Diving into most common performance problems and how to fix them

Divya Sharma

Sr. RDS PostgreSQL Solutions Architect

Amazon Web Services

© 2023, Amazon Web Services, Inc. or its affiliates.

Agenda

- Memory Management and Checkpointing
- Storage and Data Management
- Replication
- Vacuum processing
- Query performance
- Upgrading
- PostgreSQL Happiness Hints

aws

© 2023, Amazon Web Services, Inc. or its affiliates.

Memory management

aws

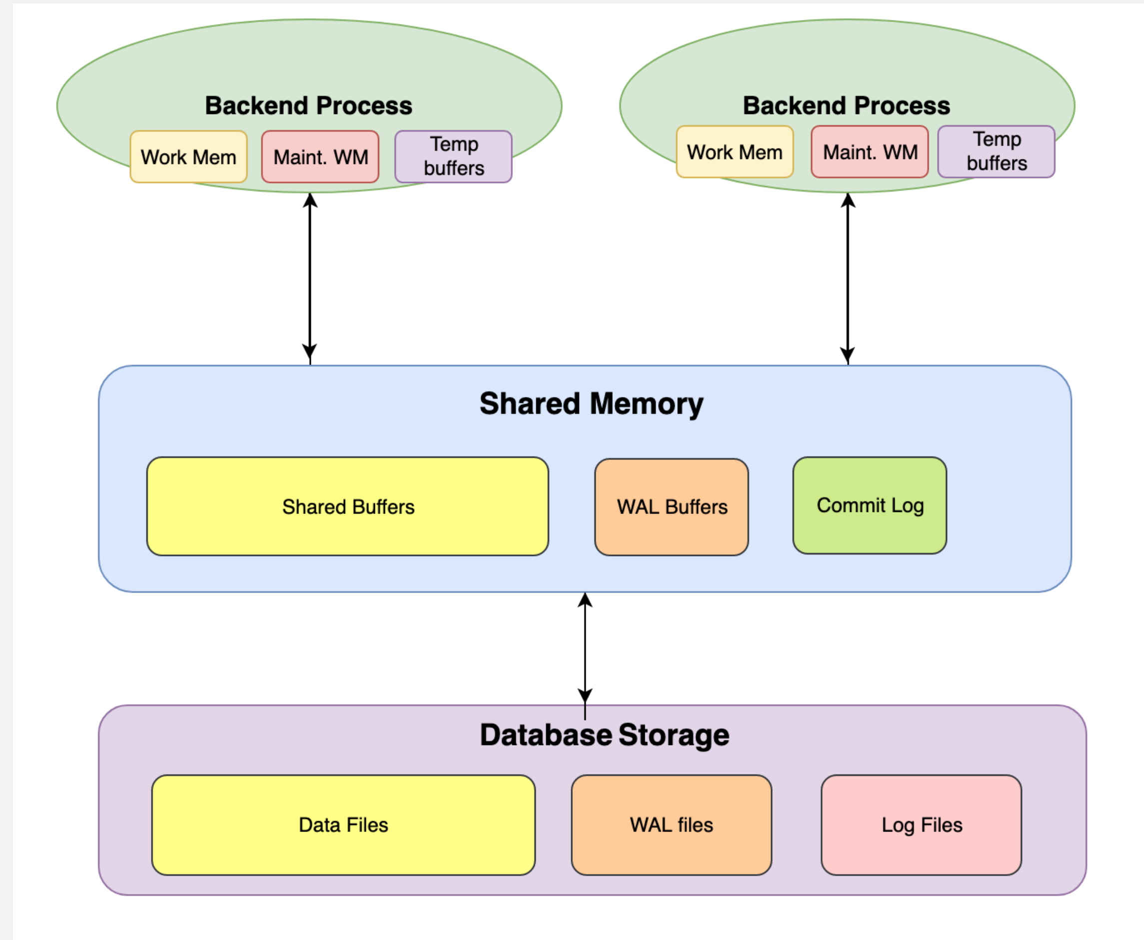
© 2023, Amazon Web Services, Inc. or its affiliates.

Customer questions

- How do we tune shared_buffers and work_mem?
- How do we pre-load a relation/index in the cache?
- Why do we see spikes in write IOPs/throughput at constant intervals?
- How to manage temporary files in PostgreSQL?

© 2023, Amazon Web Services, Inc. or its affiliates.

PostgreSQL Memory



© 2023, Amazon Web Services, Inc. or its affiliates.

Shared buffers tuning

- default value – 128MB (mostly)
- Reasonable setting/RDS PostgreSQL default - 25% of system memory
- Can try up to a value of 40%
- pg_buffercache - examining what's happening in the shared buffer cache in real time.
- pg_prewarm – load data into the OS cache or shared buffers

© 2023, Amazon Web Services, Inc. or its affiliates.

pg_buffercache and pg_prewarm

```
postgres=> select * from pg_extension where extname='pg_buffercache' OR extname='pg_prewarm';
  oid  |    extname     | extowner | extnamespace | extrelocatable | extversion | extconfig | extcondition
-------+----------------+----------+--------------+----------------+------------+-----------+--------------
 16403 | pg_buffercache |       10 |        16400 | t              | 1.3        |           |
 24836 | pg_prewarm     |       10 |        16400 | t              | 1.2        |           |
(2 rows)
```

```
postgres=> SELECT n.nspname, c.relname, count(*) AS buffers
             FROM pg_buffercache b JOIN pg_class c
             ON b.relfilenode = pg_relation_filenode(c.oid) AND
                b.reldatabase IN (0, (SELECT oid FROM pg_database
                                      WHERE datname = current_database()))
             JOIN pg_namespace n ON n.oid = c.relnamespace where nspname='public'
             GROUP BY n.nspname, c.relname ORDER BY buffers desc LIMIT 5;
 nspname |     relname      | buffers
---------+------------------+---------
 public  | rctest           |  211219
 public  | test             |   27038
 public  | emp              |       5
 public  | pgbench_history  |       4
 public  | pgbench_tellers  |       4
(5 rows)
```

