Advanced Autovacuum Tuning and working with the new pganalyze VACUUM Advisor

We'll start in a few minutes!

pganalyze

@LukasFittl

Advanced Autovacuum Tuning and working with the new pganalyze VACUUM Advisor

@LukasFittl

What We'll Talk About Today

1. Going to the source: VACUUM stats in Postgres today
2. The concepts and structure behind VACUUM Advisor
3. Bloat estimates and tuning the dead row threshold
4. How pganalyze tracks and alerts on the xmin horizon
5. Freezing metrics and anti-wraparound vacuums
6. Understanding and tuning autovacuum performance
7. What's next? Looking ahead to upcoming features

pganalyze

Going to the source: VACUUM statistics in Postgres today

The Obvious Statistics Views

pg_stat_progress_vacuum

- How far has a currently running vacuum progressed?
- Is the vacuum using multiple index phases? (= more I/O)

pg_stat_activity

- How many backends of type “autovacuum worker” are running?

The Non-Obvious Statistics Views

Xmin horizon / removable cutoff

- pg_stat_activity.backend_xmin
- pg_replication_slots.xmin
- pg_replication_slots.catalog_xmin
- pg_prepared_xacts.transaction

=> Helps you determine if MVCC visibility rules prevent VACUUM from doing cleanup

The Non-Obvious Statistics Views

Frozen Transaction ID and Minimum Multixact ID

- pg_class.relfrozenxid
- pg_class.relminmxid

=> Tells you which table gets an anti-wraparound vacuum next

=> Use logic to interpret correctly across Transaction ID epochs

=> Helps you monitor when anti-wraparound vacuums are running but not making progress (and you're getting closer to transaction ID or Multixact ID wraparound)

The Best Information:

Autovacuum Log Events

log_autovacuum_min_duration

10 minutes is the default as of Postgres 15

10 seconds is the default on RDS

Most systems can run fine with 0

(log all autovacuums)

log_autovacuum_min_duration

```
LOG: automatic aggressive vacuum to prevent wraparound of table "postgres.public.table1": index scans: 0
    pages: 0 removed, 313661 remain, 0 skipped due to pins, 313660 skipped frozen
    tuples: 0 removed, 19133309 remain, 0 are dead but not yet removable, oldest xmin: 2409257730
    index scan not needed: 0 pages from table (0.00% of total) had 0 dead item identifiers removed
    I/O timings: read: 7.421 ms, write: 0.000 ms
    avg read rate: 57.698 MB/s, avg write rate: 0.627 MB/s
    buffer usage: 69 hits, 92 misses, 1 dirtied
    WAL usage: 1 records, 1 full page images, 2057 bytes
    system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.01 s

LOG: automatic vacuum of table "postgres.public.table2": index scans: 0
    pages: 0 removed, 952311 remain, 0 skipped due to pins, 0 skipped frozen
    tuples: 0 removed, 67714455 remain, 0 are dead but not yet removable, oldest xmin: 2410997930
    index scan not needed: 0 pages from table (0.00% of total) had 0 dead item identifiers removed
    I/O timings: read: 513.942 ms, write: 505.749 ms
    avg read rate: 36.341 MB/s, avg write rate: 45.740 MB/s
    buffer usage: 232417 hits, 137168 misses, 172644 dirtied
    WAL usage: 343213 records, 172440 full page images, 358270925 bytes
    system usage: CPU: user: 17.08 s, system: 1.08 s, elapsed: 29.48 s
```

log_autovacuum_min_duration

```
LOG: automatic aggressive vacuum to prevent wraparound of table "postgres.public.table1": index scans: 0
    pages: 0 removed, 313661 remain, 0 skipped due to pins, 313660 skipped frozen
    tuples: 0 removed, 19133309 remain, 0 are dead but not yet removable, oldest xmin: 2409257730
    index scan not needed: 0 pages from table (0.00% of total) had 0 dead item identifiers removed
    I/O timings: read: 7.421 ms, write: 0.000 ms
    avg read rate: 57.698 MB/s, avg write rate: 0.627 MB/s
    buffer usage: 69 hits, 92 misses, 1 dirtied
    WAL usage: 1 records, 1 full page images, 2057 bytes
    system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.01 s

LOG: automatic vacuum of table "postgres.public.table2": index scans: 0
    pages: 0 removed, 952311 remain, 0 skipped due to pins, 0 skipped frozen
    tuples: 0 removed, 67714455 remain, 0 are dead but not yet removable, oldest xmin: 2410997930
    index scan not needed: 0 pages from table (0.00% of total) had 0 dead item identifiers removed
    I/O timings: read: 513.942 ms, write: 505.749 ms
    avg read rate: 36.341 MB/s, avg write rate: 45.740 MB/s
    buffer usage: 232417 hits, 137168 misses, 172644 dirtied
    WAL usage: 343213 records, 172440 full page images, 358270925 bytes
    system usage: CPU: user: 17.08 s, system: 1.08 s, elapsed: 29.48 s
```

Autovacuum on table 1 had to do almost no work

Autovacuum on table 2 had to do a lot of work by comparison:

~1.1 GB of data read, 1.4 GB of data written, 360 MB of WAL written

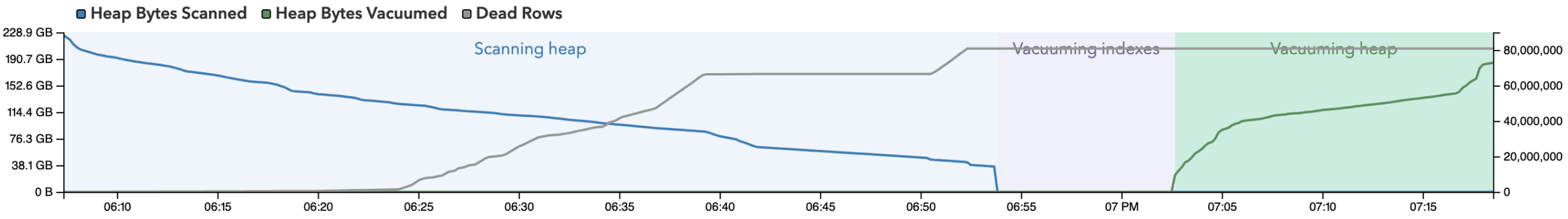
Combined Statistics and Logs in pganalyze

VACUUM Run 16898152330048412

Information

Table Name	public.snapshot_benchmarks	**Start Time**	Jul 19, 2023 6:07:12 PM PDT
Duration	an hour	**End Time**	Jul 19, 2023 7:18:40 PM PDT
Autovacuum	Yes	**Postgres Role**	n/a
Heap Blocks Total	228.1 GB · 29,891,683 blocks	**Max Dead Tuples / Phase**	178,956,969

Progress



Summary Statistics

Time Elapsed	1:11:25	**CPU Time**	user: 2281.30 s, system: 221.62 s
Index Vacuum Phases	1	**Aggressive Vacuum**	Yes
Pages Removed	0 / 0 B	**Pages Remaining**	29,891,683 / 228.1 GB
Pages Skipped Due To Pin	0 / 0 B	**Pages Skipped Frozen**	14,330,978 / 109.3 GB
Tuples Deleted	75,546,169	**Tuples Remaining**	1,757,947,289
Tuples Dead But Not Removable	0	**Data Read from Cache**	161.7 GB
Data Read from Disk	242.8 GB	**Data Flushed to Disk**	103.2 GB
Avg Read Rate	58.03 MB/s	**Avg Write Rate**	24.66 MB/s

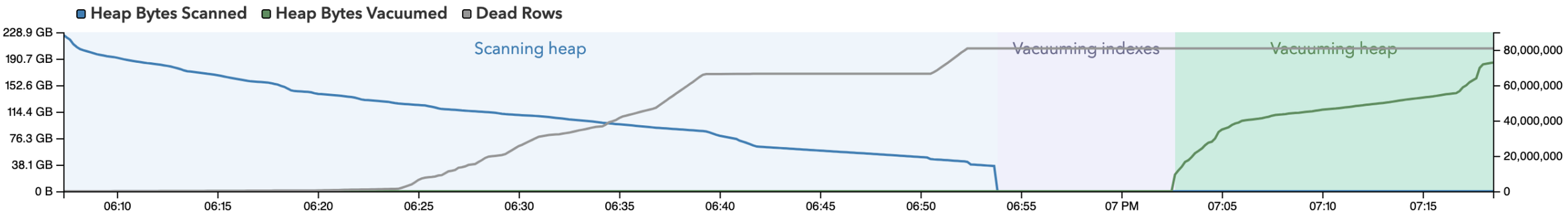
Combined Statistics and Logs in pganalyze

VACUUM Run 16898152330048412

Information

Table Name	public.snapshot_benchmarks	**Start Time**	Jul 19, 2023 6:07:12 PM PDT
Duration	an hour	**End Time**	Jul 19, 2023 7:18:40 PM PDT
Autovacuum	Yes	**Postgres Role**	n/a
Heap Blocks Total	228.1 GB · 29,891,683 blocks	**Max Dead Tuples / Phase**	178,956,969

Progress



Summary Statistics

Time Elapsed	1:11:25	**CPU Time**	user: 2281.30 s, system: 221.62 s
Index Vacuum Phases	1	**Aggressive Vacuum**	Yes
Pages Removed	0 / 0 B	**Pages Remaining**	29,891,683 / 228.1 GB
Pages Skipped Due To Pin	0 / 0 B	**Pages Skipped Frozen**	14,330,978 / 109.3 GB
Tup			
Tup			
Data Read from Disk	242.8 GB	**Data Flushed to Disk**	103.2 GB
Avg Read Rate	58.03 MB/s	**Avg Write Rate**	24.66 MB/s

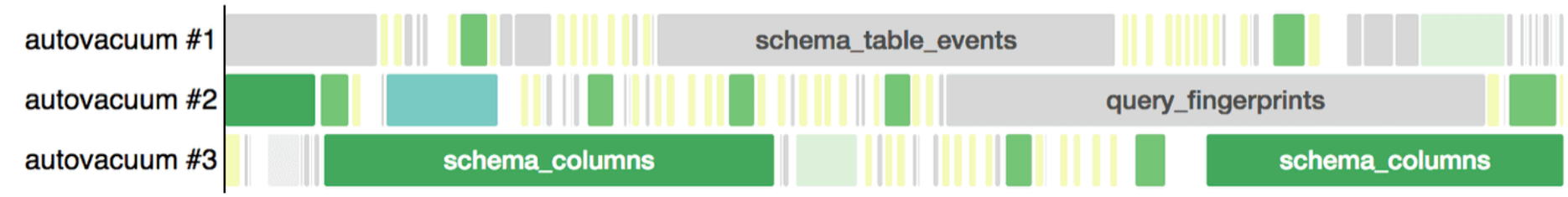
242.8 GB read

103.2 GB written

But, we've had autovacuum statistics in pganalyze for almost 6 years

(and its been very useful to many customers!)

