POSTGRESQL Replication Conflicts: Avoiding Pitfalls

Hamid Quddus Akhtar

Percona

Keeping Open Source Open

PERCONA
Distribution for PostgreSQL

PERCONA
Distribution for MongoDB

PERCONA
Distribution for MySQL

PERCONA
Monitoring and Management

About Myself

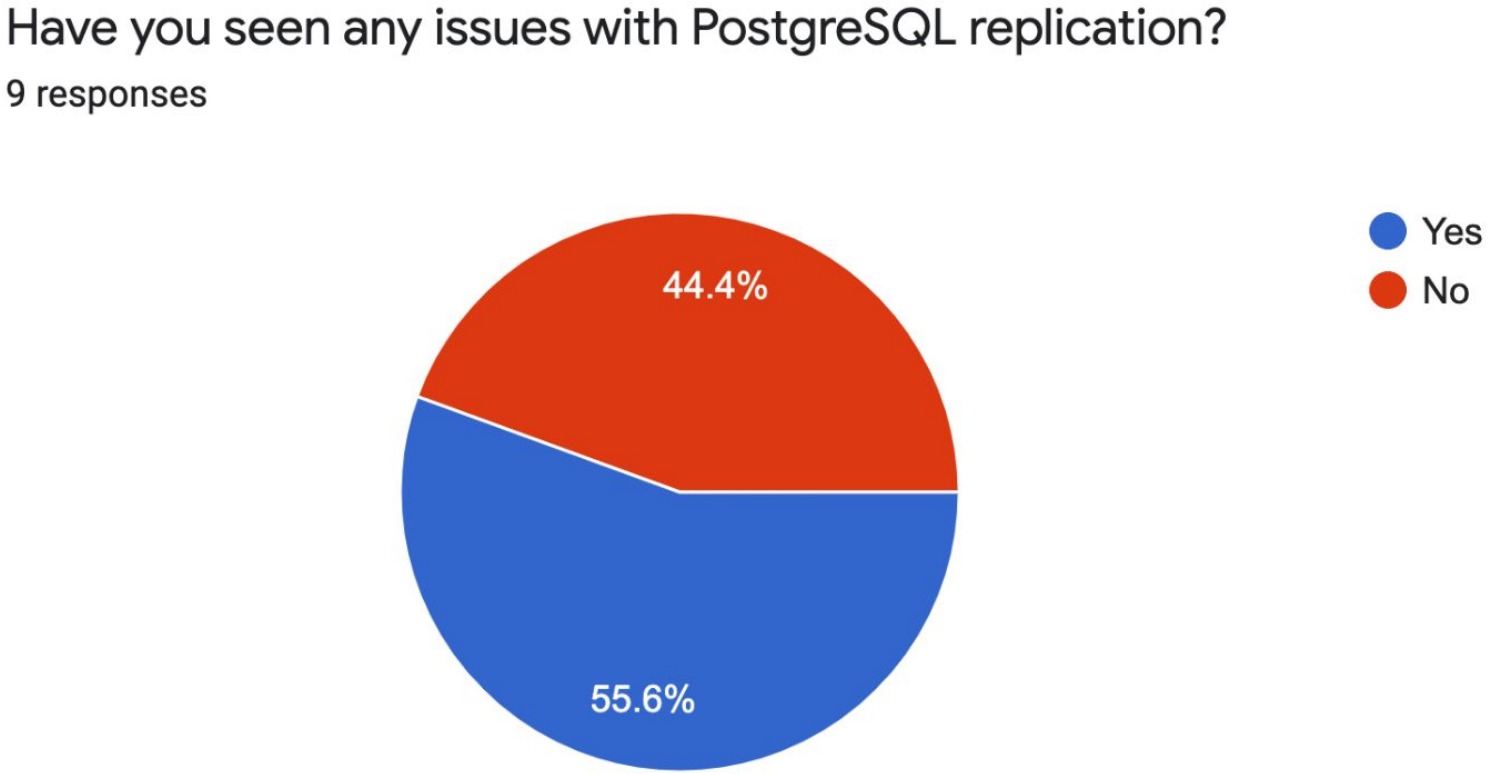
- More than two decades of professional software development.
- I’m part of Percona:
  - Percona has amazing culture.
- Prior to joining Percona, I had worked
  - HighGo, and
  - EnterpriseDB.

