(auto)VACUUM and You.

Gabrielle Roth
EnterpriseDB
@gorthx
The Plan.

- Part I: Intro
- Part II: VACUUM
- Part II: Autovacuum
- Part III: Adjusting autovacuum parameters
Part I: Intro.

shared by creative commons license
http://creativecommons.org/licenses/by/2.0/deed.en
http://www.flickr.com/photos/23am/3816472099/
My first VACUUM.

- Data "warehouse"
- Added a bunch of rows daily
- Deleted a bunch of rows daily
- Ooooh, reports!
Uh, why are my queries so slow?

- Did I write some dumb SQL?
- No, I needed VACUUM and ANALYZE
Adding a bunch of rows

- The planner needs fresh statistics to work with.
- Adding "a bunch" of rows can change the distribution of your data.
- ...causing a sub-optimal plan.
- ANALYZE fixes this.
Deleting a bunch of rows

- They're not gone, *you just can't see them.*
- They take up space unnecessarily.
- Indexes point to all versions of a row.
- VACUUM fixes this.
- UPDATEs, too
- ”Why can't they just call it 'garbage collection' like everybody else does?”
A little MVCC.

- transaction isolation
- allows multiple people to work in the db without @$#$%ing things up
- accomplished via xids
  - wraparound is VERY BAD
- data changes result in dead/obsolete rows
  - which hang around, causing problems
  - ...until you VACUUM.
Part II: VACUUM

shared by creative commons license
http://creativecommons.org/licenses/by/2.0/deed.en
http://www.flickr.com/photos/bullcitydogs/8431397458/
### Table Stats: `pg_stat_user_tables`

```
pgbench=# SELECT relname, n_tup_ins, n_tup_upd, n_tup_del, n_live_tup, n_dead_tup, last_vacuum, last_analyze
    FROM pg_stat_user_tables
WHERE relname = 'pgbench_accounts';

<table>
<thead>
<tr>
<th>relname</th>
<th>n_tup_ins</th>
<th>n_tup_upd</th>
<th>n_tup_del</th>
<th>n_live_tup</th>
<th>n_dead_tup</th>
<th>last_vacuum</th>
<th>last_analyze</th>
</tr>
</thead>
<tbody>
<tr>
<td>pgbench_accounts</td>
<td>100000</td>
<td>73254</td>
<td>0</td>
<td>100002</td>
<td>4710</td>
<td>2014-02-17 20:06:29.900437-08</td>
<td></td>
</tr>
</tbody>
</table>
```
\begin{itemize}
\item n\_tup\_* = incrementing counters
\item n\_live\_tup = this is a guess :)
\item n\_dead\_tup = reset by a vacuum.
\item last\_* fields = last manual/auto vac/analyze
\item combine with \textbackslash watch (9.3) for additional fun
\end{itemize}
table stats: pgstattuple

- contrib module
  - 9.3: CREATE EXTENSION pgstattuple;
- one-stop shopping!

```
pqbench=# SELECT tuple_count, tuple_percent, dead_tuple_count, dead_tuple_percent FROM pgstattuple('pgbench_accounts');
-[ RECORD 1 ]----------------------
 tuple_count |  100000
 tuple_percent |  91.06
 dead_tuple_count |   1592
 dead_tuple_percent |  1.45
```
planner stats: pg_class

<table>
<thead>
<tr>
<th>relname</th>
<th>pgbench_accounts</th>
</tr>
</thead>
<tbody>
<tr>
<td>reltuples</td>
<td>100002</td>
</tr>
</tbody>
</table>

```
pgbench=# SELECT relname, reltuples
       FROM pg_class
      WHERE relname = 'pgbench_accounts';
-[ RECORD 1 ]---------------------
 relname | pgbench_accounts
 reltuples | 100002
```
more planner stats: pg_stats

```
pgebench=# SELECT tablename, attname,
    most_common_vals
FROM pg_stats
WHERE tablename = 'pgbench_tellers';

    tablename | attname | most_common_vals
-----------------+---------+------------------
pgebench_tellers | tid     |                  
pgebench_tellers | tbalance | {-20716,-5820}  
pgebench_tellers | filler  |                  
pgebench_tellers | bid     | {1,2,3,4,5,...98,99,100}
```
VACUUM (the manual kind)