Visualizing & Tuning Postgres Autovacuum



By **Lukas Fittl**

November 28, 2017

In this post we'll take a deep dive into one of the mysteries of PostgreSQL: VACUUM and autovacuum.

The Postgres autovacuum logic can be tricky to understand and tune - it has many moving parts, and is hard to understand, in particular for application developers who don't spend all day looking at database documentation.

But luckily there are recent improvements in Postgres, in particular the addition of pg_stat_progress_vacuum in Postgres 9.6, that make understanding autovacuum and VACUUM behavior a bit easier.

In this post we describe an approach to autovacuum tuning that is based on sampling these statistics

What's new?

Introducing pganalyze VACUUM Advisor: Workload-aware autovacuum tuning for Postgres

Keiko Oda
Engineering

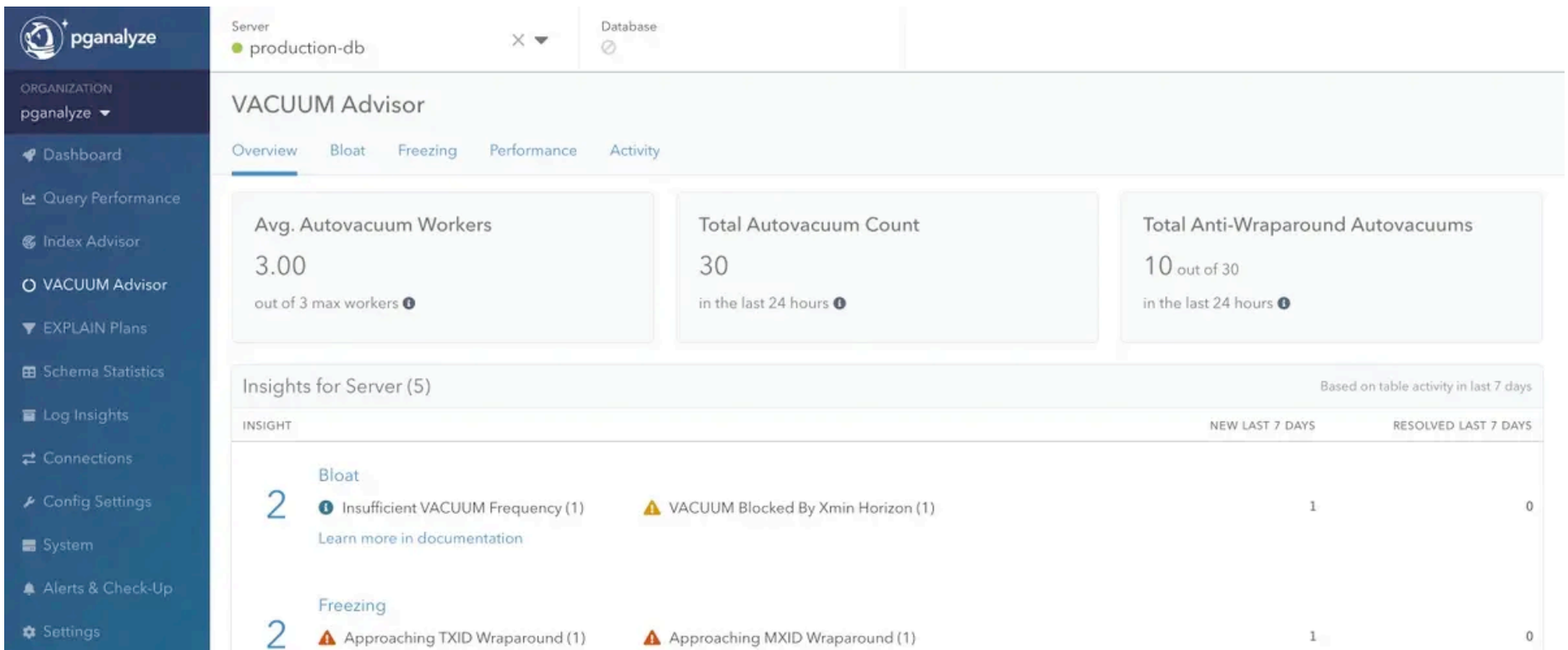
Maciek Sakrejda
Engineering

Jens Nikolaus
Design

Published on:
July 25, 2023

VACUUM in Postgres is a fact of life - every Postgres installation out there today, including those on AWS Aurora and Google AlloyDB, has to run autovacuum on each table at least every 2 billion transactions to perform freezing. And if you'd like to keep the active portion of your tables in memory, **a frequent vacuuming cycle ensures tables stay small, and queries remain fast**, with minimal bloat. Oftentimes, when your database is still small, you don't have to think about VACUUM and autovacuum. However as your database grows, the system defaults for your workload's Postgres autovacuum configuration are no longer adequate. Additionally, over time you may run into particular workload characteristics that require per-table tuning and a raised awareness of problems such as lock conflicts.

Today, we are excited to announce **a better way to help you tune Postgres autovacuum settings**, and discover problems that block VACUUM right away: The pganalyze VACUUM Advisor. With the pganalyze VACUUM Advisor, tuning Postgres to match the needs of your workloads is fast, predictable and easy.



Going from simple metrics, to:

=> Making underlying Postgres processes transparent

=> Transforming data points to meaningful information

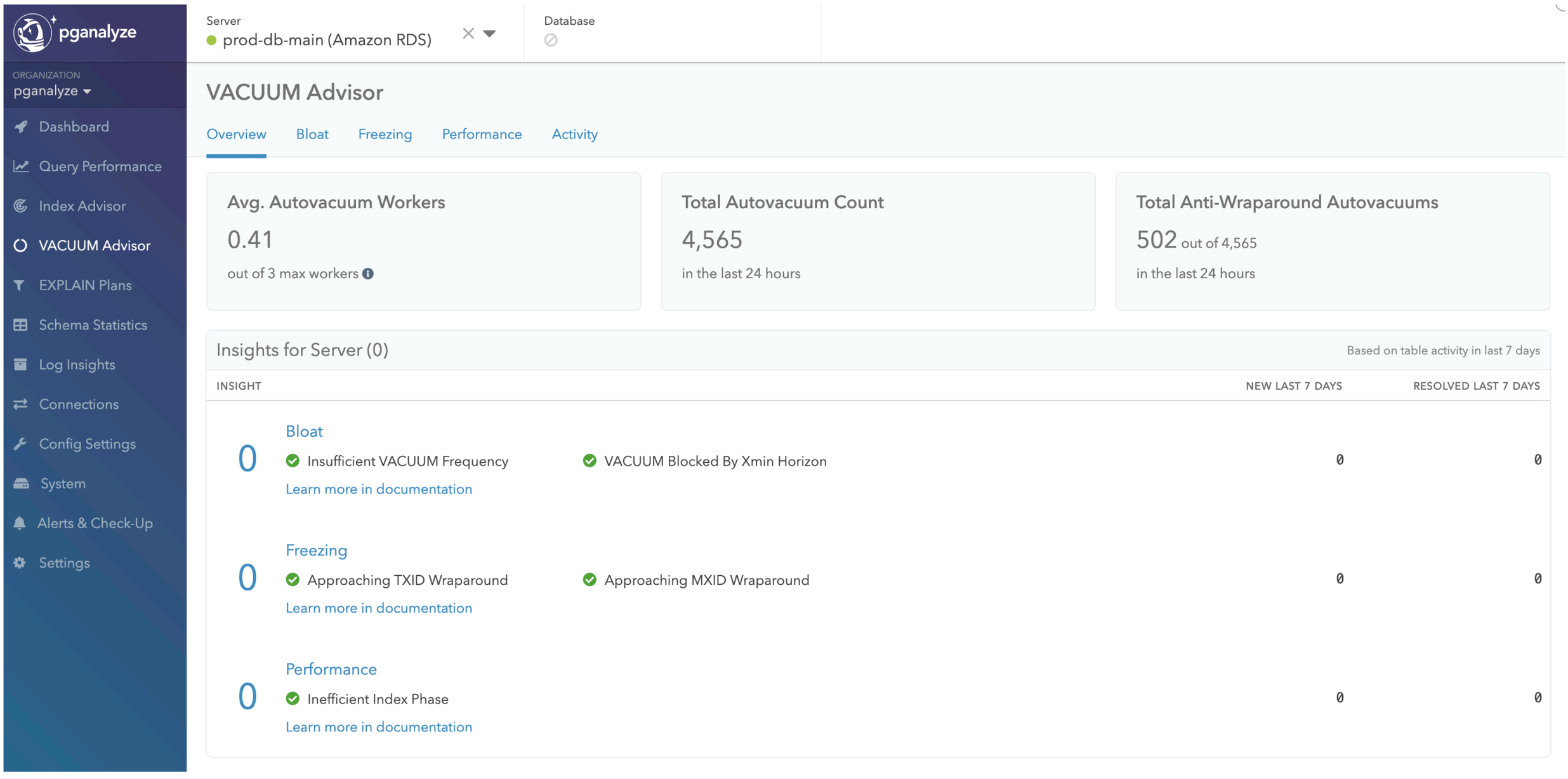
=> Finding insights for tuning workload-specific settings