Contacts

- Email:
  - hamid.akhtar@percona.com
- LinkedIn:
  - https://www.linkedin.com/in/engineeredvirus/
- Skype:
  - EngineeredVirus
- WhatsApp
  -

Initial Thoughts

Replication Issues?

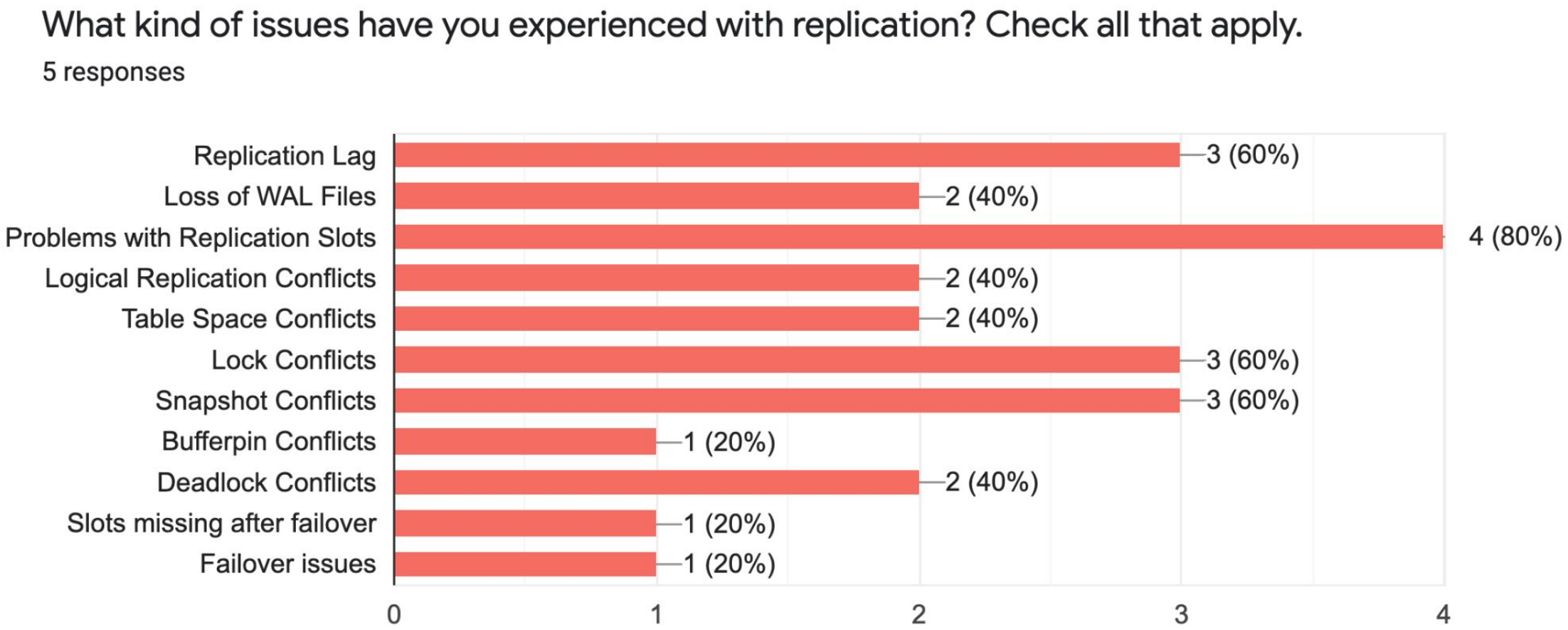


PERCONA

Replication Issues - Overview

- Reliable replication scheme requires:
  - Optimal configuration,
  - Query monitoring,
  - Analysis of relevant statistic view/relations,
  - Cool failover/switchover handling.
- And obviously, understanding and managing risks.

Types of Replication Issues Faced



PERCONA

What to Expect From This Talk

Presentation Outline

- Talk scope is limited to:
  - Synchronous streaming replication conflicts in context of pg_stat_database_conflicts
- What PostgreSQL Offers:
  - Catalogs and Views,
  - Configuration Parameters.
- Identification of Replication Conflicts by PostgreSQL Server

Replication Catalogs and Configuration

Replication: PostgreSQL Out of the Box

- Relevant System Catalogs and Views
  - [Primary] pg_stat_replication
  - [Standby] pg_stat_wal_receiver
  - [Standby] pg_stat_database_conflicts

