Myths and Truths about Synchronous Replication in PostgreSQL

Alexander Kukushkin

PGConf.DE 2025, Berlin

2025-05-09

About me

Alexander Kukushkin

Principal Software Engineer @Microsoft

The Patroni guy

akukushkin@microsoft.com

Write-Ahead Log (WAL)

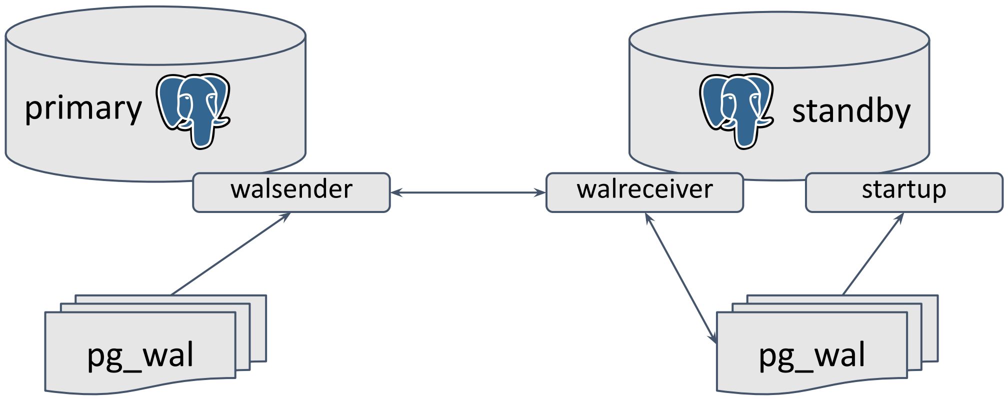
- A standard method for ensuring data integrity
- Used for recovery, archives, replication, etc…
- http://www.postgresql.org/docs/current/static/wal-intro.html

```
> ls -l pg_wal/
total 950276
-rw------- 1 akukushkin akukushkin 16777216 Jan  9 15:28 000000010000000000000001
-rw------- 1 akukushkin akukushkin 16777216 Jan  9 15:28 000000010000000000000002
-rw------- 1 akukushkin akukushkin 16777216 Jan  9 15:28 000000010000000000000003
-rw------- 1 akukushkin akukushkin 16777216 Jan  9 15:28 000000010000000000000004
-rw------- 1 akukushkin akukushkin 16777216 Jan  9 15:28 000000010000000000000005
-rw------- 1 akukushkin akukushkin 16777216 Jan  9 15:28 000000010000000000000006
-rw------- 1 akukushkin akukushkin 16777216 Jan  9 15:28 000000010000000000000007
-rw------- 1 akukushkin akukushkin 16777216 Jan  9 15:28 000000010000000000000008
-rw------- 1 akukushkin akukushkin 16777216 Jan  9 15:28 000000010000000000000009
-rw------- 1 akukushkin akukushkin 16777216 Jan  9 15:28 00000001000000000000000A
-rw------- 1 akukushkin akukushkin 16777216 Jan  9 15:28 00000001000000000000000B
-rw------- 1 akukushkin akukushkin 16777216 Jan  9 15:28 00000001000000000000000C
```

Replication

- Log-Shipping (Continuous Archiving and PITR)
  - archive_command / restore_command
- Streaming replication
  - **Physical replication**
  - Logical replication

Physical streaming replication



Streaming replication

- Asynchronous
  - default, primary doesn’t wait

- Synchronous
  - primary waits until standby(s) confirm that they wrote/flushed/applied commit WAL record
  - **synchronous_commit** = remote_write/on/remote_apply
  - **synchronous_standby_names** = 'my_standby'

synchronous_commit

value	local durable commit	standby durable commit after PG crash	standby durable commit after OS crash	standby query consistency
remote_apply	✅	✅	✅	✅
on	✅	✅	✅	
remote_write	✅	✅		
local	✅			
off				

Synchronous replication types

- **priority**
  - **synchronous_standby_names** = '**FIRST** 1 (node1, node2)'
  - waits for confirmation from **first nodes in the list**
  - if node1 failed, waits for node2
- **quorum**
  - **synchronous_standby_names** = '**ANY** 1 (node1, node2)'
  - waits for confirmation **from any node**