- VACUUM
- VACUUM FULL
- VACUUM FREEZE
- VACUUM ANALYZE

must be table owner or superuser
VACUUM

- removes dead rows
  - cleans up your indexes
- updates your xids
- (hint bits)
- SHARE UPDATE EXCLUSIVE lock
VACUUM FULL

- frees up actual disk space
- ACCESS EXCLUSIVE lock
  - ...and it's rewriting the table on disk, so you need double the space.
- don't bother if the table's just going to refill.
VACUUM FREEZE

- sets a special xid value: relFrozenXid
  - prevent xid wraparound
- ACCESS EXCLUSIVE lock
- recommended after very large loads to tables that will see a lot of OLTP
VACUUM ANALYZE

- updates the planner statistics
- SHARE UPDATE EXCLUSIVE
- ANALYZE is actually its own separate thing you can run by itself!
pgbench=# vacuum verbose pgbench_branches;
INFO: vacuuming "public.pgbench_branches"
INFO: index "pgbench_branches_pkey" now contains 1 row
versions in 2 pages
DETAIL: 0 index row versions were removed.
0 index pages have been deleted, 0 are currently reusable.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO: "pgbench_branches": found 166 removable, 1
nonremovable row versions in 1 out of 1 pages
DETAIL: 0 dead row versions cannot be removed yet.
There were 203 unused item pointers.
0 pages are entirely empty.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
pgbench=# analyze verbose;
INFO: analyzing "public.pgbench_branches"
INFO: "pgbench_branches": scanned 1 of 1 pages, containing 1 live rows and 166 dead rows; 1 rows in sample, 1 estimated total rows
Part III: autovacuum

finally I can relax!

shared by creative commons license
http://creativecommons.org/licenses/by/2.0/deed.en
http://www.flickr.com/photos/barbostick/3581760713/
caption added
How this is supposed to work.

- **pg_stat_user_tables**
  - `n_live_tup`
  - `n_dead_tup`
  - `n_tup_ins`
  - `n_tup_upd`
  - `n_tup_del`

- **pg_class.reltuples**
  - **UPDATE** w/estimate
  - **RUNS WHEN**
    - `n_dead_tup > analyze_threshold + analyze_scale * pg_class.reltuples`

- **VACUUM**
  - **RUNS WHEN**
    - `n_dead_tup > vacuum_threshold + vacuum_scale * pg_class.reltuples`

- **autovacuum daemon**
  - sets `pg_stat_user_tables.n_dead_tup = 0`
My table isn't being vacuumed!
(dramatization)

```sql
SELECT relname, n_live_tup, n_dead_tup,
last_autovacuum, last_autoanalyze
FROM pg_stat_user_tables
WHERE relname = 'pgbench_accounts';
```

<table>
<thead>
<tr>
<th>relname</th>
<th>pgbench_accounts</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_live_tup</td>
<td>1000000</td>
</tr>
<tr>
<td>n_dead_tup</td>
<td>9499</td>
</tr>
<tr>
<td>last_autovacuum</td>
<td></td>
</tr>
<tr>
<td>last_autoanalyze</td>
<td></td>
</tr>
</tbody>
</table>
Is autovacuum even running?

- `ps -ef | grep vacuum`

- `postgres 1101 972 0 06:37 ? 00:00:33 postgres: autovacuum launcher process`

- **in postgresql.conf:**

  ```
  autovacuum = on #default
  track_counts = true #default
  ```

- **psql shell:**

  ```
  pgbench=# SELECT name, setting || unit AS setting FROM pg_settings WHERE category = 'Autovacuum';
  pgbench=# SHOW autovacuum;
  ```
autovacuum: do the math.

