEGIN
martin=> select txid_current();
 txid_current
--------------
          577

martin=> update test_id set ts=now() where id>5;
UPDATE 5
```

Deleted tuples

Session 1

```
martin=> select xmin,xmax, * from test_id ;
 xmin | xmax | id |              ts
------+------+----+----------------------------
  576 |    0 |  1 | 2019-07-31 06:37:37.661925
  576 |    0 |  2 | 2019-07-31 06:37:37.662361
  576 |    0 |  3 | 2019-07-31 06:37:37.662384
  576 |    0 |  4 | 2019-07-31 06:37:37.662391
  576 |    0 |  5 | 2019-07-31 06:37:37.662397
  577 |    0 |  6 | 2019-07-31 06:39:57.201562
  577 |    0 |  7 | 2019-07-31 06:39:57.201562
  577 |    0 |  8 | 2019-07-31 06:39:57.201562
  577 |    0 |  9 | 2019-07-31 06:39:57.201562
  577 |    0 | 10 | 2019-07-31 06:39:57.201562
```

Deleted tuples

Session 2

```
martin=> select xmin, xmax, * from test_id ;
 xmin | xmax | id |             ts
------+------+----+----------------------------
  576 |    0 |  1 | 2019-07-31 06:37:37.661925
  576 |    0 |  2 | 2019-07-31 06:37:37.662361
  576 |    0 |  3 | 2019-07-31 06:37:37.662384
  576 |    0 |  4 | 2019-07-31 06:37:37.662391
  576 |    0 |  5 | 2019-07-31 06:37:37.662397
  576 |  577 |  6 | 2019-07-31 06:37:29.125214
  576 |  577 |  7 | 2019-07-31 06:37:29.125214
  576 |  577 |  8 | 2019-07-31 06:37:29.125214
  576 |  577 |  9 | 2019-07-31 06:37:29.125214
  576 |  577 | 10 | 2019-07-31 06:37:29.125214
```

Contents

1. Introduction
2. Freezing
3. Autovacuum
4. Preventive vacuuming
5. Emergency shutdown
6. Conclusion

2ndQuadrant®
PostgreSQL

Which are the cons?

- ▷ txid are 32bit integers
- ▷ Only half of those txid are visible, that is 2147483648
- ▷ Highly transactional systems exhaust available txid

2ndQuadrant®
PostgreSQL

Which are the cons?

- txid are 32bit integers
- Only half of those txid are visible, that is 2147483648
- Highly transactional systems exhaust available txid
  - 100 tx/s → 8 months
  - 1000 tx/s → 24 days

2ndQuadrant®
PostgreSQL

How does postgres wrap-around

- We Freeze!
- Freeze old tuples whose xmin is older than all running backend xmin horizon
- Frozen tuples are always visible

How does postgres wrap-around

- Vacuum and autovacuum take care of freezing tuples

How does postgres wrap-around

- Vacuum and autovacuum take care of freezing tuples
  - `autovacuum_freeze_max_age`
  - `vacuum_freeze_table_age` and `vacuum_freeze_min_age`

How does postgres wrap-around

- Vacuum and autovacuum take care of freezing tuples
  - `autovacuum_freeze_max_age`
  - `vacuum_freeze_table_age` and `vacuum_freeze_min_age`
- `VACUUM FREEZE` is vacuum with `vacuum_freeze_table_age` and `vacuum_freeze_min_age` both set to zero

2ndQuadrant® PostgreSQL

frozenxid

```
martin=> select relname, age(relfrozenxid),
                pg_size_pretty(pg_relation_size(oid)) as size
         from pg_class
         where relkind = ’r’
         order by age(relfrozenxid) desc, pg_relation_size(oid) desc
         limit 5;
    relname     | age  |  size
----------------+------+--------
 pg_proc        | 8029 | 608 kB
 pg_depend      | 8029 | 448 kB
 pg_collation   | 8029 | 432 kB
 pg_attribute   | 8029 | 392 kB
 pg_description | 8029 | 320 kB
```

frozenxid

