Avoid Angering the PostgreSQL Elder Gods

Presented by
Keith Fiske / @keithf4

Senior Database Engineer @ Crunchy Data
(pg_partman, pgMonitor, pg_extractor)

http://slides.keithf4.com/pg_elder_gods.pdf

CRUNCHY DATA SOLUTIONS, INC

- Industry leader in providing enterprise PostgreSQL support and open source solutions
- 100% Open Source PostgreSQL
  - No lock-in
- Crunchy Postgres
  - High Availability
  - Monitoring
  - Hardened
  - Common Criteria EAL 2+
- Crunchy Postgres for Kubernetes
  - Operator
- Crunchy Bridge
  - Fully-managed Postgres on your choice of cloud (AWS, Azure, GCP)

Talk Roadmap

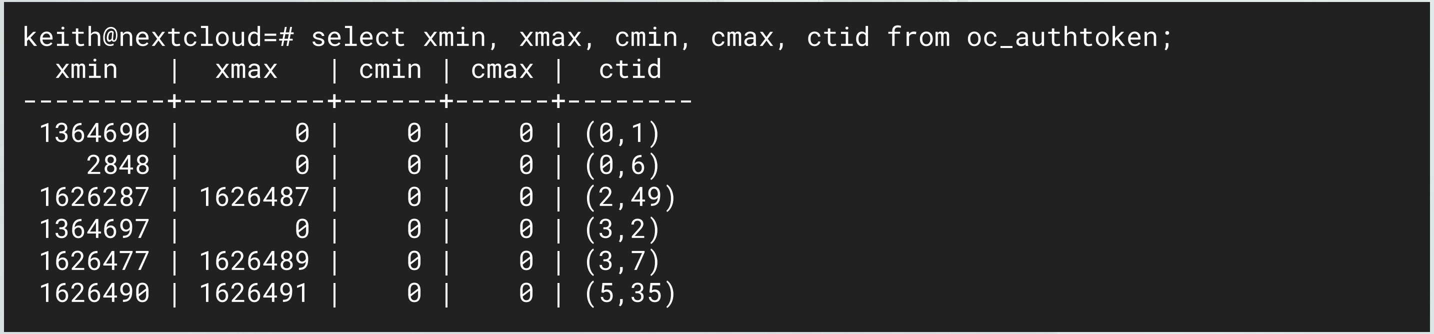
- What are Transaction IDs?
- The First God
  - Transaction ID Exhaustion
- The Second God
  - Bloat

crunchy data

Transaction IDs (XID)

- (Almost) always increasing 32-bit unsigned integer value; therefore maximum value of approximately 4 billion.
- MultiVersion Concurrency Control (MVCC) depends on being able to compare XID numbers
- In general, a tuple with an insertion XID greater than the current XID is "in the future" and should not be visible to the current transaction
- A tuple with an insertion XID less than the current is "in the past" and should be visible
- A tuple with a deletion xid is the opposite

Finding XIDs - Hidden Columns



```
keith@nextcloud=# select xmin, xmax, cmin, cmax, ctid from oc_authtoken;
  xmin   |  xmax   | cmin | cmax |  ctid
---------+---------+------+------+--------
 1364690 |       0 |    0 |    0 | (0,1)
    2848 |       0 |    0 |    0 | (0,6)
 1626287 | 1626487 |    0 |    0 | (2,49)
 1364697 |       0 |    0 |    0 | (3,2)
 1626477 | 1626489 |    0 |    0 | (3,7)
 1626490 | 1626491 |    0 |    0 | (5,35)
```

- xmin - insertion xid
- xmax - deletion xid
- cmin, cmax - transaction level xids
- ctid - physical location of the row version within its table
    - Can change with update or vacuum full, so do not use for long term identification
    - Useful for removing duplicate rows

Transaction IDs (XID)