=> Getting built-in alerts for critical problems

pganalyze

The concepts and structure behind VACUUM Advisor

pganalyze VACUUM Advisor



Behind the scenes

Production DB

pganalyze-collector

- Gather autovacuum and freezing metrics
- Transform XIDs to full 64-bit version
- Collect column statistics for bloat estimates
- Capture autovacuum logs (if available)

pganalyze

VACUUM Advisor

VACUUM Advisor Background Process

Workload Repository

pganalyze API

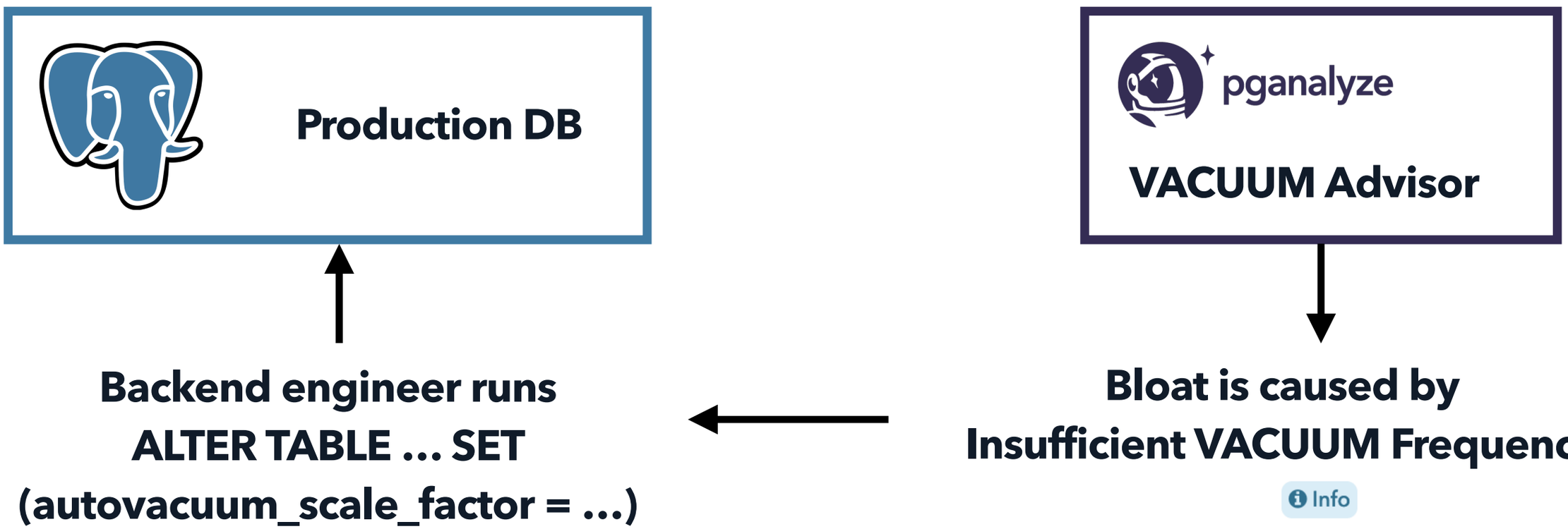
Three ways to think about VACUUM

1. Avoid **Bloat,** by vacuuming dead rows early, and preventing blocked VACUUMs due to the xmin

2. Understand **Freezing**, and how Postgres avoids Transaction ID and Multixact ID wraparound

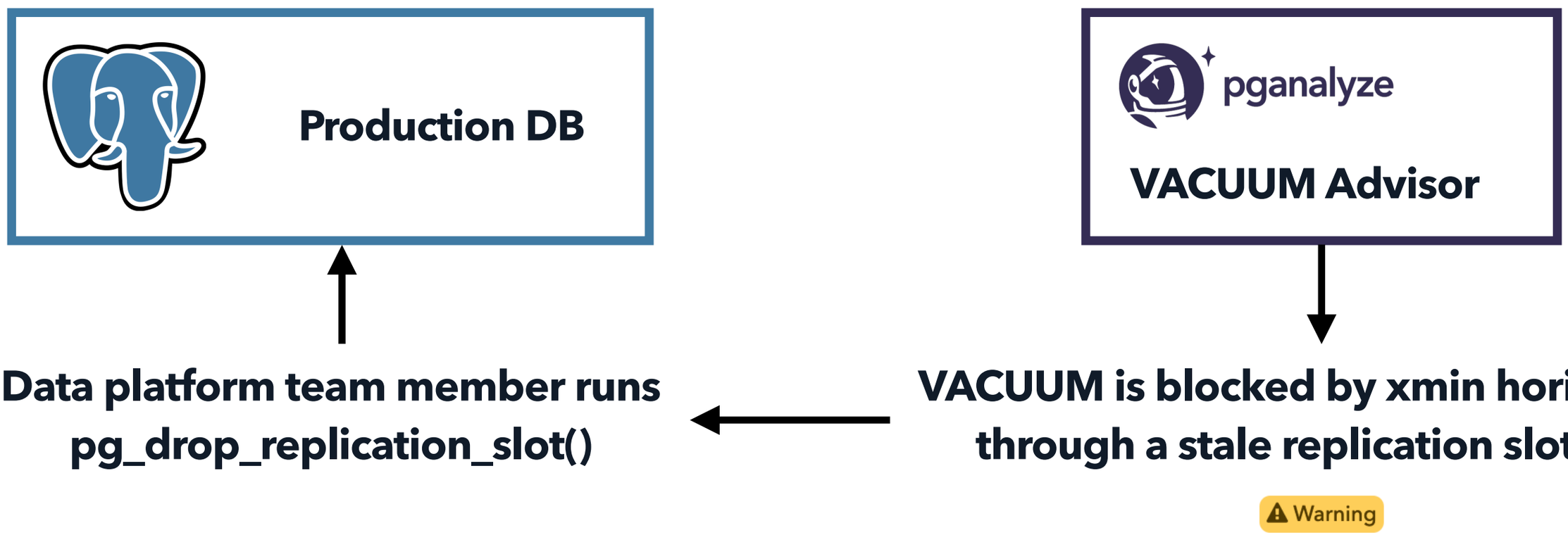
3. Optimize **Performance**, by giving autovacuum enough resources, and avoiding conflicting locks

Using VACUUM Advisor



pganalyze

Using VACUUM Advisor



pganalyze

Using VACUUM Advisor

On-call engineer checks pganalyze in detail, and considers:

1) **Letting an existing autovacuum complete**
2) **Running manual VACUUM**
3) **Taking other actions (e.g. TRUNCATE, if data can be rebuilt)**

and tracks it closely!

Database Is Approaching 80% of TXID Wraparound, and table X is the cause

Critical

pganalyze

pganalyze

Bloat estimates and tuning the dead row threshold

Table bloat =
A less than optimal “page density”

(i.e. how many live tuples are on each page,
vs how many could potentially fit there)

ioguix / pgsql-bloat-estimation Public

Code Issues 2 Pull requests 2 Actions Projects Wiki Security Insights

master pgsql-bloat-estimation / table / table_bloat.sql

63 lines (63 loc) · 3.32 KB

```
1   /* WARNING: executed with a non-superuser role, t
2   * This query is compatible with PostgreSQL 9.0 an
3   */
4   SELECT current_database(), schemaname, tblname, b
5     (tblpages-est_tblpages)*bs AS extra_size,
6     CASE WHEN tblpages > 0 AND tblpages - est_tblpa
7       THEN 100 * (tblpages - est_tblpages)/tblpages
8       ELSE 0
9     END AS extra_pct, fillfactor,
10    CASE WHEN tblpages - est_tblpages_ff > 0
11      THEN (tblpages-est_tblpages_ff)*bs
12      ELSE 0
13    END AS bloat_size,
14    CASE WHEN tblpages > 0 AND tblpages - est_tblpages_ff > 0
15      THEN 100 * (tblpages - est_tblpages_ff)/tblpages::float
16      ELSE 0
17    END AS bloat_pct, is_na
18    -- , tpl_hdr_size, tpl_data_size, (pst).free_percent + (pst).dead_tuple_percent AS real_frag -- (DEBUG INFO)
19  FROM (
20    SELECT ceil( reltuples / ( (bs-page_hdr)/tpl_size ) ) + ceil( toasttuples / 4 ) AS est_tblpages,
21      ceil( reltuples / ( (bs-page_hdr)*fillfactor/(tpl_size*100) ) ) + ceil( toasttuples / 4 ) AS est_tblpages_ff,
22      tblpages, fillfactor, bs, tblid, schemaname, tblname, heappages, toastpages, is_na
```

Bloat estimation queries look scary and complicated

But the idea is simple.

Based on the:

- Number of tuples on the table (per ANALYZE / VACUUM)
- Average column width (as collected by ANALYZE)
- Alignment of columns (as defined by the platform)
- FILLFACTOR (as configured on the table)

What is the **optimal size of the table**,
if all pages were packed efficiently?
(and how does current table size compare to that?)

Are we running bloat estimation queries to provide bloat estimates in pganalyze?

Not directly. We just need column statistics*, and a few other data points. No additional queries necessary, and no overhead!