© 2023, Amazon Web Services, Inc. or its affiliates.

pg_buffercache and pg_prewarm

```
postgres=> SELECT n.nspname, c.relname, count(*) AS buffers
             FROM pg_buffercache b JOIN pg_class c
             ON b.relfilenode = pg_relation_filenode(c.oid) AND
                b.reldatabase IN (0, (SELECT oid FROM pg_database
                                      WHERE datname = current_database()))
             JOIN pg_namespace n ON n.oid = c.relnamespace where nspname='public'
             GROUP BY n.nspname, c.relname ORDER BY buffers desc LIMIT 5;
 nspname |     relname     | buffers
---------+-----------------+---------
 public  | rctest          |  211219
 public  | test            |   27038
 public  | emp             |       5
 public  | pgbench_history |       4
 public  | pgbench_tellers |       4
(5 rows)
```

```
postgres=> SELECT pg_prewarm('rctest');
 pg_prewarm
------------
    1409825
(1 row)
```

```
postgres=> SELECT n.nspname, c.relname, count(*) AS buffers
             FROM pg_buffercache b JOIN pg_class c
             ON b.relfilenode = pg_relation_filenode(c.oid) AND
                b.reldatabase IN (0, (SELECT oid FROM pg_database
                                      WHERE datname = current_database()))
             JOIN pg_namespace n ON n.oid = c.relnamespace where nspname='public'
             GROUP BY n.nspname, c.relname ORDER BY buffers desc LIMIT 5;
 nspname | relname | buffers
---------+---------+---------
 public  | rctest  |  239121
(1 row)
```

© 2023, Amazon Web Services, Inc. or its affiliates.

Work Memory (work_mem)

- The working memory available for work operations (sorts, hash tables, joins).
- Depends on the the number of “work nodes” per query.
- Set reasonable amount globally.
- Use per database/user/session level for aggressive allocation
- work_mem and hash_mem_multiplier

© 2023, Amazon Web Services, Inc. or its affiliates.

Managing Temporary Files

- When the work_mem is not sufficient, temporary files are created to store the results.
- Written to disk and automatically removed after the query completes.

© 2023, Amazon Web Services, Inc. or its affiliates.

Managing Temporary files

- log_temp_files

```
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG: temporary
file: path "base/pgsql_tmp/pgsql_tmp31236.5", size 140353536
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT: select
a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by a.bid limit 10;


2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG: temporary
file: path "base/pgsql_tmp/pgsql_tmp31236.4", size 180428800
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT: select
a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by a.bid limit 10;
```

aws

© 2023, Amazon Web Services, Inc. or its affiliates.

Managing Temporary files

- temp_file_limit - cancels any query exceeding the size of temp_files in KB

```
postgres=> select * from pgbench_accounts, pg_class, big_table;
.
.
.
ERROR: temporary file size exceeds temp_file_limit (64kB)
```

© 2023, Amazon Web Services, Inc. or its affiliates.

Managing Temporary files

- pg_ls_tmpdir() function

```
postgres=> select replace(left(name, strpos(name, '.')-
1),'pgsql_tmp','') as pid, count(*), sum(size) from pg_ls_tmpdir() group
by pid;


pid  | count | sum
-----+--------------------
8355 |     2 | 2144501760
8351 |     2 | 2090770432
8327 |     1 | 1072250880
8328 |     2 | 2144501760
(4 rows)
```

Managing Temporary files

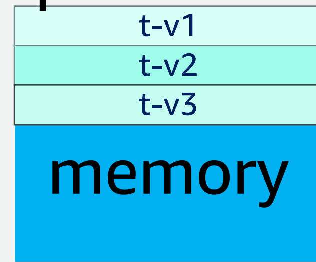
- pg_stat_statement - temp_blks_read, temp_blks_written
- EXPLAIN (ANALYZE,BUFFERS)

```
GroupAggregate  (cost=17612.84..19769.68 rows=107842 width=40) (actual time=861.091..884.817 rows=521 loops=1)
  Group Key: (st_geohash(geometry, 2))
  ->  Sort  (cost=17612.84..17882.44 rows=107842 width=32) (actual time=861.084..872.597 rows=107842 loops=1)
        Sort Key: (st_geohash(geometry, 2))
        Sort Method: external merge  Disk: 1376kB
        ->  Seq Scan on plan_item  (cost=0.00..6015.02 rows=107842 width=32) (actual time=0.018..50.245 rows=107842 loops=1)
Planning time: 0.094 ms
Execution time: 891.762 ms
```

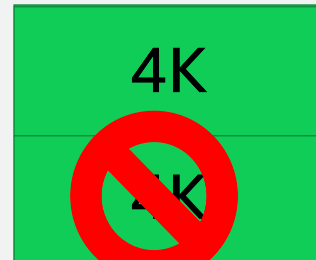
© 2023, Amazon Web Services, Inc. or its affiliates.

Checkpointing and FPW

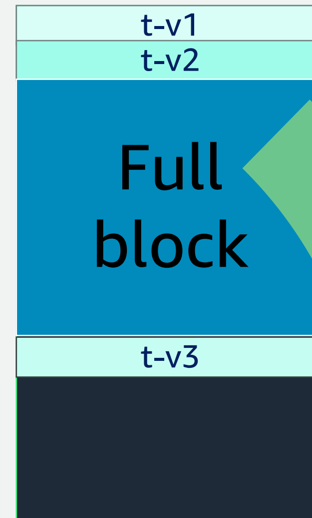
update t set y = 6



Checkpoint

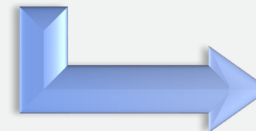


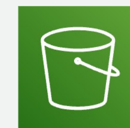
Datafile



WAL

Archive





Archive Location

© 2023, Amazon Web Services, Inc. or its affiliates.