- Transaction Isolation Level can also affect visibility of committed transactions
  - https://www.postgresql.org/docs/current/transaction-iso.html
- Normal XIDs are compared using modulo-2^{32} arithmetic. This means that for every normal XID, there are two billion XIDs that are “older” and two billion that are “newer”;
- One of the more important PG Administration doc pages to read and understand
  - https://www.postgresql.org/docs/current/routine-vacuuming.html

crunchydata

Freezing Tuples

- One of vacuum's jobs: mark tuples so they are visible to all future transactions.
  - Also updates Visibility Map.
- Sets flag bit in tuple that row is "frozen" so that it is always in the past
  - Prior to 9.4, would actually set xmin to FrozenTransactionId value
- Cannot freeze rows being used by active transactions
  - Monitoring for long running transactions is an easy step in avoiding exhaustion
  - Fewer long running transactions leads to more efficient vacuuming
- Modern PG versions can check page level frozen flag in Visibility Map
  - Tremendously speeds up vacuum on large tables with fewer changes
- So what happens after billions of transactions with no freezing?

crunchydata

XID Exhaustion

- Normal XID space is circular with no endpoint
- Wraparound is fine, the real problem is XID exhaustion
  - Wraparound happens normally when the current XID reaches max uint
  - But it's not fine when there's no new XIDs for comparison
- Suddenly transactions that were in the past appear to be in the future
  - Valid tuples no longer visible; they're there but no one can see them
- Database automatically shuts down
  - Must be started in single user mode
  - Perform a vacuum on entire database or targeted tables to freeze rows
- To avoid this, it is necessary to vacuum every table in every database at least once every two billion transactions
  - Autovacuum can be disabled, but vacuuming SHOULD be done manually on active databases.

Transaction Age

- datfrozenxid is a lower bound on the unfrozen XIDs appearing in that database; ie the oldest unvacuumed tuple
- age() applied to XID computes the given value compared to the current normal XID
- Watch for maximum age approaching 2 billion

```
SELECT datname, datfrozenxid, age(datfrozenxid), txid_current() FROM pg_database;

  datname   | datfrozenxid |   age   | txid_current
------------+--------------+---------+--------------
 keith      |          720 | 1364151 |      1364871
 nextcloud  |          716 | 1364155 |      1364871
 postgres   |          716 | 1364155 |      1364871
 template0  |          716 | 1364155 |      1364871
 template1  |          716 | 1364155 |      1364871
```

crunchydata

Emergency Vacuuming

- When a table's oldest tuple age reaches autovacuum_freeze_max_age, PostgreSQL will run an "emergency" autovacuum

```
autovacuum: VACUUM public.orders (to prevent wraparound)
```

- Default value is 200 million; well below the max value of 2 billion
- This vacuum is more aggressive and runs even with autovacuum disabled
    - Normal vacuum skips pages that have no dead tuples even if there are unfrozen XIDs
    - Aggressive freezes all eligible unfrozen XIDs
- vacuum_failsafe_age (PG14+)
    - Ignores vacuum cost delay (discussed later) & index vacuuming
    - 1.6 billion
- Do not rely on this if autovac is disabled. Often triggers many tables needing vacuuming at the same time
- Other less common situations can cause this as well
    - See [Routing Vacuuming]()

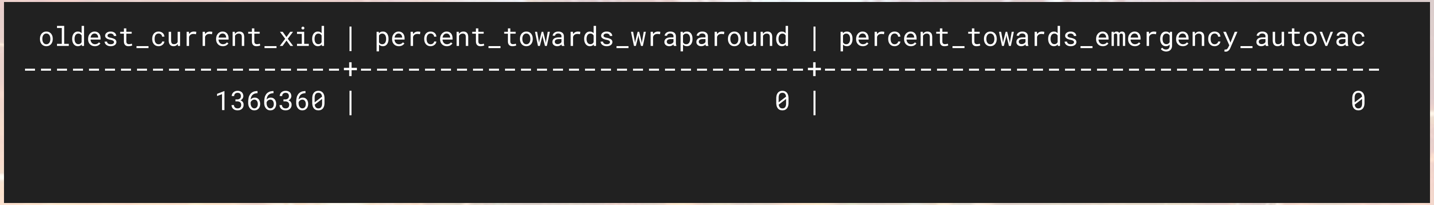
Monitoring for Exhaustion

```
WITH max_age AS (
  SELECT 2000000000 AS max_old_xid
    , setting AS autovacuum_freeze_max_age
  FROM pg_catalog.pg_settings
  WHERE name = 'autovacuum_freeze_max_age')

, per_database_stats AS (
  SELECT datname
    , m.max_old_xid::INT
    , m.autovacuum_freeze_max_age::INT
    , age(d.datfrozenxid) AS oldest_current_xid
  FROM pg_catalog.pg_database d
  JOIN max_age m ON (TRUE)
  WHERE d.datallowconn)

SELECT MAX(oldest_current_xid) AS oldest_current_xid
   , MAX(ROUND(100*(oldest_current_xid/max_old_xid::FLOAT))) AS
      percent_towards_wraparound
   , MAX(ROUND(100*(oldest_current_xid/autovacuum_freeze_max_age::FLOAT))) AS
      percent_towards_emergency_autovac
FROM per_database_stats;
```