- in postgresql.conf:

  #autovacuum_vacuum_threshold = 50
  # min number of row updates before vacuum
  #autovacuum_vacuum_scale_factor = 0.2
  # fraction of table size before vacuum
autovacuum: do the math.

- in postgresql.conf:

  `#autovacuum_vacuum_threshold = 50`
  `# min number of row updates before vacuum`
  `#autovacuum_vacuum_scale_factor = 0.2`
  `# fraction of table size before vacuum`

- vacuum threshold =
  `autovacuum_vacuum_threshold + autovacuum_vacuum_scale_factor * pgclass.reltuples`

- 1000 row table = 50 + (0.2 * 1000) = 250
- 1,000,000 row table = 50 + (0.2 * 1000000) = 200,050
- 9500 dead tuples is not even close
in postgresql.conf:

```sql
#autovacuum_analyze_threshold = 50
    # min number of row updates before analyze
#autovacuum_analyze_scale_factor = 0.1
    # fraction of table size before analyze
```

- `analyze threshold =
  autovacuum_analyze_threshold +
  autovacuum_analyze_scale_factor * pgclass.reltuples`

- 1000 row table = 50 + (0.1 * 1000) = 150
- 1,000,000 row table = 50 + (0.1 * 1000000) = 100,050
You still need to manually:
  - VACUUM [FREEZE] ANALYZE after a large load.
  - ANALYZE temp tables.

THIS JUST IN:
  - apply the latest update! Has a fix for potential data corruption if you have frequent xid wrap.
  - Also some new tuning params I haven't tried yet :)
Part IV: Adjusting autovacuum parameters

shared by creative commons license
http://creativecommons.org/licenses/by/2.0/deed.en
http://www.flickr.com/photos/colley/3878176777/
GUCs

- 6 of 'em:
  - autovacuum_vacuum_threshold
  - autovacuum_vacuum_scale_factor
  - autovacuum_max_workers
  - autovacuum_nap_time
  - autovacuum_cost_limit
  - autovacuum_cost_delay
- + autovacuum_analyze_threshold and scale_factor
Before we begin...

- have a backup!
- have metrics!
- change ONE thing at a time: measure, change, remeasure, repeat.
- make use of 'include' in postgresql.conf
My picks

- CPU, mem, I/O, connections, locks, long queries, vac jobs, ...

- from the Pg activity log:
  - log_line_prefix in a pgbadger-compatible format
    - \%t [\%p]: [\%l-1]
  - log_min_duration_statement = [YMMV]
  - log_autovacuum_min_duration = [YMMV]
  - log_lock_waits = on

- collect table stats JFK

- \watch!
sample log message from autovacuum

- log_autovacuum_min_duration = 0

- %LOG: automatic vacuum of table “ttrss.public.ttrss_feedbrowser_cache”: index scans: 1
  pages: 0 removed, 11 remain
  tuples: 303 removed, 303 remain
  buffer usage: 82 hits, 0 misses, 10 dirtied
  avg read rate: 0.000 MB/s, avg write rate: 3.585 MB/s
  system usage: CPU 0.00s/0.00u sec elapsed 0.02 sec

- %LOG: automatic analyze of table “ttrss.public.ttrss_feedbrowser_cache” system usage: CPU 0.00s/0.00u sec elapsed 0.03 sec
GUCs: when will vac happen