Replication: PostgreSQL Out of the Box

- Configuration Parameters
  - Sending Server:
    - max_wal_senders, max_replication_slots, wal_keep_size, max_slot_wal_keep_size, wal_sender_timeout, track_commit_timestamp.
  - Primary Server:
    - synchronous_standby_names, vacuum_defer_cleanup_age
  - Standby Server:
    - primary_conninfo, primary_slot_name, promote_trigger_file, hot_standby, max_standby_archive_delay, wal_receiver_create_temp_slot, wal_receiver_status_interval, hot_standby_feedback, wal_receiver_timeout, wal_retrieve_retry_interval, recovery_min_apply_delay

Identification of Replication Conflicts by PostgreSQL Server

Replication: Configuration

- Server version 14.x built from source on CentOS 7
  - One primary, and
  - One standby.
- Let’s generate some database conflicts:
  - Lock conflict, and
  - Snapshot conflict.

Replication: Lock Conflict

Primary Server	Standby

`CREATE TABLE lock_test`
`AS`
`(SELECT generate_series(1, 100) AS id);`	

	`BEGIN;` `SELECT * FROM lock_test WHERE id < 5;`
`DROP TABLE lock_test;`	
	<Wait for max_standby_streaming_delay to expire>
	`SELECT * FROM lock_test WHERE id < 5;`

	`FATAL:  terminating connection due to conflict with recovery`
	`DETAIL:  User was holding a relation lock for too long.`
	`...`

Replication: Lock Conflict

Primary Server	Standby Server

	`SELECT datid, datname, confl_lock`
	`FROM pg_stat_database_conflicts`
	`WHERE datname = CURRENT_DATABASE();`

	`datid | datname | confl_lock`
	`-------+-----------+------------`
	`13892 | postgres | 1`

Lock Conflict - Code Flow

- Start XLog Processing
- Acquire Access Exclusive Lock(s)
  - Acquire session lock,
- Identify conflicting backends.
- Sleep on the Lock
  - Wait for other transactions to release it or for timeout to occur (GetStandbyLimitTime),
  - Lots and lots of checks here.
- Resolve Recovery Conflict With Lock
  - Issue signals to kill all backends causing the deadlock.

Lock Conflicts - GUCs and Data That Matter

- Use the available resources to set up health parameters:
  - pg_stat_activity and pg_locks:
    - Join on pid to identify connections that are holding locks:
    - pg_stat_activity provides timestamps for connection start, transaction start, and query start.
    - So transaction aging can be identified.
- [Standby] max_standby_streaming_delay
  - Consider max age for transactions and tune this parameter accordingly.

Replication: Snapshot Conflicts

Primary Server	Standby Server
`CREATE TABLE snap_test AS (SELECT generate_series(1, 100) AS id);`	

	`BEGIN;`
	`DECLARE c CURSOR FOR`
	`SELECT * FROM snap_test;`
	`FETCH c;`

`DELETE FROM snap_test WHERE id % 3 = 0;` `VACUUM snap_test;`	
	`FETCH c;`

	`FATAL: terminating connection due to conflict with recovery`
	`DETAIL: User query might have needed to see row versions that must be removed.`
	`...`

Replication: Snapshot Conflicts

Primary Server	Standby Server

	`SELECT datid, datname, confl_snapshot`
	`FROM pg_stat_database_conflicts`
	`WHERE datname = CURRENT_DATABASE();`

	`datid | datname | confl_snapshot`
	`-------+----------+----------------`
	`13892 | postgres | 1`

PERCONA

Snapshot Conflicts - Code Flow

- Start XLog Processing
- Prepare for performing required heap (access method) operation.
  - Heap cleanup in this particular case.
- Identify conflicting backends.
- Resolve Recovery Conflict With Snapshot
  - Wait for other transactions to release it or for timeout to occur (GetStandbyLimitTime),
  - Issue signals to kill all conflicting backends.

Snapshot Conflicts - GUCs That Matter

- [Primary] vacuum_defer_cleanup_age
  - Not honored if manual vacuum command is issued.
- [Standby] hot_standby_feedback
  - Eliminates query cancels because of cleanup operations.
  - Works with cascaded standbys.

Percona stands for evolution

Percona stands for ease-of-use

Percona stands for freedom

Percona & PostgreSQL - Better Together

PERCONA


Are you passionate about Open Source?!
We're hiring!
Join us!
#RemoteWork
percona.com/careers

Thank you!

Questions?