crunchy data

Monitoring for Exhaustion

- Simplified query result for easy monitoring



```
 oldest_current_xid | percent_towards_wraparound | percent_towards_emergency_autovac
--------------------+----------------------------+-----------------------------------
            1366360 |                          0 |                                 0
```

- Emergency threshold - warn 110%, critical 125%
  - Reaching 100% isn't a problem unless many large tables all do it at once
  - Exceeding emergency for extended periods of time means that autovacuum is not keeping up
  - Resolving this alert ALWAYS prevents wraparound/exhaustion
- Wraparound threshold - warn 60%, critical 75%

crunchy data

Vacuum Multitasking - Row Cleanup

- Delete only marks tuples "unavailable" or "dead"
  - Sets xmax to determine tuple visibility
- Update internally is Delete/Insert
- Vacuum marks "dead" tuples as available space
  - bloat = dead tuples + available space

  -
    ```
    select n_dead_tup from pg_stat_all_tables;
    ```
- Excessive bloat can cause heavier IO
  - Smallest data size that PG can return is a page (default 8K)
  - Data spread thinly across pages means more pages need to be fetched
- Not all bloat is bad
  - Re-using available space saves on IO resource usage
- Find the balance!

Bloat is Rising

Monitoring Bloat - Old Way

- Fancy queries (https://wiki.postgresql.org/wiki/Show_database_bloat)
- Instant result, based on statistics. Mostly good, but can be wildly inaccurate.

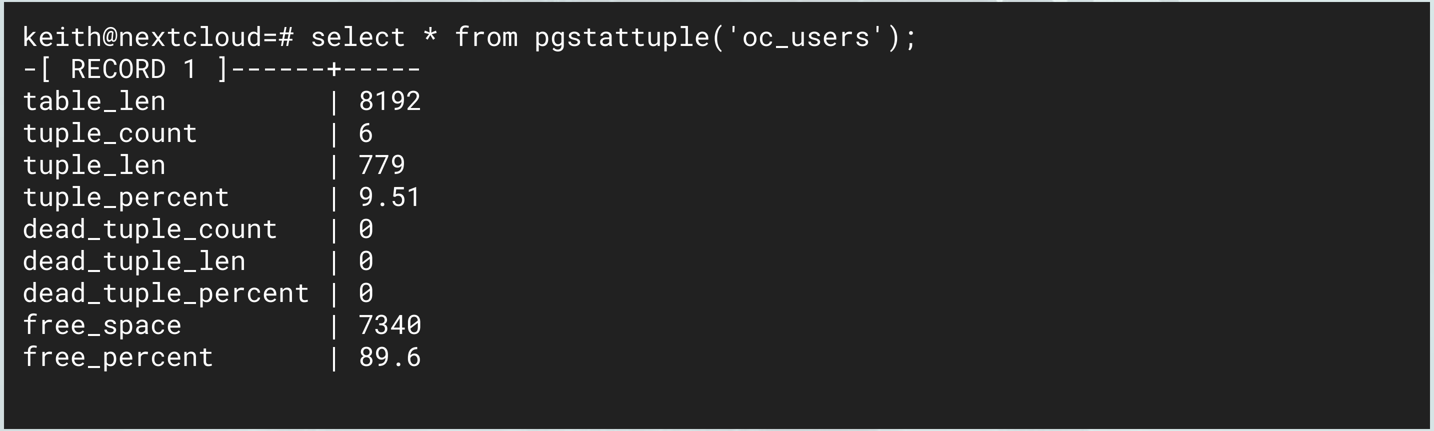
```
SELECT
  current_database(), schemaname, tablename, /*reltuples::bigint, relpages::bigint, otta,*/
  ROUND((CASE WHEN otta=0 THEN 0.0 ELSE sml.relpages::float/otta END)::numeric,1) AS tbloat,
  CASE WHEN relpages < otta THEN 0 ELSE bs*(sml.relpages-otta)::BIGINT END AS wastedbytes,
  iname, /*ituples::bigint, ipages::bigint, iotta,*/
  ROUND((CASE WHEN iotta=0 OR ipages=0 THEN 0.0 ELSE ipages::float/iotta END)::numeric,1) AS ibloat,
  CASE WHEN ipages < iotta THEN 0 ELSE bs*(ipages-iotta) END AS wastedibytes
FROM (
  SELECT
    schemaname, tablename, cc.reltuples, cc.relpages, bs,
    CEIL((cc.reltuples*((datahdr+ma-
      (CASE WHEN datahdr%ma=0 THEN ma ELSE datahdr%ma END))+nullhdr2+4))/(bs-20::float)) AS otta,
    COALESCE(c2.relname,'?') AS iname, COALESCE(c2.reltuples,0) AS ituples, COALESCE(c2.relpages,0) AS ipages,
    COALESCE(CEIL((c2.reltuples*(datahdr-12))/(bs-20::float)),0) AS iotta -- very rough approximation, assumes all cols
  FROM (
    SELECT
[...]
```

crunchy data

Monitoring Bloat - Better Ways

- pgstattuple
  - https://www.postgresql.org/docs/current/pgstattuple.html
- Statistics summary for tables and indexes
- Free space and dead tuple stats for tables and B-tree indexes
- Stats for other index types available, but nothing bloat related
- Full-table scan to gather 100% accurate stats
  - Large tables/databases can take a while to scan
  - Approximate function reports accurate dead and estimated live and free space
- Must target individual table OR index for each call
  - Does not include TOAST in table scan

pgstattuple