* requires setup of the pganalyze.get_column_stats helper function.

pganalyze

Bloat estimates and tuning the dead row threshold

VACUUM Advisor

Overview | Bloat | Freezing | Performance | Activity

Est. Table Bloat
525.6 GB 8%
based on 834 of 1,135 tables

Total Tables
1,135

Xmin Horizon Assigned
0.3
hours ago

Insufficient VACUUM Frequency

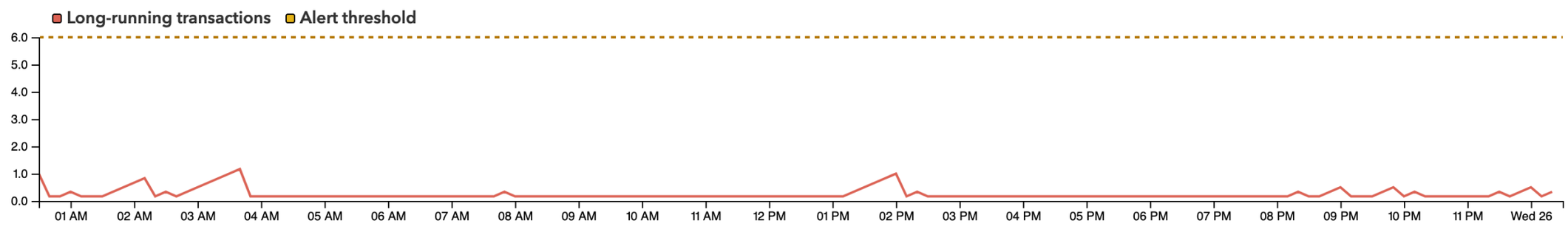
Detects when insufficient VACUUM frequency leads to new bloat in a table. It analyzes the table statistics from the past seven days and calculates the amount of table growth that could have been avoided if VACUUM had been performed more frequently. Detection works best after removing existing bloat on the table, e.g. through pg_repack. Learn more

Looks good (last checked 3 minutes ago)

VACUUM Blocked by Xmin Horizon

Detects when the xmin horizon on a server was assigned too far in the past, preventing vacuum from performing necessary cleanup of dead rows. Learn more

Looks good (last checked 6 minutes ago)



Bloat-Related Config Settings

SETTING	CURRENT VALUE
autovacuum_vacuum_scale_factor	0.1
autovacuum_vacuum_threshold	50

Table Bloat Info

Search...

DATABASE	SCHEMA	TABLE	SCALE FACTOR	THRESHOLD	DATA SIZE	EST. BLOAT	EST. BLOAT %	LAST VACUUMED
pgaweb	public	snapshot_benchma…	[default]	[default]	228.1 GB	62.2 GB	27%	2023-07-23 04:38…
pgaweb	public	query_fingerprints	[default]	[default]	37.9 GB	30.1 GB	79%	2023-07-23 03:09

pganalyze

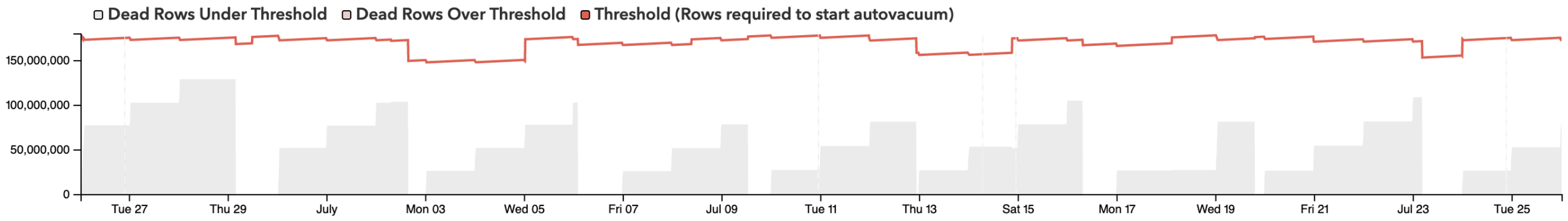
Bloat estimates and tuning the dead row threshold

Table: public.snapshot_benchmarks

Statistics | Partitions | Queries | Columns | Indexes | Constraints | VACUUM/ANALYZE Activity

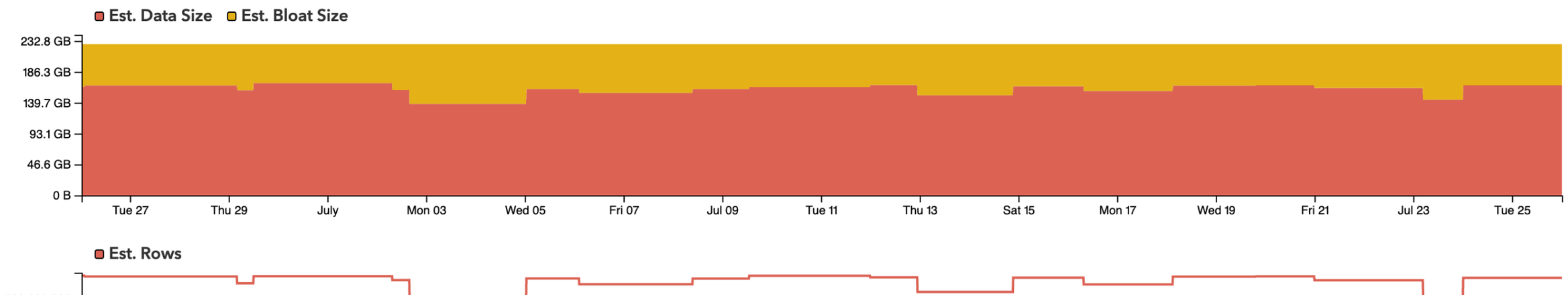
VACUUM Activity

Avg. VACUUM Duration	An hour	**Number of VACUUMs / day**	0
Autovacuum Enabled	Yes	**Autovacuum Freeze Max Age**	400,000,000 / 400,000,000
Autovacuum Cost Delay	2 ms	**Autovacuum Cost Limit**	1,800
Autovacuum Threshold	50	**Autovacuum Scale Factor**	10%



Estimated Table Bloat

Estimated Data Size	165.9 GB	**Estimated Table Bloat**	62.2 GB



pganalyze

Bloat estimates and tuning the dead row threshold

Issue #138: VACUUM: Bloat - Insufficient VACUUM Frequency

Overview

Severity
Info

Check Frequency
Daily

Last Updated
2021-03-22 04:00:00pm UTC

State
Acknowledged

Re-open

Description
Insufficient vacuuming is causing avoidable growth on table my_table

Guidance

```
ALTER TABLE my_schema.my_table SET
 (autovacuum_vacuum_scale_factor = 0.01);
```

Copy ALTER TABLE command

Avoidable table growth due to insufficient VACUUM frequency was detected. You can adjust the frequency of autovacuum by changing relevant config settings. Lowering `autovacuum_vacuum_threshold` or `autovacuum_vacuum_scale_factor` will increase the frequency of autovacuum in general.

› How to apply changes

› What to check following ALTER TABLE

To confirm the impact of changes, it is also recommended to run pg_repack to reclaim disk space. You can check out a graph of estimated bloat over time on the VACUUM/ANALYZE Activity page.

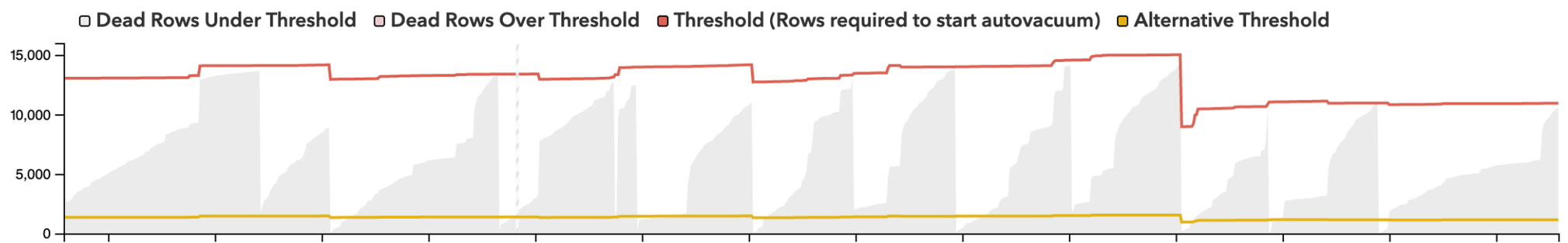
Recommendation

	Current	Recommended
autovacuum_vacuum_threshold	50	50
autovacuum_vacuum_scale_factor	0.10	0.01

Estimated Improvement

	Current	Estimate
Total autovacuum count	7	68
Avoidable growth		
Rows	58,527	30,966
Percentage	25.9%	13.7%
Bytes	11.1 MB	5.8 MB

VACUUM Activity



To start, the **Insufficient VACUUM Frequency** insight
will *not* trigger for tables with existing, stable bloat.

In those cases you will need to first run
VACUUM FULL (or rather, pg_repack)
in order to have VACUUM Advisor
detect new bloat caused by bad settings.

pganalyze

How pganalyze tracks and alerts on the xmin horizon

What is the xmin horizon?

It limits which dead tuples (row versions) can be cleaned up by a VACUUM.

“At this point we decided we needed some help.

…

It turned out that the issue wasn’t that auto-vacuum wasn’t being run often enough, but **that it was blocked by something** (spoiler alert: at least one transaction) and couldn’t remove dead tuples while working. **It was really clear when we examined the logs.”**

- Mathieu Tamer at Precogs

https://medium.com/precogs-tech/vacuuming-dead-tuples-not-yet-removable-in-postgresql-and-other-tales-303bfb2d87f4

```
LOG: automatic vacuum of table "mydb.myschema.mytable": index scans: 0
    pages: 0 removed, 14761 remain, 0 skipped due to pins, 12461 skipped frozen
    tuples: 0 removed, 122038 remain, 14433 are dead but not yet removable, oldest xmin: 538040633
    index scan bypassed: 255 pages from table (1.73% of total) have 661 dead item identifiers
    I/O timings: read: 0.000 ms, write: 0.000 ms
    avg read rate: 0.000 MB/s, avg write rate: 0.000 MB/s
    buffer usage: 4420 hits, 0 misses, 0 dirtied
    WAL usage: 1 records, 0 full page images, 245 bytes
    system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.01 s
```

This VACUUM did not remove any dead rows.

It just scanned them and determined it can't clean them up.

It will run again shortly, to try once more.