#autovacuum_vacuum_threshold = 50
    # min number of row updates before vacuum

#autovacuum_vacuum_scale_factor = 0.2
    # fraction of table size before vacuum

<table>
<thead>
<tr>
<th>live_tup</th>
<th>50 th, 0.2 sf</th>
<th>5k th, 0.2 sf</th>
<th>50 th, 0.02 sf</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(default)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,000</td>
<td>250</td>
<td>5,200</td>
<td>70</td>
</tr>
<tr>
<td>10,000</td>
<td>2,050</td>
<td>7,000</td>
<td>250</td>
</tr>
<tr>
<td>100,000</td>
<td>20,050</td>
<td>25,000</td>
<td>2,050</td>
</tr>
<tr>
<td>1,000,000</td>
<td>200,050</td>
<td>205,000</td>
<td>20,050</td>
</tr>
<tr>
<td>10,000,000</td>
<td>2,000,050</td>
<td>2,005,000</td>
<td>200,050</td>
</tr>
<tr>
<td>100,000,000</td>
<td>20,000,050</td>
<td>20,005,000</td>
<td>2,000,050</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>200,000,050</td>
<td>200,005,000</td>
<td>20,000,050</td>
</tr>
</tbody>
</table>
GUCs: how many tables can be vacced at ~ the same time

- `#autovacuum_max_workers = 3`
  
  # max number of autovacuum subprocesses

- **requires a restart**

- `#autovacuum_naptime = 1min`
  
  # time between autovacuum runs

- These are per-cluster.

- Be mindful of `maintenance_work_mem`:
  
  \[ \text{av\_max\_workers} \times \text{maint\_work\_mem} < \text{memory} \]
GUCs: how fast can I make this thing go

- `#autovacuum_vacuum_cost_limit = -1`
  
  # default vacuum cost limit for autovacuum,
  
  # -1 means use vacuum_cost_limit (default: 200 "credits")

- `#autovacuum_vacuum_cost_delay = 20ms`
  
  # default vacuum cost delay for autovacuum, in milliseconds;
  
  # -1 means use vacuum_cost_delay (default: 0ms)

- speed this up by:
  
  - increasing cost_limit to some value in the hundreds, or
    (and?)

  - setting cost_delay to 0
An unfriendly reminder.

- All 6 of these GUCs that we just looked at* interact together.
- If your table changes size dramatically, you will likely need to readjust these settings.
- You still need to manually:
  - VACUUM [FREEZE] ANALYZE after a large load.
  - ANALYZE temp tables.
- ISN'T THIS FUN.

*and some others that outside the scope of this talk
per-table adjustment

- can't do this with naptime or max_workers
- CREATE TABLE mytable (blahblah) WITH (autovacuum_vacuum_threshold = 2000);
- ALTER TABLE mytable SET (autovacuum_vacuum_threshold = 2000);
- view with \d+:
  Options: autovacuum_vacuum_threshold=2000
- -- reset to value from postgresql.conf!
  ALTER TABLE mytable RESET (autovacuum_vacuum_threshold);
Epilogue.
pgbench=# SELECT relname, n_tup_ins AS ins, n_tup_upd AS upd, n_tup_del AS del, n_live_tup AS live, n_dead_tup AS dead, last_autovacuum AS l_aa, last_autoanalyze AS l_av
FROM pg_stat_user_tables;

<table>
<thead>
<tr>
<th>relname</th>
<th>ins</th>
<th>upd</th>
<th>del</th>
<th>live</th>
<th>dead</th>
<th>l_aa</th>
<th>l_av</th>
</tr>
</thead>
<tbody>
<tr>
<td>pgbench_branches</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pgbench_tellers</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pgbench_history</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pgbench_accounts</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(4 rows)
streaming rep + vacuum

- table stats don't get replicated
- planner stats do, but we can't see those
- You can't run a VACUUM on the standby:
  ```sql
  postgres=# vacuum mytable;
  ERROR: cannot execute VACUUM during recovery
  ```
- vacuum jobs are WAL logged
Wishlist

- An easier way to see what's being vacuumed & the progress thereof
  - combo of ps & looking at the locks table hoping to catch something going by
- A way to view the vacuum queue & see WHO'S NEXT.
Help! (and further reading)

- Pg docs + -admin + Pg wiki
- xid wraparound:
  https://devcenter.heroku.com/articles/postgresql-concurrency
- Josh B's "Freezing Your Tuples Off" series
- https://wiki.postgresql.org/wiki/VacuumHeadaches
Thank you!