```
keith@nextcloud=# select * from pgstattuple('oc_users');
-[ RECORD 1 ]------+-----
table_len          | 8192
tuple_count        | 6
tuple_len          | 779
tuple_percent      | 9.51
dead_tuple_count   | 0
dead_tuple_len     | 0
dead_tuple_percent | 0
free_space         | 7340
free_percent       | 89.6
```

Freespace Map

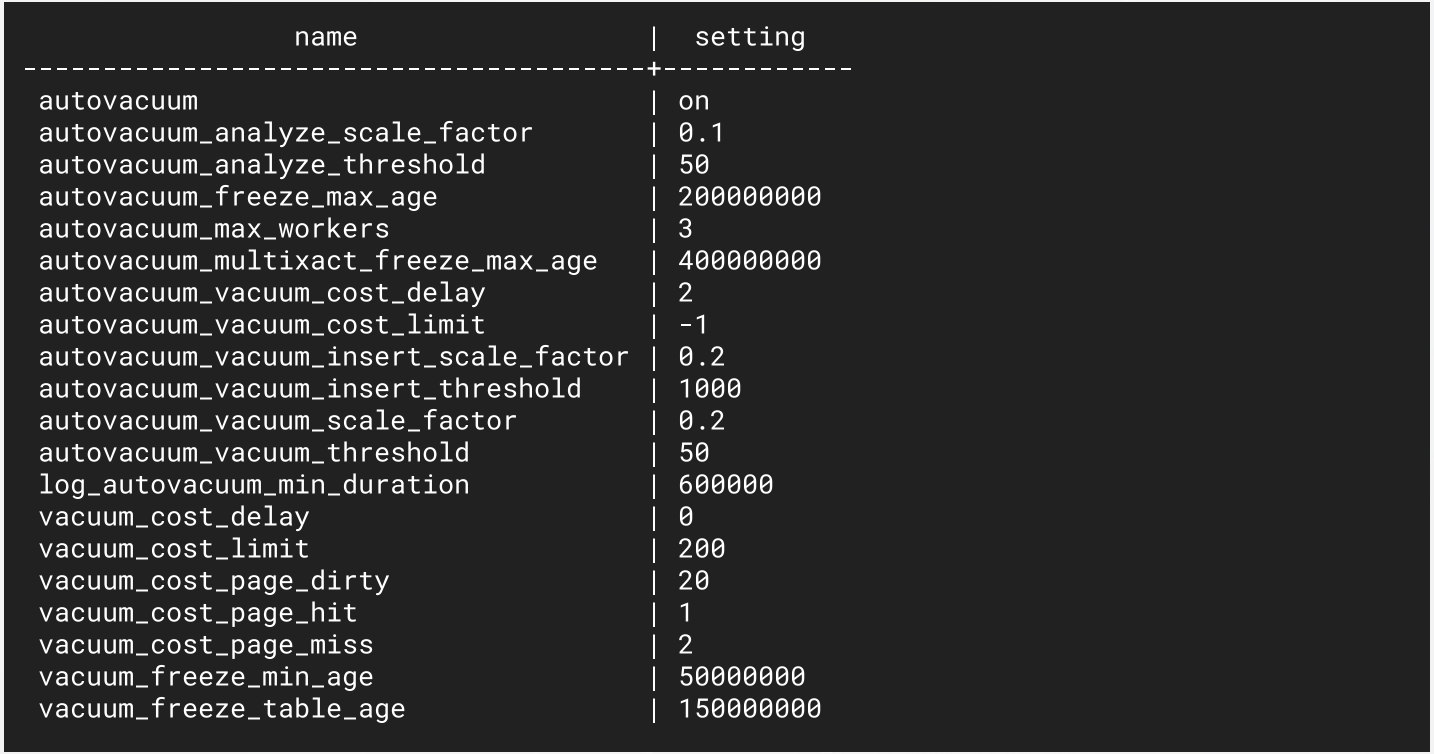
- pg_freespacemap
  - https://www.postgresql.org/docs/current/pgfreespacemap.html
- Functions to show the value recorded in the free space map for a given page, or for all pages in the relation
- Shows approximate free space on each page, one row per page
- Not kept fully up-to-date in real time. Another job for Vacuum!