```
martin=> vacuum freeze pg_proc;
VACUUM
martin=> select relname, age(relfrozenxid),
                pg_size_pretty(pg_relation_size(oid)) as size
         from pg_class
         where relkind = ’r’
         order by age(relfrozenxid) desc, pg_relation_size(oid) desc
         limit 5;
    relname      | age  |  size
----------------+------+--------
 pg_depend       | 8029 | 448 kB
 pg_collation    | 8029 | 432 kB
 pg_attribute    | 8029 | 392 kB
 pg_description  | 8029 | 320 kB
 pg_operator     | 8029 | 120 kB
```

frozenxid

```
martin=> select relname, age(relfrozenxid),
                pg_size_pretty(pg_relation_size(oid)) as size
         from pg_class
         where relkind = 'r'
         order by age(relfrozenxid) asc, pg_relation_size(oid) desc
         limit 5;
   relname    | age  |  size
--------------+------+--------
 pg_proc      |    0 | 608 kB
 test_id      |    0 | 224 kB
 pg_statistic |    0 | 136 kB
 pg_depend    | 8029 | 448 kB
 pg_collation | 8029 | 432 kB
```

frozenxid

```
martin=> select datname, age(datfrozenxid)
         from pg_database
         where not datistemplate;
 datname  | age
----------+-------
 postgres | 8029
 martin   | 8029
```

2ndQuadrant®
PostgreSQL

Freezing issues

- Only tuples visible to *all* backends can get frozen
- Long running transactions hold back freezing
- On PG 9.5 and older vacuum freeze would scan the whole table
  - PG 9.6 added *allfrozen* bit to the visibility map

2ndQuadrant®
PostgreSQL

Contents

1. Introduction
2. Freezing
3. Autovacuum
4. Preventive vacuuming
5. Emergency shutdown
6. Conclusion

Autovacuum work

- Normal autovacuum will cancel itself if there are lock conflicts
- If `age(relfrozenxid) > autovacuum_freeze_max_age` autovacuum will run a **vacuum to prevent wraparound**
- An autovacuum to prevent wraparound will lock conflicting backends for as long as it's running
- Autovacuum to prevent wraparound will start at any time, very likely in your business peak hours
- Administrators increase `autovacuum_freeze_max_age` so autovacuum doesn't run these *annoying* vacuums. This just delays the inevitable

Autovacuum work

- On postgres 8.1 autovacuum didn’t have freeze capabilities.

2ndQuadrant PostgreSQL

Autovacuum work

- On postgres 8.1 autovacuum didn’t have freeze capabilities. PLEASE UPGRADE!!!

Make Autovacuum Great (Again?)

- Default autovacuum parameters lax on aggressiveness
- Autovacuum needs to scan indexes as well

Make Autovacuum Great (Again?)

- Default autovacuum parameters lax on aggressiveness
- Autovacuum needs to scan indexes as well
- Big tables exacerbate the items above
- Too many indexes on a table will slow down autovacuum

Good news!

```
commit cbccac371c79d96c44fcd8c9cbb5ff4dedaaa522
Author: Tom Lane <tgl@sss.pgh.pa.us>
Date:   Sun Mar 10 15:16:21 2019 -0400

    Reduce the default value of autovacuum_vacuum_cost_delay to 2ms.

    This is a better way to implement the desired change of increasing
    autovacuum's default resource consumption.

    Discussion: https://postgr.es/m/28720.1552101086@sss.pgh.pa.us
```

Even better good news!

```
commit a96c41feec6b6616eb9d5baee9a9e08c20533c38
Author: Robert Haas <rhaas@postgresql.org>
Date:   Thu Apr 4 14:58:53 2019 -0400

    Allow VACUUM to be run with index cleanup disabled.

    This commit adds a new reloption, vacuum_index_cleanup, which
    controls whether index cleanup is performed for a particular
    relation by default.  It also adds a new option to the VACUUM
    command, INDEX_CLEANUP, which can be used to override the
    reloption.  If neither the reloption nor the VACUUM option is
    used, the default is true, as before.