The xmin horizon can be “held back” by:

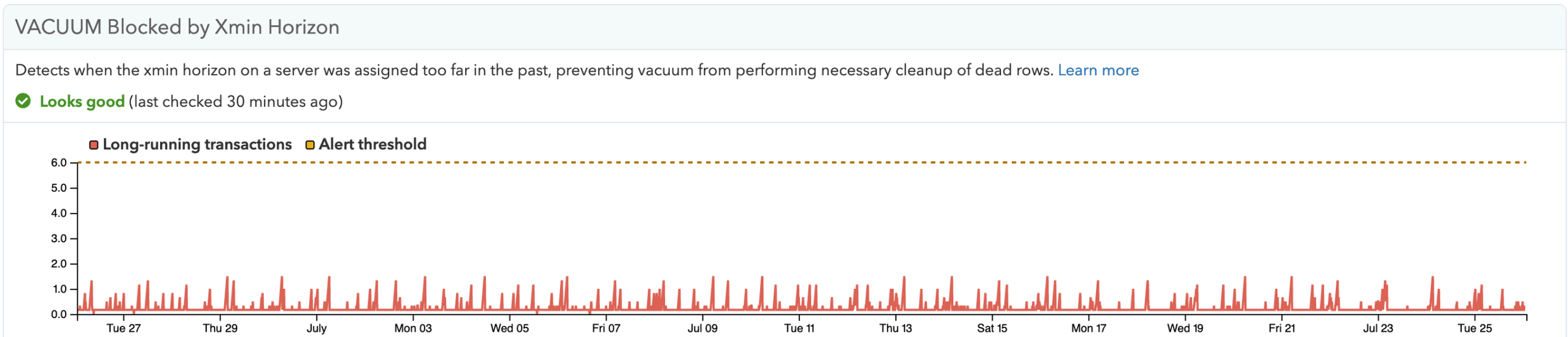
- Long running transactions
  (involving any table, not just the one being vacuumed!)
- Lagging or stale physical/logical replication slots
- Abandoned prepared transactions
- Long running queries on standbys
  (with hot_standby_feedback = on)

pganalyze

How pganalyze tracks and alerts on the Xmin Horizon

The usual situation:

Long-running transactions prevent some cleanup, but its not a problem



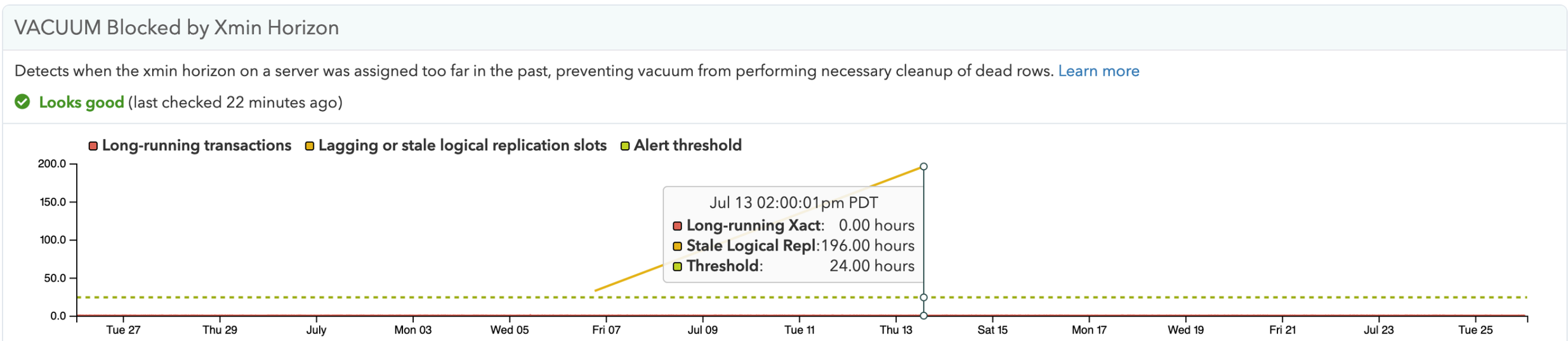
pganalyze

How pganalyze tracks and alerts on the Xmin Horizon

The bad situation:

For 196 hours no VACUUM could clean up Bloat,

because a stale logical replication slot was still open



=> This will directly cause bloat, slow down your queries,

and require pg_repack to get table sizes down again

pganalyze

How pganalyze tracks and alerts on the Xmin Horizon

Issue #139: VACUUM: Bloat - VACUUM Blocked By Xmin Horizon

Overview

Severity
⚠ Warning

Check Frequency
Every 30 minutes

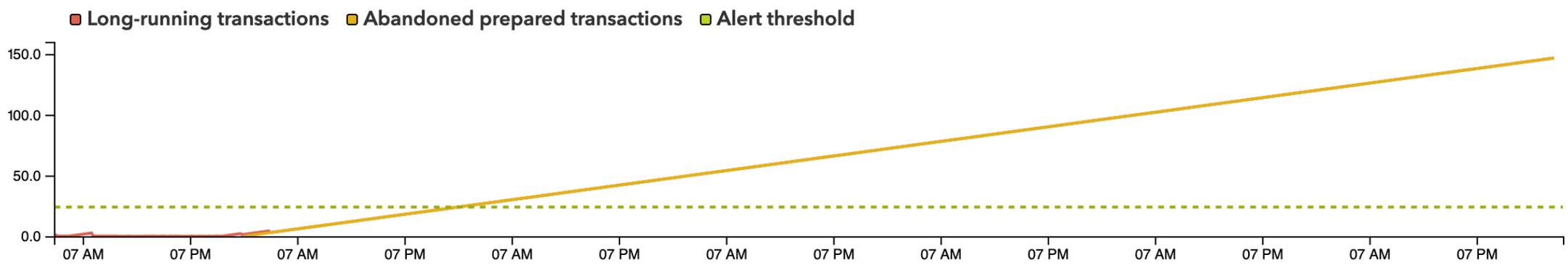
Last Updated
2021-03-22 04:00:00pm UTC

State
Acknowledged

Re-open

Description
VACUUM is potentially blocked on the server **server1** due to the xmin horizon being behind

Xmin Horizon Assignment Age Last 7 Days (in hours)



Identified Causes

CAUSE	XMIN HORIZON	XMIN ASSIGNED AT
Long-running transactions	10,569,525	2023-07-20 04:00:00pm UTC (6 days 8 hours ago)
Abandoned prepared transactions	10,569,525	2023-07-20 04:00:00pm UTC (6 days 8 hours ago)

Blocked VACUUMs Last 24 Hours

TABLE	OLDEST XMIN	NOT REMOVABLE DEAD ROWS	BLOCKED VACUUM COUNT
public.table2	2,000	8,287	2
public.table1	1,000	1,234	7

One essential implementation detail:

pganalyze tracks and alerts on **calendar time (hours),**
not age(…xmin) - aka transaction ID distance - like the typical monitoring queries for xmin horizon.

Is 200,000 transactions ago a lot? Hard to say.
Is no dead rows being cleaned up for 24 hours bad? Very likely.

pganalyze

Freezing metrics and anti-wraparound vacuums

pganalyze

Freezing metrics and anti-wraparound vacuums

VACUUM Advisor

Overview | Bloat | Freezing | Performance | Activity

Anti-Wraparound Vacuum Frequency

3 days 9 hours

for each table

Transaction ID Allocation

72,057

per minute

Transaction ID Utilization

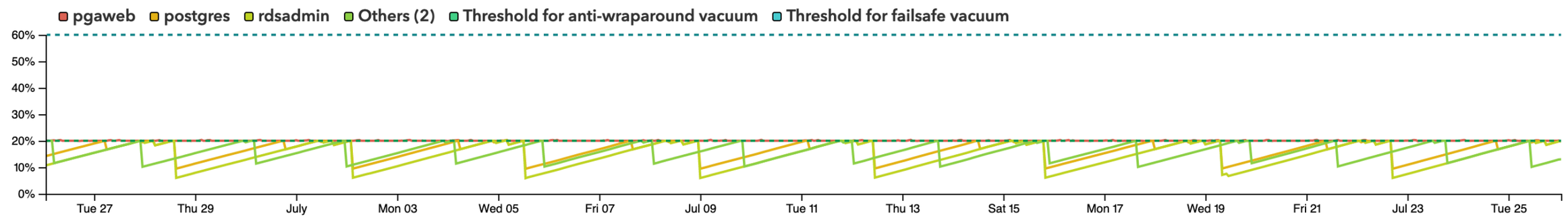
19.9 %

Approaching Transaction ID Wraparound

Detects when a server is consuming a large portion of the Transaction ID space and in danger of triggering transaction ID wraparound prevention mechanisms. Learn more

Looks good (last checked 5 minutes ago)

autovacuum_freeze_max_age 400,000,000 (20.00%) vacuum_failsafe_age 1,200,000,000 (60.00%)