```
keith@nextcloud=# select * from pg_freespace('oc_jobs');
 blkno | avail
-------+-------
     0 |  5248
     1 |  5152
     2 |  7680
```

Monitoring Bloat - Easy Way

- pg_bloat_check
  - https://github.com/keithf4/pg_bloat_check
- Reports table and B-tree bloat using pgstattuple
- For each table, scans all indexes and TOAST
  - Accounts for fillfactor
- Can scan entire database or target tables
- Filters for minimum object size, wasted space size/percent
  - Fine-grained exclude filter based on config file
- Stores results in table
  - Allows real-time monitoring without having to wait for full table scans

Vacuum Tuning



```
                 name                  |  setting
---------------------------------------+-----------
 autovacuum                            | on
 autovacuum_analyze_scale_factor       | 0.1
 autovacuum_analyze_threshold          | 50
 autovacuum_freeze_max_age             | 200000000
 autovacuum_max_workers                | 3
 autovacuum_multixact_freeze_max_age   | 400000000
 autovacuum_vacuum_cost_delay          | 2
 autovacuum_vacuum_cost_limit          | -1
 autovacuum_vacuum_insert_scale_factor | 0.2
 autovacuum_vacuum_insert_threshold    | 1000
 autovacuum_vacuum_scale_factor        | 0.2
 autovacuum_vacuum_threshold           | 50
 log_autovacuum_min_duration           | 600000
 vacuum_cost_delay                     | 0
 vacuum_cost_limit                     | 200
 vacuum_cost_page_dirty                | 20
 vacuum_cost_page_hit                  | 1
 vacuum_cost_page_miss                 | 2
 vacuum_freeze_min_age                 | 50000000
 vacuum_freeze_table_age               | 150000000
```

When Does Autovacuum Run?

- autovacuum_freeze_max_age
  - Controls emergency wraparound vacuum run
  - Increase to give busy databases more time for normal autovac to run
- vacuum_freeze_table_age controls when aggressive vacuum runs (non-wraparound)
- autovacuum_vacuum_scale_factor, autovacuum_analyze_scale_factor
  - Percentage of table that has gotten updated/deleted
- autovacuum_vacuum_threshold, autovacuum_analyze_threshold
  - Number of tuples updated/deleted
- scale factor + threshold = run vacuum
- autovacuum_vacuum_insert_scale_factor, autovacuum_vacuum_insert_threshold
  - Settings added in PG13 for insert-only tables
  - Previous versions would only trigger vacuum during emergency

[creature] Release me.

Autovacuum Resource Usage

- vacuum_cost_page_dirty, vacuum_cost_page_hit, vacuum_cost_page_miss
  - Accumulates cost points while running
- vacuum_cost_limit, autovacuum_vacuum_cost_limit
  - When accumulation reaches limit …
- vacuum_cost_delay, autovacuum_vacuum_cost_delay
  - … delay for this time
  - Manual vacuum has no cost delay and is why it can run faster

[creature] Release me.

crunchy data

Per-Table Tuning