Checkpointing

- Checkpointing - every "***checkpoint_timeout***" seconds, or if "***max_wal_size***" is about to be exceeded, whichever comes first
- causes an I/O load
- "***full_page_writes***"
- Recovery impact

WAL option with **EXPLAIN** now can be used to see WAL record generation including Full Page Images (fpi). This option can only be used along with **ANALYZE**.

© 2023, Amazon Web Services, Inc. or its affiliates.

Checkpointing process

```
2023-06-24 18:45:03 UTC::@:[377]:LOG: checkpoint starting: time
2023-06-24 18:45:03 UTC::@:[377]:LOG: checkpoint complete: wrote 2 buffers (0.0%); 0 WAL file(s) added, 0 removed, 1 recycled; write=0.105 s, sync=0.003 s, total=0.118 s; sync files=2, longest=0.002 s, average=0.002 s; distance=65536 kB, estimate=65536 kB


2023-06-24 18:45:16 UTC::@:[377]:LOG: checkpoint starting: immediate force wait
2023-06-24 18:45:16 UTC::@:[377]:LOG: checkpoint complete: wrote 0 buffers (0.0%); 0 WAL file(s) added, 0 removed, 0 recycled; write=0.004 s, sync=0.001 s, total=0.012 s; sync files=0, longest=0.000 s, average=0.000 s; distance=0 kB, estimate=58982 kB
```

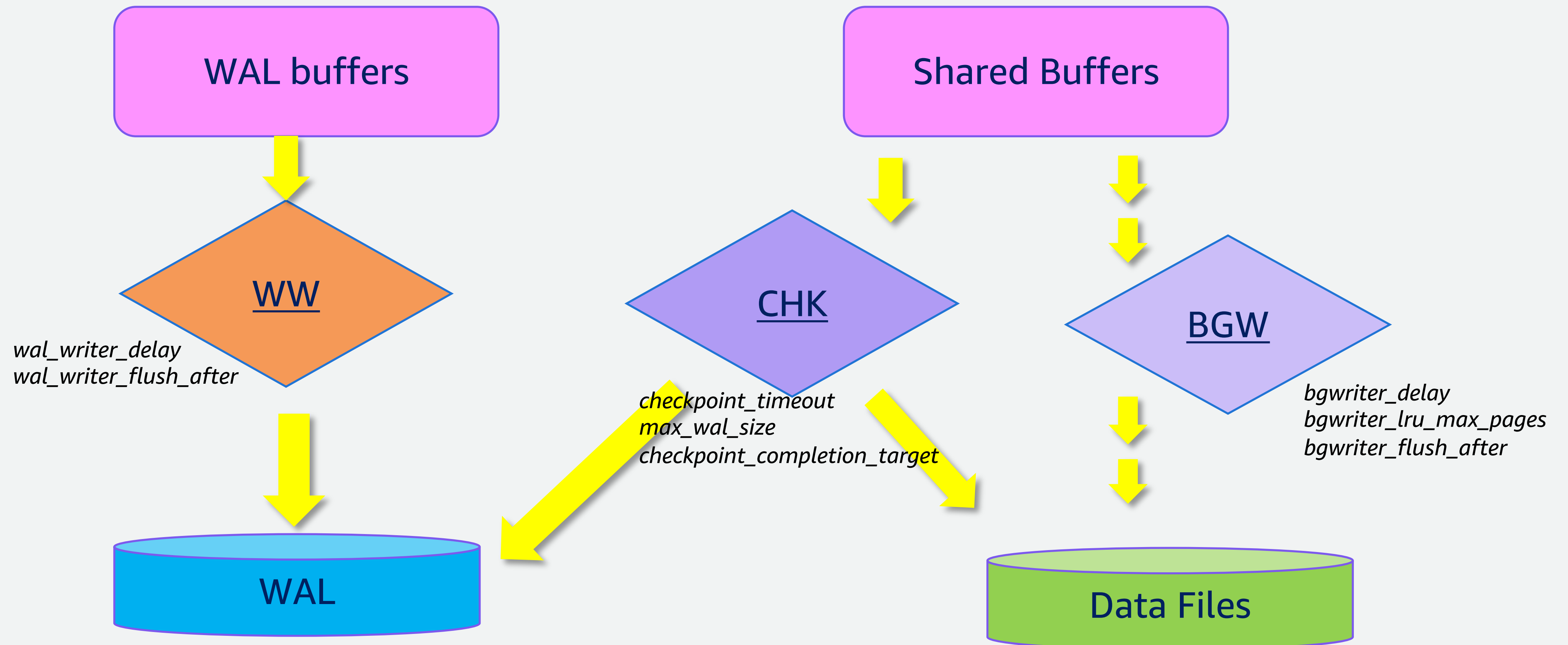

© 2023, Amazon Web Services, Inc. or its affiliates.

Checkpointing

```
postgres=> select * from pg_stat_bgwriter;
-[ RECORD 1 ]---------+------------------------------
checkpoints_timed     | 24251
checkpoints_req       | 25
checkpoint_write_time | 4615057
checkpoint_sync_time  | 101547
buffers_checkpoint    | 808661
buffers_clean         | 513394
maxwritten_clean      | 5130
buffers_backend       | 3898039
buffers_backend_fsync | 0
buffers_alloc         | 3347563
stats_reset           | 2023-03-25 17:41:58.47806+00
```


© 2023, Amazon Web Services, Inc. or its affiliates.

Auxiliary processes – WAL writer, Checkpointer, BG writer



aws

© 2023, Amazon Web Services, Inc. or its affiliates.

Storage and Data Management

aws

© 2023, Amazon Web Services, Inc. or its affiliates.

Customer questions

- How to manage increasing load on our production instance/server?
- Our main production table is growing too much and performance is degrading, what should we do?
- What is the impact of using temporary tables?

© 2023, Amazon Web Services, Inc. or its affiliates.

Storage and Data Management

- Splitting reads and writes
- Sharding
- Partitioning
- Selective Archival of Data

© 2023, Amazon Web Services, Inc. or its affiliates.