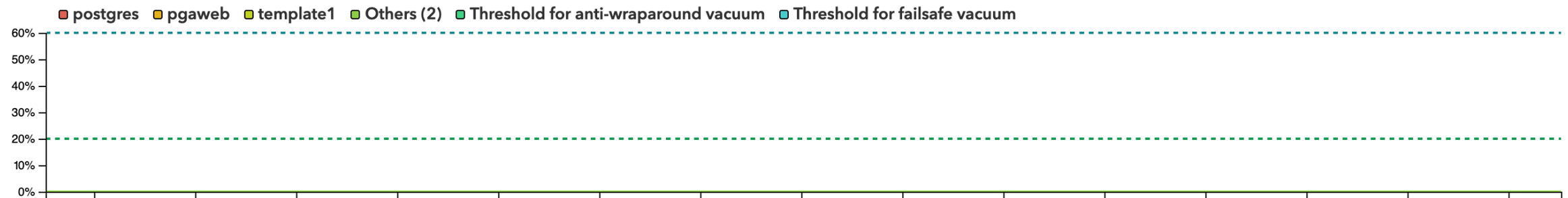


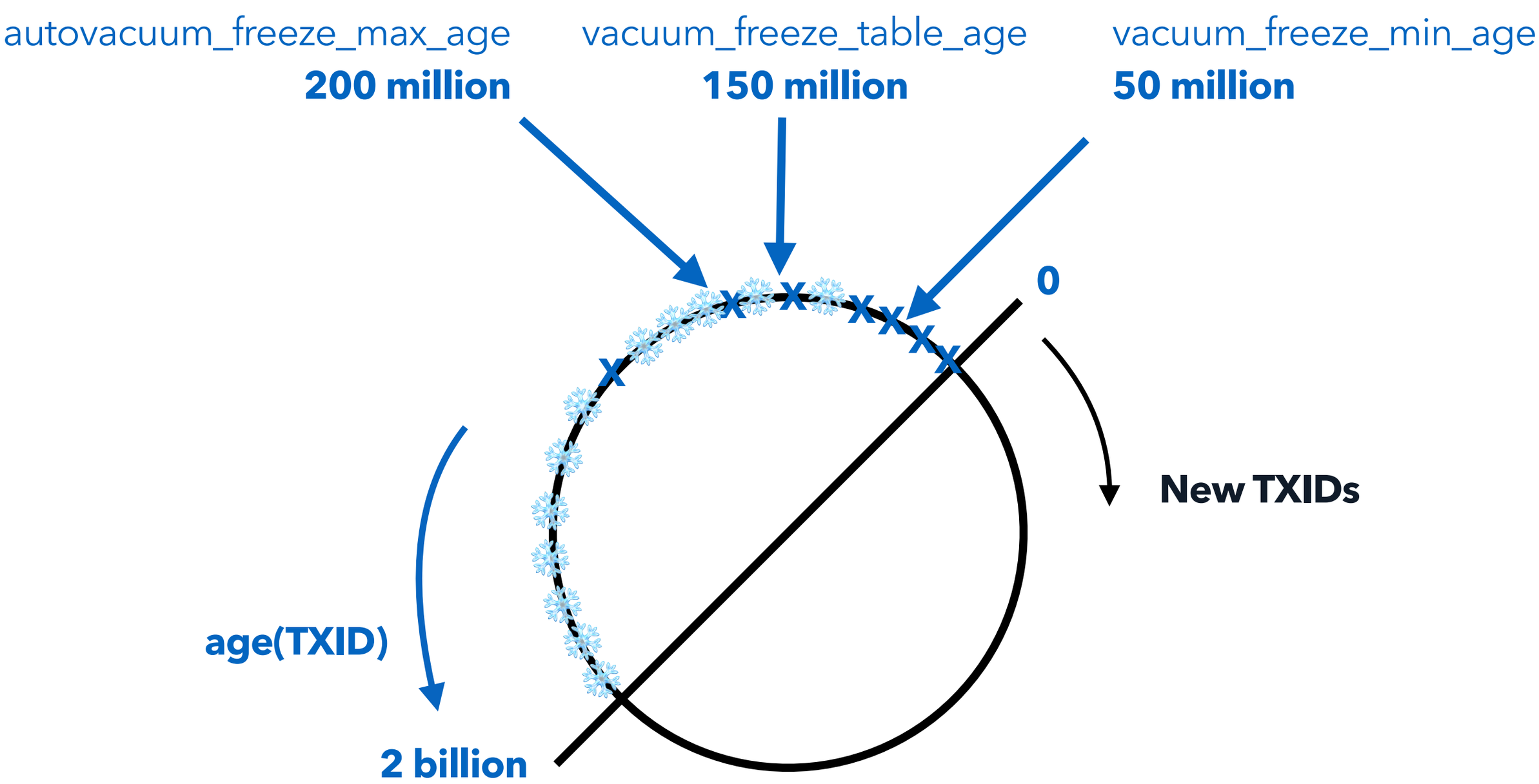
Approaching Multixact ID Wraparound

Detects when a server is consuming a large portion of the Multixact ID space and in danger of triggering multixact ID wraparound prevention mechanisms. Learn more

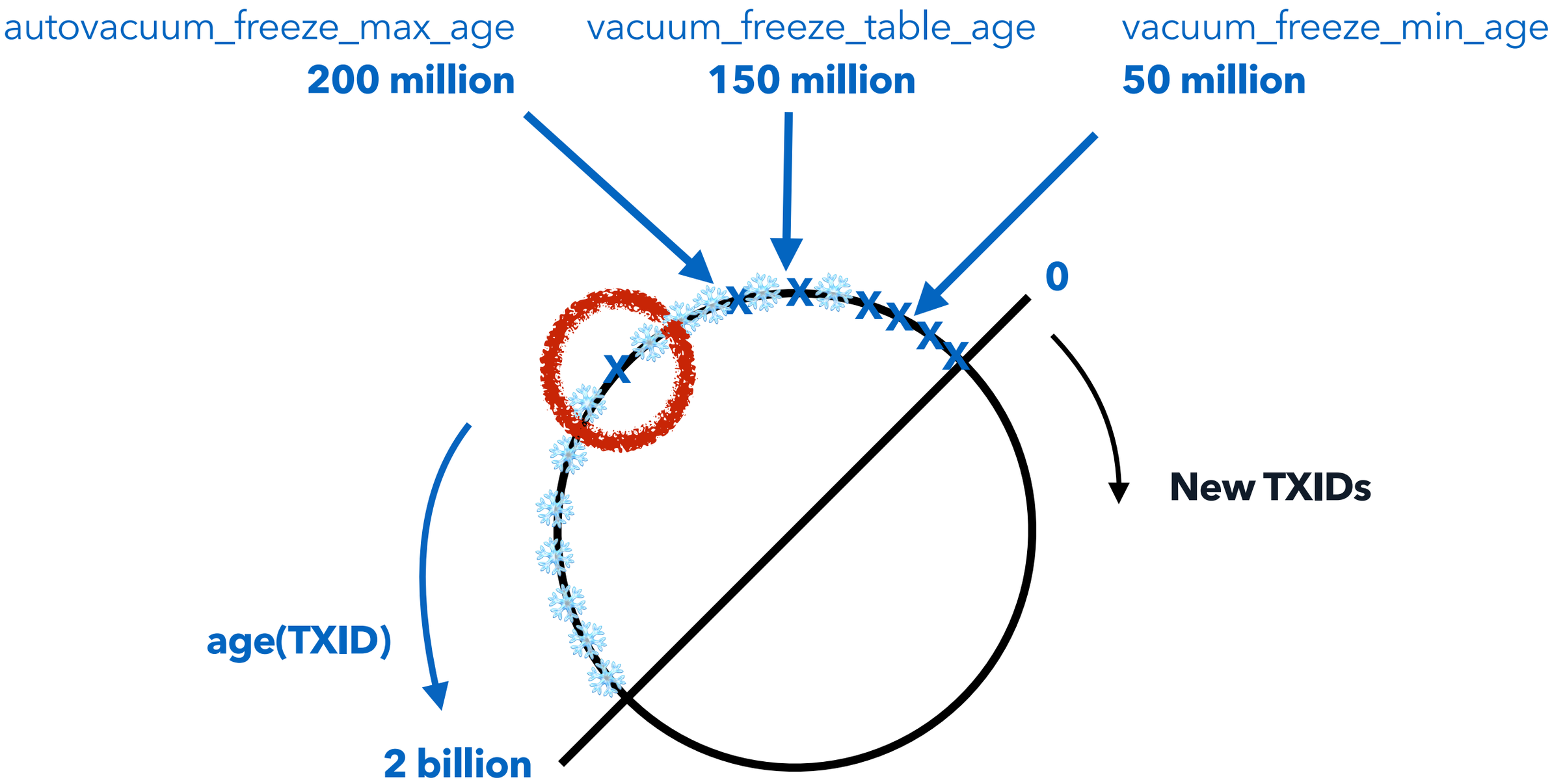
Looks good (last checked 5 minutes ago)

autovacuum_multixact_freeze_max_age 400,000,000 (20.00%) vacuum_multixact_failsafe_age 1,200,000,000 (60.00%)



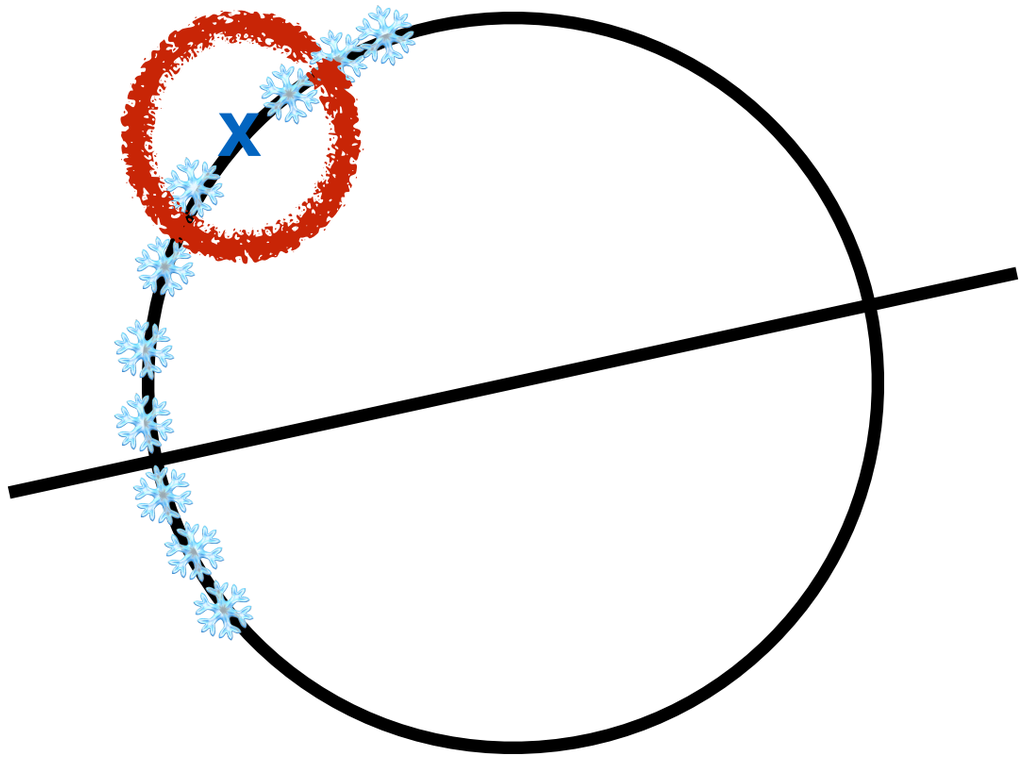

autovacuum_freeze_max_age
200 million
vacuum_freeze_table_age
150 million
vacuum_freeze_min_age
50 million
0
New TXIDs
age(TXID)
2 billion

pganalyze


autovacuum_freeze_max_age
200 million
vacuum_freeze_table_age
150 million
vacuum_freeze_min_age
50 million
0
New TXIDs
age(TXID)
2 billion