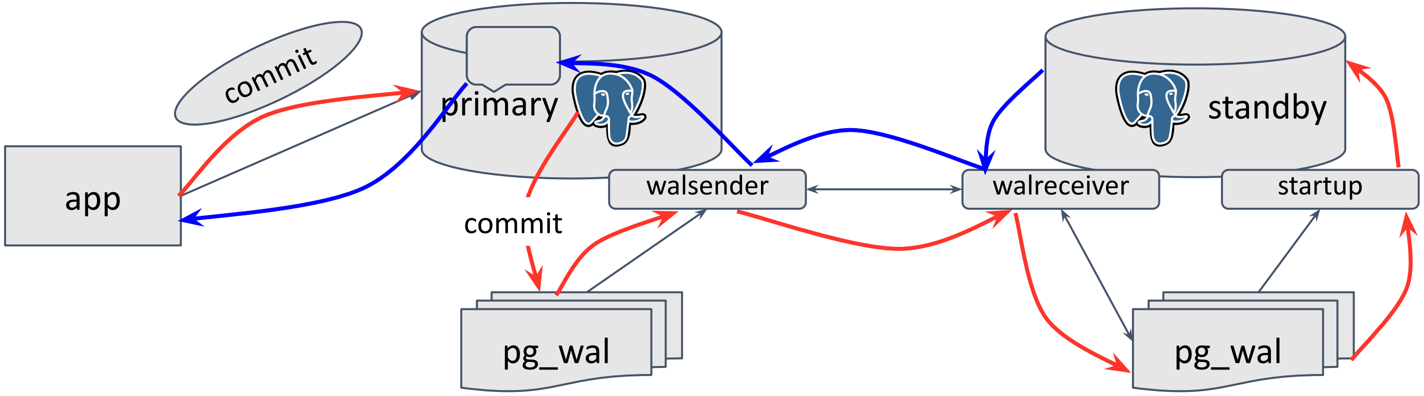
Myth №1

Transaction is committed after receiving confirmation from synchronous standby nodes.

Truth

- Transaction is always **committed locally first**!
- **Primary holds locks** until commit WAL record is confirmed to be received/flushed/applied by standby nodes
- **Locks are released** and transaction becomes visible **when sufficient number standby nodes confirmed**, **when query is cancelled, connection is broken, or Postgres is restarted**

synchronous_commit = remote_apply



Myth №2

Synchronous replication guarantees Zero Recovery Point Objective (RPO) / no data loss

Truth

- It depends!
- synchronous_commit = local could be set per connection
  - disables waiting for synchronous nodes
- transaction becomes visible when lock wait is cancelled:
  - Query cancellation
  - **TCP connection reset**
  - **Postgres restart**

```
postgres=# alter system set synchronous_standby_names = 'unknown';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)

postgres=# show synchronous_standby_names;
 synchronous_standby_names
---------------------------
 unknown
(1 row)

postgres=# show synchronous_commit;
 synchronous_commit
--------------------
 on
(1 row)

postgres=# create table test as select i from generate_series(0, 100) i;
^CCancel request sent
WARNING:  canceling wait for synchronous replication due to user request
DETAIL:  The transaction has already committed locally, but might not have been replicated to the standby.
SELECT 101
postgres=# select count(*) from test;
 count
-------
   101
(1 row)
```

Cancelled wait problem

- If wait is cancelled, transaction is immediately visible to other connections, even if it wasn't confirmed by standby nodes!
  - If primary fails there could be a visible data loss when synchronous standby is promoted.
- Postgres should disallow cancellation of wait for sync replication. Discussion on #pgsql-hackers

Cancelled wait problem (continue)

- If TCP connection is interrupted application doesn’t know whether transaction was committed or not!
- Finding transaction state (e.g. before retrying)
    - Two Phase Commit (2PC)
    - txid_status(bigint) function -> committed, aborted, in progress, or null

txid_status()

```
postgres=# begin;
    create table test as select i from generate_series(0, 100) i;
    select txid_current();
commit;
BEGIN
SELECT 101
 txid_current
--------------
          764
(1 row)

Killed
$ psql -U postgres -h localhost
psql (17.2 (Ubuntu 17.2-1.pgdg22.04+1))
Type "help" for help.

postgres=# select txid_status(764);
 txid_status
-------------
 committed
(1 row)
```

```
postgres=# select pid, query, wait_event_type, wait_event
from pg_stat_activity where backend_xid = 764;
   pid    |  query   | wait_event_type | wait_event
---------+---------+-----------------+------------
 1029064 | commit; | IPC             | SyncRep
(1 row)
```

Myth №3

Reading from sync standby nodes is like reading from the primary.