Storage and Data Management - Splitting reads and writes

- Offloading reads reduces overall load on the primary server, leaving more resources for write workload
- Target the list of “candidate queries” to move to readers - not dependent on immediate “read after write” consistency.
- Take into consideration the replica lag when moving read queries to a replica – not to have stale data

aws

© 2023, Amazon Web Services, Inc. or its affiliates.

Storage and Data Management - Sharding

- Storing a large database across multiple servers
- Improved response time
- Avoid total service outage
- Scale efficiently
- Eg. – Range based sharding, Geo sharding etc.

© 2023, Amazon Web Services, Inc. or its affiliates.

Storage and Data Management - Partitioning

- Provides for faster queries on large tables
- Partition Pruning – less I/O - **partition key must be used in the WHERE clause**
- Know your workload patterns in advance to design better from the start
- Large number of partitions - server's memory consumption may grow significantly over time
- Dropping partitions – avoiding table bloat
- **Never just assume that more partitions are better than fewer partitions, nor vice-versa.**

© 2023, Amazon Web Services, Inc. or its affiliates.

Storage and Data Management – Selective archival of data

- Segregate historical data from live data, for example using a live table and an archive table in the same database

- Use partitions to move data from the recent dataset - detaching a partition from the recent table and attaching it to the old table

- Move old data to another “archive” storage (Eg. Amazon S3 – cheaper than RDS)

- Keep only live data in the database

aws

© 2023, Amazon Web Services, Inc. or its affiliates.

Temporary tables

- Automatically dropped at the end of a session, or optionally at the end of the current transaction
- Indexes created on a temporary table are automatically temporary as well.
- The autovacuum daemon cannot access and therefore cannot vacuum or analyze temporary tables.
- **Too often creation of temporary tables is likely to cause bloating of catalog tables (pg_class, pg_attribute etc.)**

aws

© 2023, Amazon Web Services, Inc. or its affiliates.

Managing Temporary tables

- If possible, reduce recreating of temporary tables.
- Analyze PostgreSQL log and look for CREATE TEMP TABLE
- Setup monitoring on catalog tables
- If vacuum is not able to clean the catalog tables **all query processing slows down**
- Using CTEs where possible

© 2023, Amazon Web Services, Inc. or its affiliates.

Replication

© 2023, Amazon Web Services, Inc. or its affiliates.

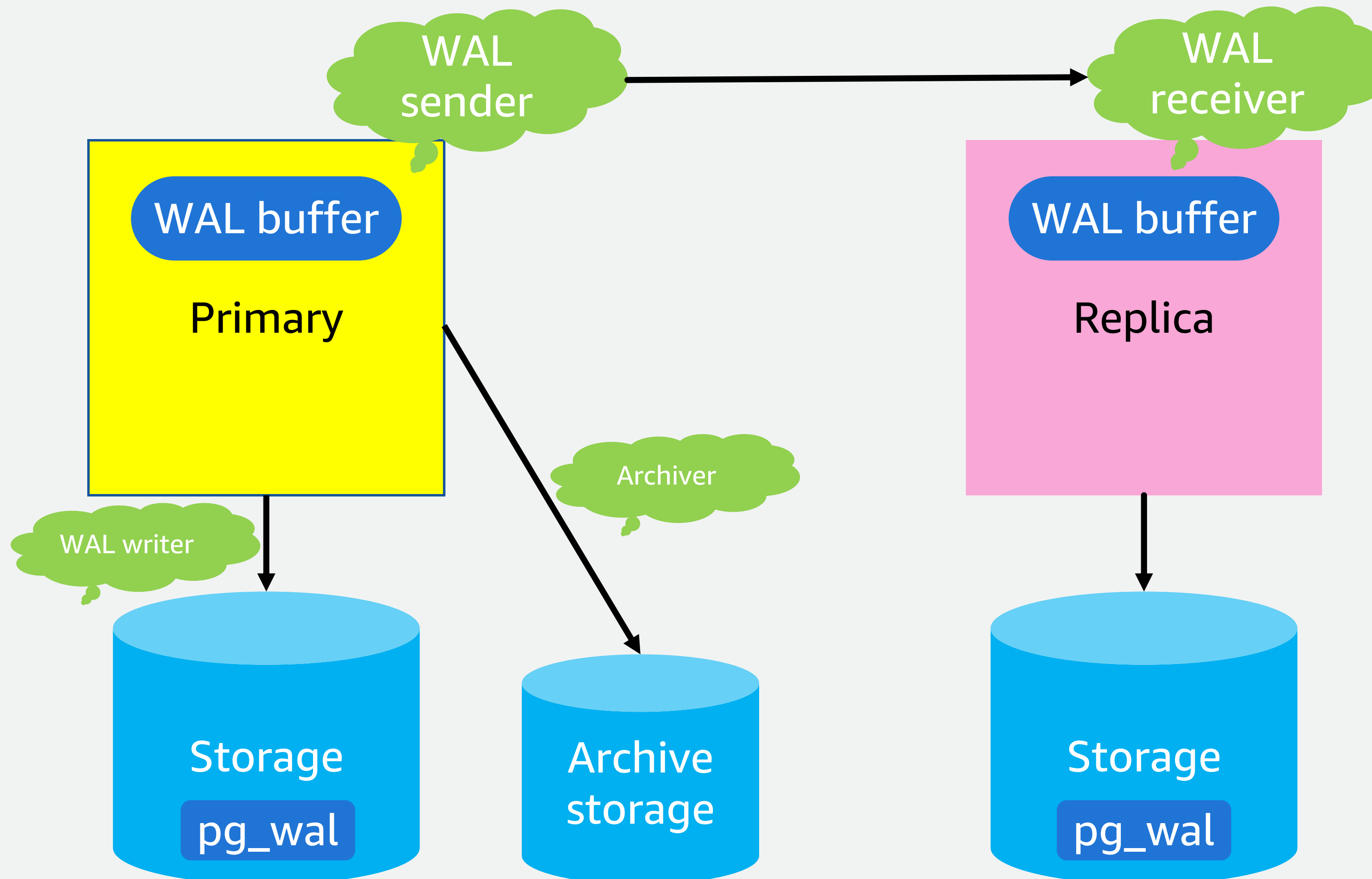
Customer questions

- How to deal with replica lag?
- Relation between vacuum and replication process?
- How to manage query cancelations on replicas?