    Masahiko Sawada, reviewed and tested by Nathan Bossart, Alvaro
    Herrera, Kyotaro Horiguchi, Darafei Praliaskouski, and me.
    The wording of the documentation is mostly due to me.
```

2ndQuadrant®
PostgreSQL

What can we do in the mean time?

- Make autovacuum more aggressive
- Partition large tables
- Remove unnecessary indexes
- Don’t turn autovacuum off!!!

Contents

1. Introduction
2. Freezing
3. Autovacuum
4. Preventive vacuuming
5. Emergency shutdown
6. Conclusion

What can we do in the mean time?

- Apply all the changes from the previous section on autovacuum
- During lower load hours run preventive vacumming:
    - Query `pg_class` looking for tables with old `relfrozenxid`
    - `VACUUM` those tables with `vacuum_freeze_table_age` set to zero
    - Analyze the possiblity of using more aggressive *cost* settings while vacumming
- This prevents autovacuum from launching `VACUUM` to prevent wraparound during busier hours

Query for tables close to vacuum to prevent wraparound

```
SELECT c.oid::regclass as table,
       current_setting(’autovacuum_freeze_max_age’)::INT8 -
             age(c.relfrozenxid) as xid_left,
       pg_relation_size(c.oid) as relsize
FROM (pg_class c JOIN pg_namespace n ON (c.relnamespace=n.oid))
WHERE c.relkind = ’r’ and
      age(c.relfrozenxid)::INT8 >
          (current_setting(’autovacuum_freeze_max_age’)::INT8 * 0.9)
ORDER BY 2 ASC
```

This query will only gather tables from a specific database so you will need to repeat the process for every database

Contents

1. Introduction
2. Freezing
3. Autovacuum
4. Preventive vacuuming
5. Emergency shutdown
6. Conclusion

Once upon a time...

Once upon a time we consumed 2 billion txid's before autovacuum could freeze old tuples

- Server will send **WARNING** messages to the logs
- Once you reach wraparound system will effectively reject request for new txid
- Message will indicate shutting down, starting in single mode and running a database wide `VACUUM`

You are very close to wraparound territory

```
WARNING: database "martin" must be vacuumed within 123456789
transactions
HINT: To avoid a database shutdown, execute a database-wide
VACUUM in that database.
You might also need to commit or roll back old prepared
transactions, or drop stale replication slots.
```

You arrived to wraparound territory

```
ERROR: database is not accepting commands to avoid wraparound data
loss in database "martin"
HINT: Stop the postmaster and vacuum that database in single-user
mode.
You might also need to commit or roll back old prepared
transactions, or drop stale replication slots.
```

How to get out of wraparound territory

```
/usr/pgsql-11/bin/postgres --single martin -D 11/data

PostgreSQL stand-alone backend 11.4
backend> VACUUM
backend>
```

Contents

1. Introduction
2. Freezing
3. Autovacuum
4. Preventive vacuuming
5. Emergency shutdown
6. Conclusion

Conclusion

- Before postgres 8.2 you had to run manual `VACUUM` to freeze
- With 9.6 `VACUUM` can skip pages which are *allfrozen*
- Smaller tables are quicker to vacuum: Partition very big tables
- Postgres 12 adds feature to `INDEX_CLEANUP` when vacuuming
- `zheap` storage will likely eliminate all these problems when available in the future

2ndQuadrant® PostgreSQL

Conclusion

- Before postgres 8.2 you had to run manual `VACUUM` to freeze
- With 9.6 `VACUUM` can skip pages which are *allfrozen*
- Smaller tables are quicker to vacuum: Partition very big tables
- Postgres 12 adds feature to `INDEX_CLEANUP` when vacuuming
- `zheap` storage will likely eliminate all these problems when available in the future

UPGRADE!!!

Conclusion

- Configure `autovacuum` with more aggressive *cost* values
- Run preventive vacuuming during lower load
- Monitor age of `relfrozenxid`
- Never set `autovacuum_freeze_max_age` to a value larger than 1 billion

Questions

?