Truth

- Not entirely!
- transaction on standby is immediately visible
  - primary could be still waiting for more standby nodes to confirm!
- Never do write based on read from standby!

Side effects

- Asynchronous standby nodes can be ahead of sync nodes
- Logical replication connections as well
    - Logical failover slots (PG17) or pg_failover_slots extension help to mitigate it.
- Quorum-based synchronous replication
    - we don’t know which nodes confirmed transaction!

Read from standby after write to primary

- **synchronous_standby_names = 'N (node1, …, nodeN)'**

- wait (in a loop) until standby caught up:
    - pg_current_wal_insert_lsn() + pg_last_wal_replay_lsn()
    - txid_current() + txid_status()

- Future work: Wait for LSN replay function

Myth №4

We just need to promote synchronous replica to avoid data loss

Truth

- Yes. But…
- Let’s assume we have a node1 (primary), and node2 (async standby)
- we set synchronous_standby_names = 'node2'
- SELECT pg_reload_conf()
- and… node1 (primary) crashed
- Are you sure latest transactions made to node2?

Synchronous replication for HA

1. SET synchronous_standby_names
2. SELECT pg_reload_conf()
3. wait until GUC change becomes visible (reload isn’t instant)
4. remember pg_current_wal_insert_lsn() => 'X/YZ'
5. wait until standby received/flushed/applied LSN from 4

Only after that you can count standby as synchronous

Myth №5

With synchronous replication we don't need pg_rewind

Truth

- WAL on primary is written independently from standby nodes and generated not only by transactions (e.g. VACUUM)
- There is always a chance that sync standby didn’t received some parts of WAL
  - Doesn’t mean there is a data loss!
  - However, **pg_rewind** is required.