aws

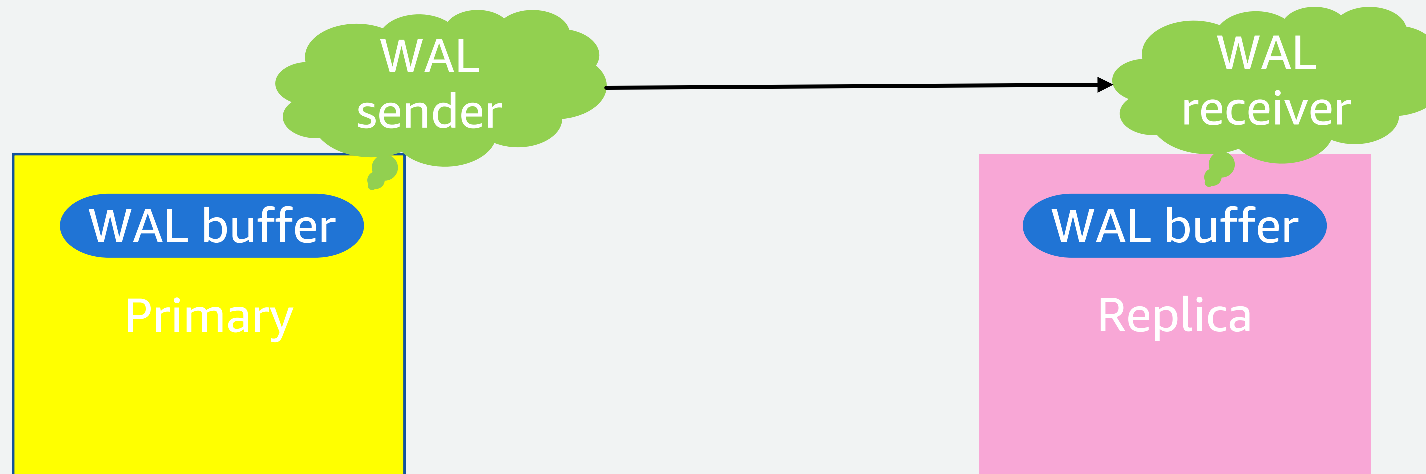
© 2023, Amazon Web Services, Inc. or its affiliates.

Physical Streaming Replication Basics



aws
© 2023, Amazon Web Services, Inc. or its affiliates.

Recovery Conflicts And Replication Lag



- *Auto commit is ON*
- *Synchronous commit is ON*

Time	Primary	Replica
t0	-	Reading from "A"
t1	WAL generating operation on "A"	Reading from "A"
t2	WAL generating operation completed	Still Reading from "A"
t4	-	Cancelling statement due to conflict with recovery

© 2023, Amazon Web Services, Inc. or its affiliates.

What causes recovery conflicts? – most common reasons

- Access Exclusive locks taken on the primary server
- Vacuum cleaning up records on primary and sending same info in WAL to standby - standby snapshots can still “see” some rows which are to be removed.
- Application of WAL on standby waiting due to long running on standby as it is still seeing old snapshot of data.

aws

© 2023, Amazon Web Services, Inc. or its affiliates.

Checking what conflicts are occurring

pg_stat_database_conflicts

```
postgres=> select * from pg_stat_database_conflicts where
datname='postgres';
-[ RECORD 1 ]----+----------
datid            | 14717
datname          | postgres
confl_tablespace | 0
confl_lock       | 1
confl_snapshot   | 5
confl_bufferpin  | 0
confl_deadlock   | 0
```

© 2023, Amazon Web Services, Inc. or its affiliates.

Important parameters

- For “Delaying” conflicts with recovery:
  max_standby_streaming_delay (on standby)
  max_standby_archive_delay (on standby)
- For “Avoiding” conflicts due to vacuum on primary:
  hot_standby_feedback (on standby)

© 2023, Amazon Web Services, Inc. or its affiliates.

Other factors for replication lag

- Network issue
- Server bottleneck – compute/storage
- wal_keep_segments
- New replica? – check logs to see if recovery is from archive location

aws

© 2023, Amazon Web Services, Inc. or its affiliates.

Vacuum and Autovacuum

aws
© 2023, Amazon Web Services, Inc. or its affiliates.

Customer Questions

- Why does a select from a table take a lot of time?
- Why are index scans taking time?
- Why does a table with only few rows occupying a lot of space?
- Why is vacuum taking time?
- Why is autovacuum not running/slow running?
- When do I need a manual vacuum?

aws

© 2023, Amazon Web Services, Inc. or its affiliates.

Vacuum and Autovacuum

- Vacuum is a SQL command that performs certain maintenance operations
- Autovacuum processes run the “Vacuum” and/or “Analyze” command based on certain thresholds OR when the system is approaching towards “Transaction ID Wraparound”

Autovacuum – automatic execution of vacuum and analyze

- When the number of dead tuples generated since the last VACUUM exceeds the "vacuum threshold"

```
vacuum threshold = vacuum base threshold + vacuum scale factor * number of tuples
```

where the vacuum base threshold is autovacuum_vacuum_threshold, the vacuum scale factor is autovacuum_vacuum_scale_factor, and the number of tuples is `pg_class.reltuples`.

- When the number of tuples inserted since the last vacuum has exceeded the defined insert threshold (**v13+**)

```
vacuum insert threshold = vacuum base insert threshold + vacuum insert scale factor * number of tuples
```

where the vacuum insert base threshold is autovacuum_vacuum_insert_threshold, and vacuum insert scale factor is autovacuum_vacuum_insert_scale_factor.