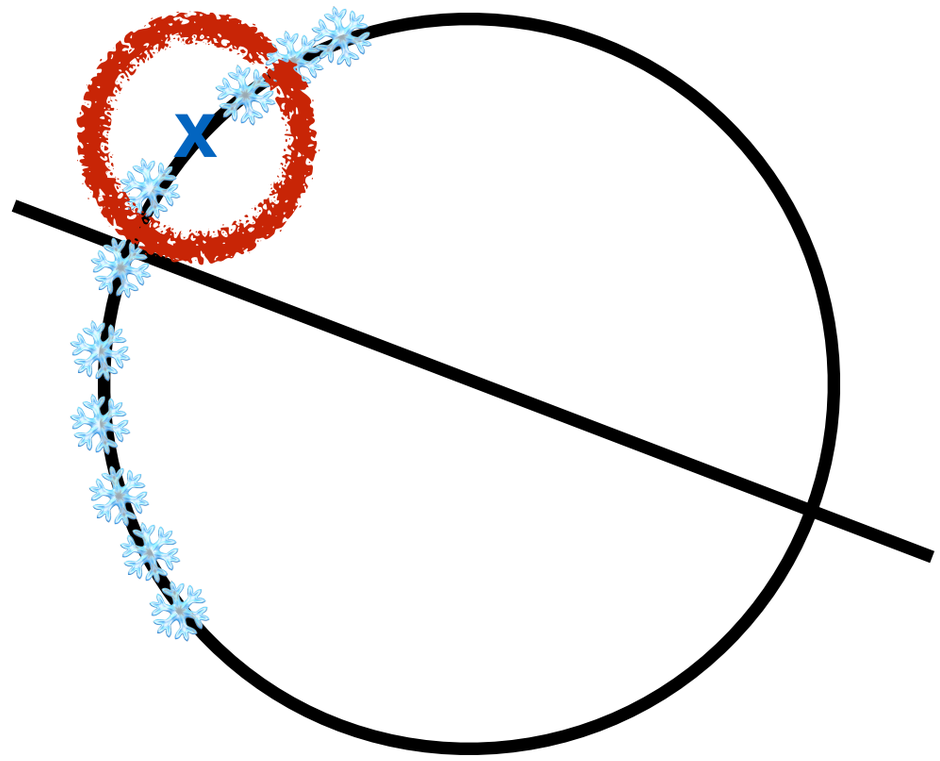
pganalyze

VACUUM is running,
but it hasn't finished :(



pganalyze

Database shut down and
we have to manually VACUUM it



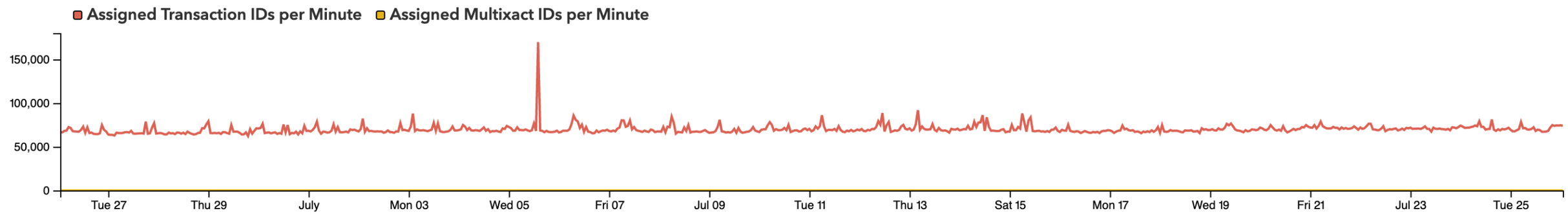
pganalyze

pganalyze

Freezing metrics and anti-wraparound vacuums

Assigned Transaction IDs per Minute

Est. Frequency of Transaction ID Wraparound Autovacuums	3 days 12 hours
Est. Frequency of Multixact ID Wraparound Autovacuums	-



Oldest Unfrozen Transaction ID Age Stats by Table (871)

Jul 26, 2023 1:00:00 AM PDT

Search...

DATABASE	SCHEMA	TABLE	OLDEST UNFROZEN XID AGE	LAST VACUUM AT
pgaweb	public	indexing_engine_runs_35d_20230716	397,073,249	Jul 22 04:29:25pm PDT
pgaweb	public	schema_table_events_7d_20230722	9	Jul 22 04:30:50pm PDT
pgaweb	public	postgres_settings	8	Jul 22 04:57:48pm PDT
pgaweb	public	schema_table_infos_35d_20230723	9	Jul 23 10:30:58am PDT
pgaweb	public	schema_index_stats_35d_20230723	394,921,399	Jul 23 04:24:26pm PDT
pgaweb	public	query_stats_35d_20230723	394,921,399	Jul 23 03:43:34pm PDT
pgaweb	public	schema_column_stats_7d_20230723	394,921,399	Jul 23 03:10:23pm PDT
pgaweb	public	log_files_7d_20230723	394,812,988	Jul 23 01:35:43pm PDT
pgaweb	public	backend_counts_7d_20230723	394,812,988	Jul 23 02:21:05pm PDT
pgaweb	public	database_stats_35d_20230723	394,812,988	Jul 23 03:20:19pm PDT

Oldest Unfrozen XID:
31:2,026,045,442
Oldest Unfrozen XID was assigned at:
3 days 20 hours ago

1-10 of 871

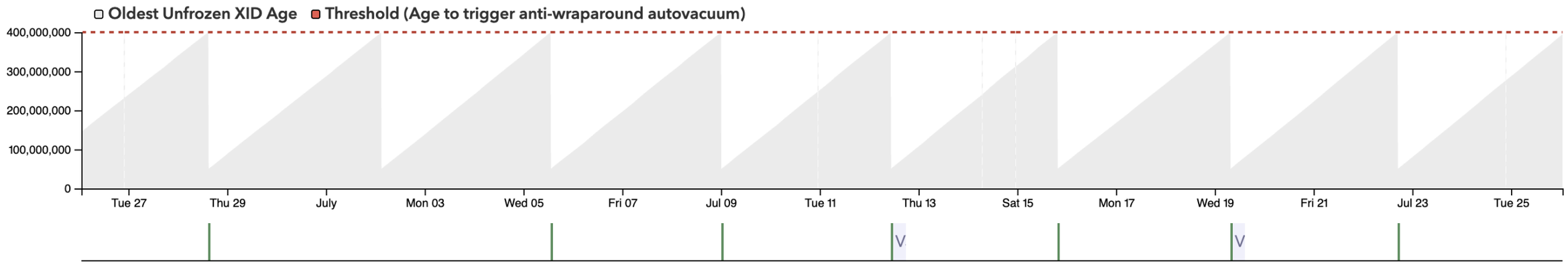
pganalyze

Freezing metrics and anti-wraparound vacuums

Table: public.postgres_settings

Statistics | Partitions | Queries | Columns | Indexes | Constraints | VACUUM/ANALYZE Activity

VACUUM Activity



Show TOAST VACUUMs

ID	STARTED BY	TOAST	TIME STARTED	TIME FINISHED	FREEZING PROGRESS	DEAD TUPLE PROGRESS
n/a	autovacuum	No	Jul 22 04:57:44pm PDT	Jul 22 04:57:48pm PDT	+350,006,146	-15,302
16897775570011909	autovacuum	No	Jul 19 07:39:16am PDT	Jul 19 07:39:24am PDT	+350,000,252	-42,725
n/a	autovacuum	No	Jul 15 07:38:22pm PDT	Jul 15 07:38:26pm PDT	+349,557,546	-13,785
16891833360041383	autovacuum	No	Jul 12 10:35:35am PDT	Jul 12 10:35:43am PDT	+350,005,074	-12,791
n/a	autovacuum	No	Jul 09 12:08:33am PDT	Jul 09 12:08:36am PDT	+349,952,510	-16,883
n/a	autovacuum	No	Jul 05 01:12:12pm PDT	Jul 05 01:12:16pm PDT	+350,008,857	-15,940
n/a	autovacuum	No	Jun 28 02:51:50pm PDT	Jun 28 02:51:52pm PDT	+350,003,938	-20,308

pganalyze

Understanding and tuning autovacuum performance

pganalyze

Understanding and tuning autovacuum performance

VACUUM Advisor

Overview | Bloat | Freezing | Performance | Activity

Avg. Autovacuum Workers

0.42

out of 3 max workers

Total Autovacuum Count

4,546

in the last 24 hours

Skipped Autovacuums

0

skipped due to locks in the last 24 hours

Inefficient Index Phase

Detects when autovacuum runs in this server are forced to perform inefficient multiple index scan phases due to limited configured memory. Learn more

Looks good (last checked An hour ago)

Performance-Related Config Settings

Max Read I/O Impact 1.4 GB / s

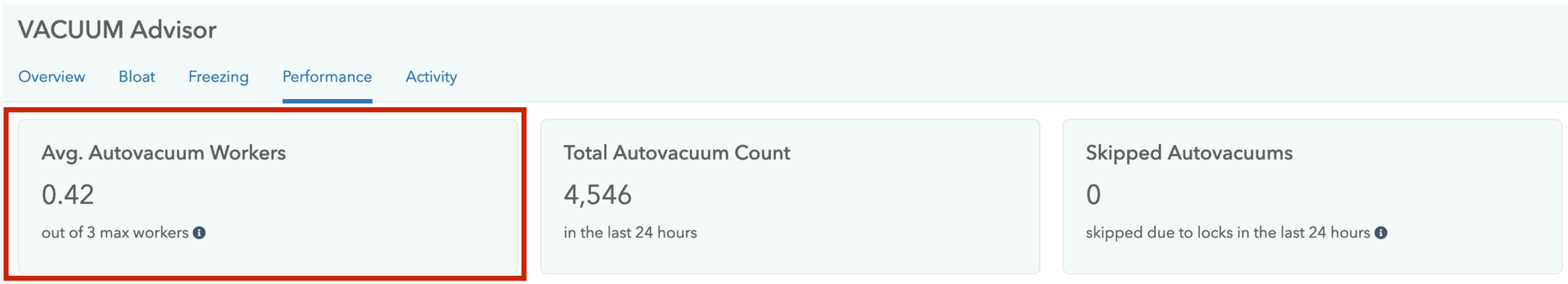
Max Write I/O Impact 351.6 MB / s

SETTING	CURRENT VALUE
autovacuum_max_workers	3
autovacuum_naptime	30 s
autovacuum_vacuum_cost_delay	2 ms
autovacuum_vacuum_cost_limit	1,800
autovacuum_work_mem	-1

Table VACUUM summary

Search...