```
select * from pg_stat_all_tables where relname = 'oc_user_status';
-[ RECORD 1 ]-------+------------------------------
relid               | 20386
schemaname          | public
relname             | oc_user_status
seq_scan            | 58480
seq_tup_read        | 175440
idx_scan            | 2655
idx_tup_fetch       | 2653
n_tup_ins           | 3
n_tup_upd           | 253
n_tup_del           | 0
n_tup_hot_upd       | 2
n_live_tup          | 3
n_dead_tup          | 51
n_mod_since_analyze | 54
n_ins_since_vacuum  | 0
last_vacuum         |
last_autovacuum     | 2023-02-01 18:05:19.362647-05
last_analyze        |
last_autoanalyze    | 2023-02-01 17:41:18.713626-05
vacuum_count        | 0
autovacuum_count    | 2
analyze_count       | 0
autoanalyze_count   | 2
```

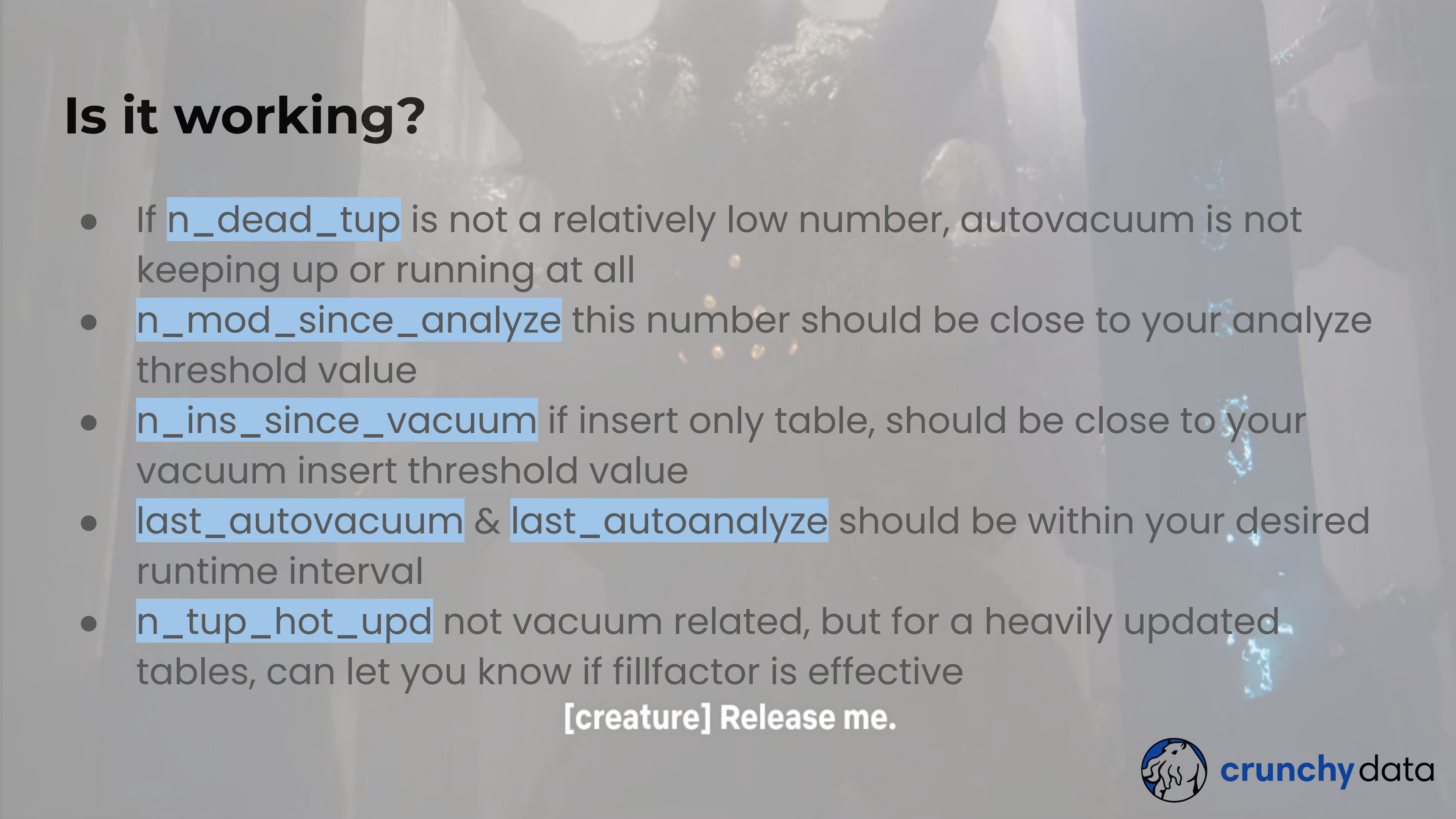
Per-Table Tuning

- Tune database level for most common case
- Tune at table level depending on how table is used
- Determine tuple change rate
- Run hourly export to CSV file (use COPY command)
- Determine hourly/daily/weekly rate of n_tup_del + n_tup_upd
  - Insert only tables can look at n_tup_ins
- Set scale factors to zero for autovacuum and analyze
  - Percentage means autovac could run less often as table gets larger
- Set threshold to values of tuple change to determine autovacuum run intervals
  - Ex. 22432 updates per day + 32432 deletes per day = 54864
  - Set vacuum threshold to 54864 * 7 to have (auto)vacuum about once a week