© 2023, Amazon Web Services, Inc. or its affiliates.

Autovacuum – automatic execution of vacuum and analyze

- When the total number of tuples inserted, updated, or deleted since the last ANALYZE exceeds the "analyze threshold"

```
analyze threshold = analyze base threshold + analyze scale factor * number of tuples
```

aws

© 2023, Amazon Web Services, Inc. or its affiliates.

Vacuum Best practices

- In order to remember the tuples maintenance_work_mem is used, make sure you have enough of it!
- The more the indexes, the more time it will take, irrespective of the size of the indexes. Therefore, check and drop unused indexes (pg_stat_user_indexes).
- Vacuum cleans both tables and associated indexes, however, this causes bloat in the indexes. Therefore, it might be useful to re-index to remove index bloat.

© 2023, Amazon Web Services, Inc. or its affiliates.

Vacuum Best practices

- Vacuum full is not recommended unless absolutely needed – Try CTAS instead of deleting a major chunk of data OR partitioning approach for managing data (discussed earlier)
- autovacuum_vacuum_cost_limit is divided amongst autovacuum_max_workers – so increase them both if you need to.
- Know when to run manual vacuum
- Removes only internal fragmentation and not external.

aws

© 2023, Amazon Web Services, Inc. or its affiliates.

Troubleshooting autovacuum not running/slow running

- Is autovacuum threshold met?
- Is ‘autovacuum’ set to ’off’ in the parameter group?
- Is autovacuum disabled for the relation?
- Any bottlenecks observed on compute/storage level when AV is running? – pg_stat_progress_vacuum

aws

© 2023, Amazon Web Services, Inc. or its affiliates.

Troubleshooting autovacuum not running/slow running

- Are there any open/long running transactions blocking AV to run? – *pg_stat_activity view ; idle_in_transaction_session_timeout parameter*
- Any locks conflicting with AV, taken by another transaction on the same resource? – *pg_stat_activity and pg_locks view*
- Is hot_standby_feedback enabled on the replica?
- Any open prepared transactions? - *max_prepared_transactions parameter ; pg_prepared_xacts view*

aws

© 2023, Amazon Web Services, Inc. or its affiliates.

Query Tuning

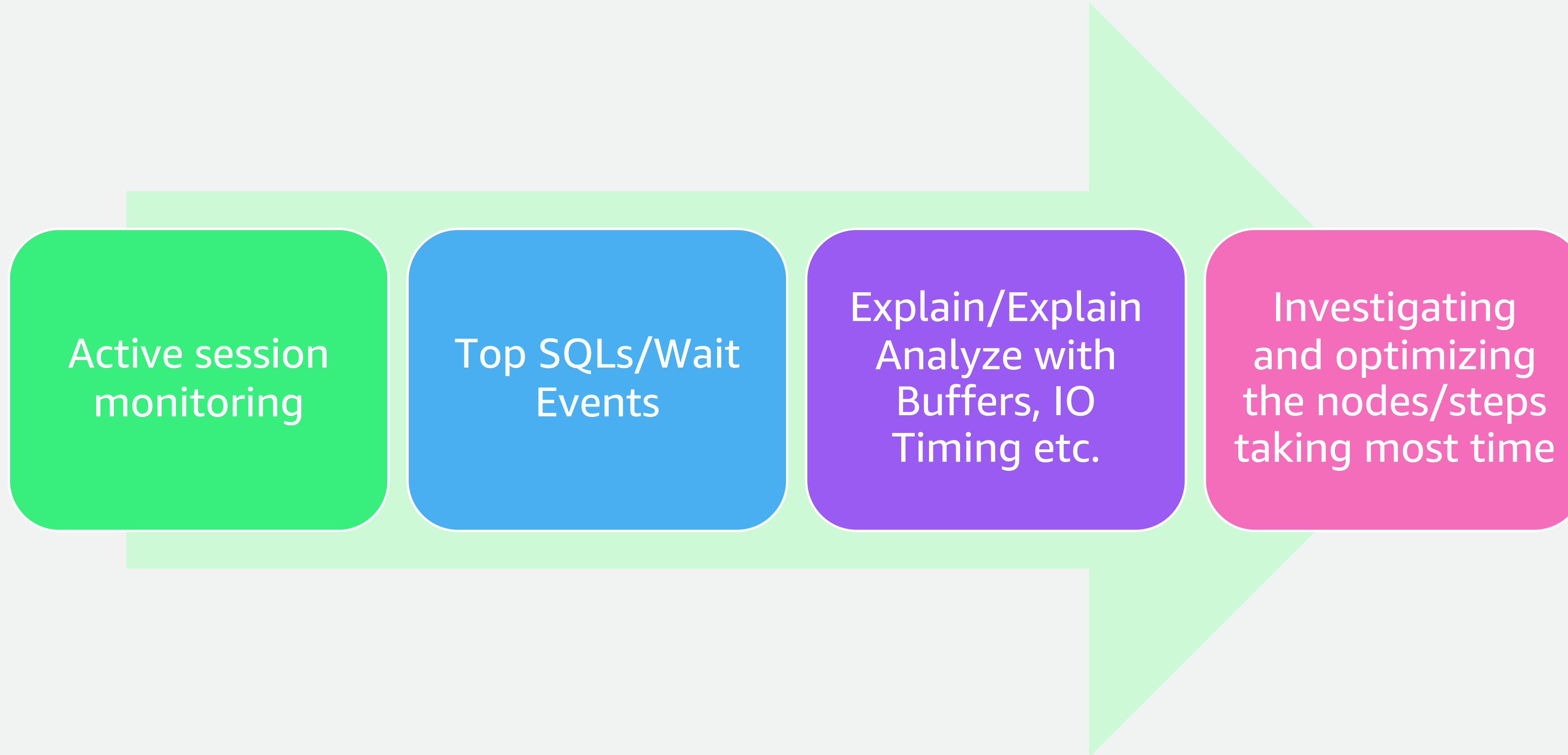
© 2023, Amazon Web Services, Inc. or its affiliates.