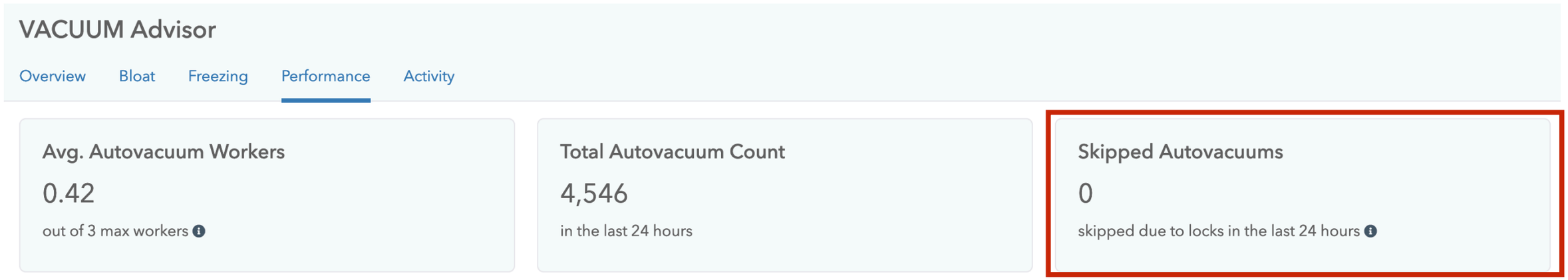
DATABASE	SCHEMA	TABLE	COUNT	MAX INDEX PHASES	MAX DURATION
pgaweb	public	schema_functions	1	1	A few seconds
pgaweb	public	schema_constraints	1	1	A few seconds
pgaweb	public	schema_table_scan_query_associations	1	1	2 minutes
pgaweb	public	issue_references	1	1	6 minutes
pgaweb	public	schema_table_scan_methods	1	1	An hour
pgaweb	public	query_analyses	1	1	15 minutes
template1	pg_toast	pg_toast_13799	0	0	-
template1	pg_toast	pg_toast_13794	0	0	-



Number of active autovacuum workers over 24 hours, compared to the maximum allowed workers

If this is close to the maximum, **you likely have an autovacuum backlog.**

Increasing autovacuum_max_workers *and* adjusting cost delay settings is recommended.



Regular autovacuums (which are not for anti-wraparound, i.e. freezing, reasons) **will be skipped when a conflicting lock is held on a table.**

The most common problem is doing an explicit “LOCK TABLE” as part of your application logic, e.g. to implement gapless invoice sequence numbers.

This will prevent these regular autovacuums from running at all, requiring a later anti-wraparound vacuum to do all the work, **causing more I/O at once**.

When pganalyze VACUUM Advisor tells you about **multiple index phases, your autovacuum_work_mem might be too low!**

Issue #137: VACUUM: Performance - Inefficient index phase

Overview

Severity
Info

Check Frequency
Daily

Last Updated
2021-03-22 04:00:00pm UTC

State
Acknowledged

Re-open

Description
Autovacuum needs multiple index vacuum phases when vacuuming indexes. Try increasing autovacuum_work_mem (currently 128MB)

VACUUM Runs

ID	START TIME	END TIME	DATABASE	SCHEMA	TABLE	INDEX PHASES
1	Mar 22 03:00:00pm UTC	Mar 22 03:20:00pm UTC	pgaweb	public	queries	3
2	Mar 22 01:10:00pm UTC	Mar 22 01:15:00pm UTC	pgaweb	public	servers	2

Guidance

Impact

Autovacuum on affected tables may take significantly longer. VACUUM processes tables and their indexes in phases. To avoid impacting your primary workload, autovacuum is limited to using a certain amount of memory. If this is not enough to process all dead rows, the index scan phase will need to be repeated (possibly several times). This is inefficient in terms of CPU and I/O.

Common Causes

Check Configuration

Detects when autovacuum runs in this server are forced to perform inefficient multiple index scan phases due to limited configured memory, and creates an issue with severity "warning". Triggers when at least `1` autovacuum run in the last 24h have had multiple index scan phases. Resolves automatically once fewer than `1` autovacuum run in the last 24h have had multiple index scan phases.

Ignores situations where configured autovacuum memory is already set to the maximum possible value of `1GB`.

Configure this check

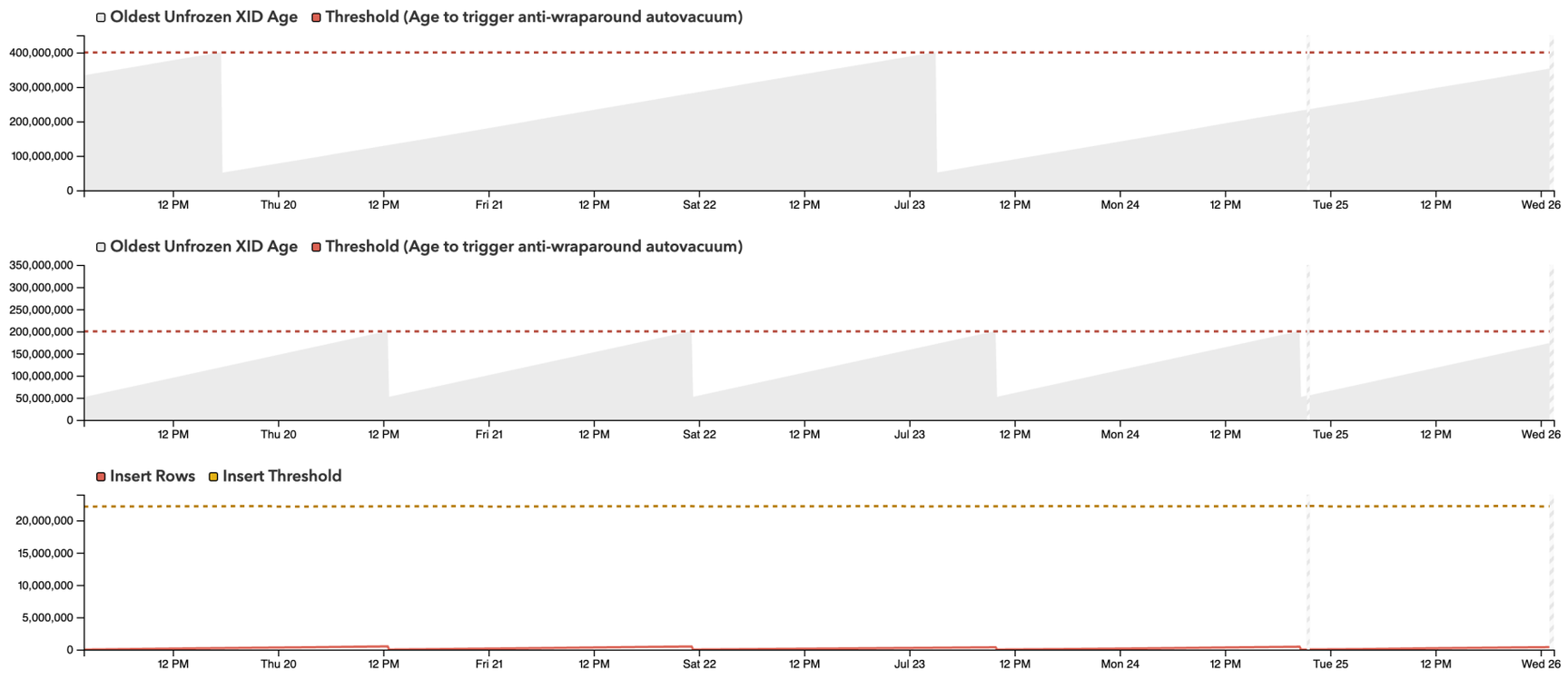
pganalyze

What's next? Looking ahead to upcoming features

The upcoming **VACUUM Simulator will show you how pganalyze insights work behind the scenes**

VACUUM Simulator — Reset Simulation Data

Total autovacuum count:	2	**Simulation autovacuum count:**	5 (dead rows: 0, freezing: 5, inserts: 0)
› **VACUUM triggered by: dead rows (bloat)**			
⌄ **VACUUM triggered by: freezing**			
Freeze Max Age:	400,000,000	**Freeze Max Age:**	200,000,000
Multixact Freeze Max Age:	400,000,000	**Multixact Freeze Max Age:**	400,000,000
Insert threshold:	1,000	**Insert threshold:**	1,000
Insert scale factor:	20.0%	**Insert scale factor:**	20.0%



pganalyze VACUUM Advisor will soon:

- Recommend cost limit / delay settings for long-running autovacuums
- Create insights to avoid backlogs due to insufficient worker capacity
- Track index bloat over time, to notify you when REINDEX is needed

We're also working on:

- Improved methods to optimize autovacuum frequency
- Insights for eager freezing to reduce I/O impact of autovacuum
- Automation features with approval workflows, to optionally have pganalyze run per-table settings changes on your behalf

thanks!

Get a free trial of pganalyze

PGANALYZE.COM

Get free pganalyze eBooks and Postgres blog posts

PGANALYZE.COM/RESOURCES **PGANALYZE.COM/BLOG**

PGANALYZE.COM/NEWSLETTER

We will send you an email with a recording of this webinar tomorrow!

Feel free to get in touch with us at pganalyze.com/contact