  - Set analyze threshold to 54864 / 2 to have analyze run 2 times per day (keep stats updated)

[creature] Release me.

crunchy data

Is it working?



- If n_dead_tup is not a relatively low number, autovacuum is not keeping up or running at all
- n_mod_since_analyze this number should be close to your analyze threshold value
- n_ins_since_vacuum if insert only table, should be close to your vacuum insert threshold value
- last_autovacuum & last_autoanalyze should be within your desired runtime interval
- n_tup_hot_upd not vacuum related, but for a heavily updated tables, can let you know if fillfactor is effective

[creature] Release me.

Keep Them Contained

- Transaction IDs are how PostgreSQL manages data visibility

- Ensure any PostgreSQL monitoring solution you use has the Exhaustion/Wraparound metric

- Exhaustion and Bloat are not going to happen right away
  - Could be years before they are a problem
  - Monitor now so they never are

crunchydata

Keep Them Contained

- More on Bloat tomorrow
  - Peter Geoghegan @ 11 in Ballroom B (this room)
  - Chelsea Dole @ 3:30 in Ballroom A
- These slides - http://slides.keithf4.com/pg_elder_gods.pdf
- PostgreSQL Home Page - postgresql.org
- Crunchy Data Solutions, Inc - crunchydata.com
- Planet PostgreSQL Community News Feed - planet.postgresql.org
- PostgreSQL Extension Network - pgxn.org
- Art Credit
  - Cthulhu Images - https://andreewallin.com/
  - Netflix: Love, Death & Robots
    - Season 3: In Vaulted Halls Entombed

crunchy data