Customer Questions

- How do we identify slow queries?
- How to identify which queries to be tuned?
- What are the things to look for in an EXPLAIN plan?
- What are some techniques for query tuning?

© 2023, Amazon Web Services, Inc. or its affiliates.

Query tuning Methodology



aws
© 2023, Amazon Web Services, Inc. or its affiliates.

Active Session Monitoring

pg_stat_activity : One row per server process showing information related to the current activity of that process

```
postgres=> select pid,query,state,wait_event_type, wait_event from pg_stat_activity;
  pid  |                                       query                                        | state  | wait_event_type |     wait_event
-------+------------------------------------------------------------------------------------+--------+-----------------+---------------------
   381 |                                                                                    |        | Activity        | AutoVacuumMain
   383 |                                                                                    |        | Activity        | LogicalLauncherMain
 30924 | SELECT value FROM rds_heartbeat2                                                   | idle   | Client          | ClientRead
   394 | START_REPLICATION SLOT "rds_eu_west_1_db_l6m3njkhwqhgbpfvpo5w7i2lhu" 239/48000000 TIMELINE 1 | active | Activity        | WalSenderMain
 30969 | COMMIT                                                                             | idle   | Client          | ClientRead
 11641 | select pid,query,state,wait_event_type, wait_event from pg_stat_activity;         | active |                 |
   378 |                                                                                    |        | Activity        | BgWriterHibernate
   382 |                                                                                    |        | Activity        | ArchiverMain
   377 |                                                                                    |        | Activity        | CheckpointerMain
   380 |                                                                                    |        | Activity        | WalWriterMain
(10 rows)
```

- Check if the query is blocked by joining **pg_stat_activity** with **pg_locks**
- Monitor and understand wait events
- **idle_session_timeout**
- **idle_in_transaction_session_timeout**

© 2023, Amazon Web Services, Inc. or its affiliates.

Understanding Explain

EXPLAIN (Analyze, Buffers)

```
GroupAggregate  (cost=17612.84..19769.68 rows=107842 width=40) (actual time=861.091..884.817 rows=521 loops=1)
  Group Key: (st_geohash(geometry, 2))
  ->  Sort  (cost=17612.84..17882.44 rows=107842 width=32) (actual time=861.084..872.597 rows=107842 loops=1)
        Sort Key: (st_geohash(geometry, 2))
        Sort Method: external merge  Disk: 1376kB
        ->  Seq Scan on plan_item  (cost=0.00..6015.02 rows=107842 width=32) (actual time=0.018..50.245 rows=107842 loops=1)
Planning time: 0.094 ms
Execution time: 891.762 ms
```

- If using ANALYZE, run inside transaction block for DMLs
- Actual vs Estimated Rows
- Execution Time

© 2023, Amazon Web Services, Inc. or its affiliates.

Understanding EXPLAIN

```
postgres=*> explain (analyze,buffers,wal) Update rctest set num1=8 where num1<4;
                                                                 QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 Update on rctest  (cost=0.00..3775020.80 rows=0 width=0) (actual time=569674.180..569674.181 rows=0 loops=1)
   Buffers: shared hit=201553414 read=2728750 dirtied=2275195 written=659534
   I/O Timings: shared/local read=228071.556 write=3986.989
   WAL: records=128452589 fpi=2012384 bytes=15585451334
   ->  Seq Scan on rctest  (cost=0.00..3775020.80 rows=73579665 width=14) (actual time=0.008..153568.843 rows=69998254 loops=1)
         Filter: (num1 < 4)
         Rows Removed by Filter: 109998989
         Buffers: shared hit=27660 read=1382165 written=142592
         I/O Timings: shared/local read=119685.580 write=1266.970
 Planning:
   Buffers: shared hit=3
 Planning Time: 0.056 ms
 Execution Time: 569675.333 ms
(13 rows)
```

- Large no. of buffers read (not hit)
- Scan method – Seq scan vs indexes?
- Full Page images
- Bitmap heap scan reporting lossy

© 2023, Amazon Web Services, Inc. or its affiliates.

Actions to take after slow query investigation

- Collect more statistics (default_statistics_target) or extended statistics
- Modify relevant parameters – work_mem ; maintenance_work_mem etc.
- Fix the query plan, if needed using pg_hint_plan
- Add relevant indexes and drop unused ones
- Reduce planning overhead by using prepared statement (*monitor to not cause wraparound!*)
- Implement or change table partitioning strategy
- Have another cache in front of the database (eg – ElastiCache in front of RDS)

© 2023, Amazon Web Services, Inc. or its affiliates.

Query Tuning Cycle

Monitoring Performance

Datadog
Grafana
AWS Cloudwatch, EM and PI for RDS/ Aurora

Find SQL Query for tuning

Logging paramters/Error Log
pg_stat_statement
auto_explain extension

Query Analysis

Explain
Visualize plan

Query Optimization

Collecting more statistics
Relevant indexes (create/drop)
pg_hint_plan
update parameters database/session/user level

© 2023, Amazon Web Services, Inc. or its affiliates.

Upgrading

© 2023, Amazon Web Services, Inc. or its affiliates.

Customer Questions

- Why to upgrade?
- When to upgrade?
- How to upgrade?

aws

© 2023, Amazon Web Services, Inc. or its affiliates.

Version Upgrades

- **Minor version upgrades**
  - Patches to the binaries
  - No new functionality
  - May contain important security fixes

- **Major version upgrades**
  - Tracks the community yearly release cycle
  - Introduces new functionality
  - May change system catalogs and page formats
  - Supports skip version in-place upgrade

© 2023, Amazon Web Services, Inc. or its affiliates.

Why should you upgrade?

why-upgrade.depesz.com/show?from=14.5&to=15.3&keywords=

source repo
author info
meta-info

Why upgrade PostgreSQL?

Upgrade from: 14.5 to: 15.3 matching: gives me ...

Upgrading from 14.5 to 15.3 gives you 9.1 months worth of fixes (351 of them)

Security fixes:

- Remove PUBLIC creation permission on the public schema (Noah Misch)
  The new default is one of the secure schema usage patterns that Section 5.9.6 has recommended since the security release for CVE-2018-1058. The change applies to new database clusters and to newly-created databases in existing clusters. Upgrading a cluster or restoring a database dump will preserve public's existing permissions. For existing databases, especially those having multiple users, consider revoking CREATE permission on the public schema to adopt this new default. For new databases having no need to defend against insider threats, granting CREATE permission will yield the behavior of prior releases.
- libpq can leak memory contents after GSSAPI transport encryption initiation fails (Jacob Champion)
  A modified server, or an unauthenticated man-in-the-middle, can send a not-zero-terminated error message during

Jump to:

- Security fixes
- Configuration changes
  - Removed
  - Added
  - Changed default value
- List of changes
- ... to 15 from 2022-10-13
- ... to 15.1 from 2022-11-10
- ... to 15.2 from 2023-02-09

© 2023, Amazon Web Services, Inc. or its affiliates.

Why should you upgrade?

postgresql.org/about/featurematrix/#replication

Replication

	16	15	14	13	12	11
ALTER SUBSCRIPTION ... SKIP	Yes	Yes	No	No	No	No
Cascading streaming replication	Yes	Yes	Yes	Yes	Yes	Yes
Configure max WAL retention for replication slots	Yes	Yes	Yes	Yes	No	No
Logical replication	Yes	Yes	Yes	Yes	Yes	Yes
Logical replication avoids replication loops	Yes	No	No	No	No	No
Logical replication column lists	Yes	Yes	No	No	No	No
Logical replication for partitioned tables	Yes	Yes	Yes	Yes	No	No
Logical replication from standbys	Yes	No	No	No	No	No

https://www.postgresql.org/docs/13/runtime-config-replication.html#GUC-MAX-SLOT-WAL-KEEP-SIZE

© 2023, Amazon Web Services, Inc. or its affiliates.

Things to remember for upgrades

- Update DB engine, when a new release becomes available (recommended) – A major version is depreciated every 5 years
- Set an appropriate maintenance window
- Read replica can have different **minor** version than primary – can be used for testing newer versions for **major** (promote and upgrade replica)
- Use pglogical or native logical replication for minimum downtime major version upgrade (alternatives to pg_upgrade)
- Test engine update process in a representative pre-prod environment
- Run ”Analyze” to make sure statistics are up-to-date for planners use
- A PostgreSQL engine upgrade doesn't upgrade most PostgreSQL extensions, make sure you upgrade them after upgrade (ALTER EXTENSION UPDATE).

© 2023, Amazon Web Services, Inc. or its affiliates.

PostgreSQL Happiness Hints

version: jer_s/2022-04-26

Checksums and Huge Pages Enabled

Connection Pooling

- Centralized (e.g. pgbouncer) and decentralized (e.g. JDBC) architectures
- Recycle server connections (e.g. server_lifetime)
- Limit or avoid dynamic growth when practical – queue at a tier above the DB

Default Limits: Temp Usage, Statement & Idle Transaction Timeout

- Timeouts 5-15 minutes or lower, increase at session level if needed

Scaling

- Measure conn count in hundreds (not thousands), table count in thousands (not hundreds of thousands), relation size in GB (not TB), indexes per table in single digits (not double digits)
- Higher ranges work, but often require budget for experienced & expensive PostgreSQL staff
- To scale workloads, shard across instances or carefully partition tables

Updates and Upgrades

- PostgreSQL quarterly stable "minors" = security and critical fixes only
  - On Aurora: minors can have new development work
- Before major version upgrade, compare plans and latencies of top SQL on upgraded test copy
- Remember to upgrade extensions; it's not automatic
- Stats/analyze after major version upgrade

Logging

- Minimum 1 month retention (on AWS: use max retention and publish to Cloudwatch)
- Log autovacuum minimum duration = 10 seconds or lower
- Log lock waits
- Log temp usage when close to the default limit
- On AWS: autovacuum force logging level = WARNING

Multiple Physical Data Centers (= Multi-AZ on AWS)

Physical Backups

- Minimum 1 month retention
- Regular restore testing

Logical Backups (at least one)

- Scheduled exports/dumps and redrive/replay
- Logical replication

Active Session Monitoring (= Performance Insights on AWS)

- Save snapshots of pg_stat_activity making sure to include wait events
- Keep historical data, minimum 1 month retention (hopefully much more)

SQL and Catalog and Other Database Statistics Monitoring

- Preload pg_stat_statements
- Save snapshots of pg_stat_statements and key statistics
  - Exec plans (eg. auto_explain or others), relation sizes (bytes & rows incl catalogs), unused indexes
  - Rates: tuple fetch & return, WAL record & fpi & byte, DDL, XID, subtransaction, multixact, conn
- Keep historical data, minimum 1 month retention (hopefully much more)

OS Monitoring (= Enhanced Monitoring on AWS)

- Granularity of 10 seconds or lower (1 second if possible)
- Keep historical data, minimum 1 month retention (hopefully much more)

Alarms

- **Average active sessions** (= dbload cloudwatch metric on AWS)
- Memory / swap
- Disk space: %space and %inodes (and free local storage on Aurora)
- Hot standby & logical replication lag / WAL size (disk space) on primary
- Unexpected errors in the logs, both database and application tier
- Maximum used transaction IDs (aka time to wraparound)
- Checkpoint: time since latest & warnings in log (doesn't apply to Aurora)

Thank you!

Divya Sharma

Sr. Database Specialist SA
Amazon Web Services

https://ie.linkedin.com/in/divyasharma95

© 2023, Amazon Web Services, Inc. or its affiliates.