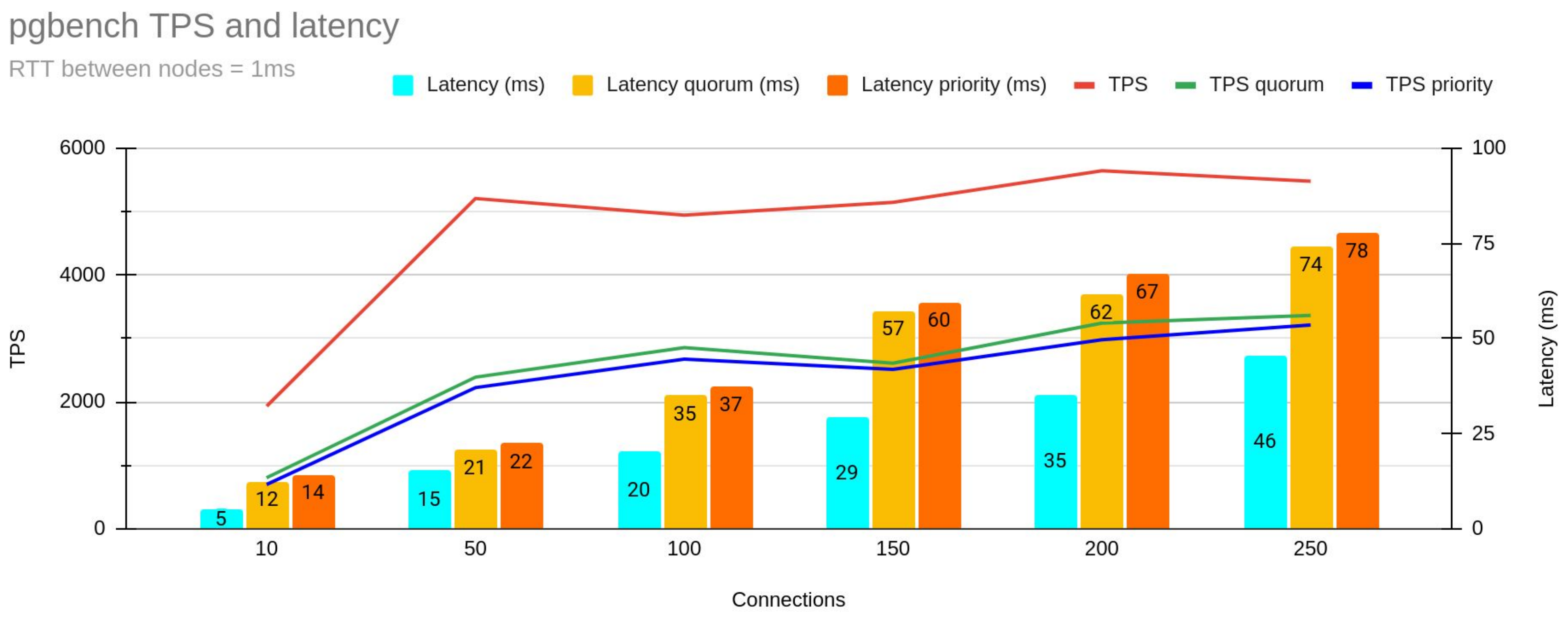
Myth №6

Synchronous replication is slow

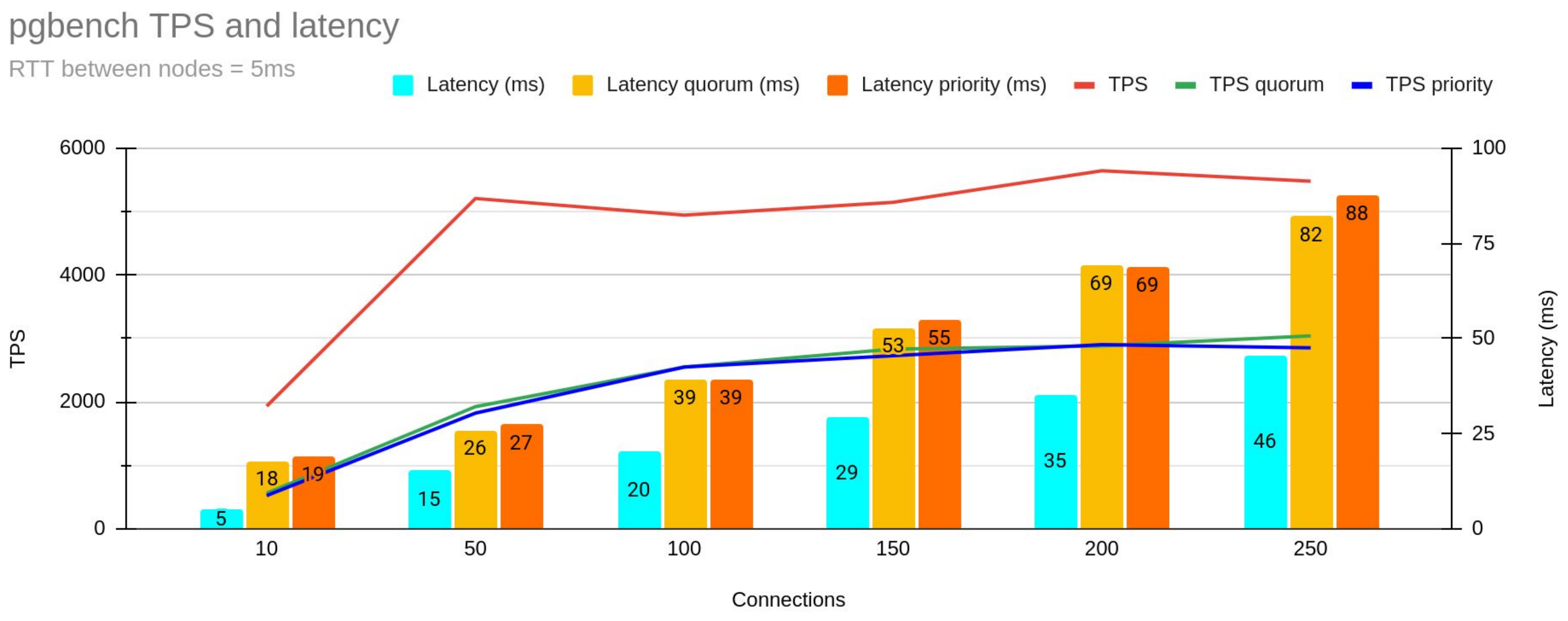
“Benchmarking” synchronous replication

- laptop + docker-compose (3 containers) + iproute2 (tc) to emulate latency
    - Default Postgres config, **max_connections** = **252**
- pgbench -i **-s 100**
- pgbench -c **$connection_num** -T 60
    - where **connection_num** = 10, 50, 100, 150, 200, 250
- **synchronous_commit** = **on**
- **synchronous_standby_names** = **'FIRST|ANY 1 (*)'**

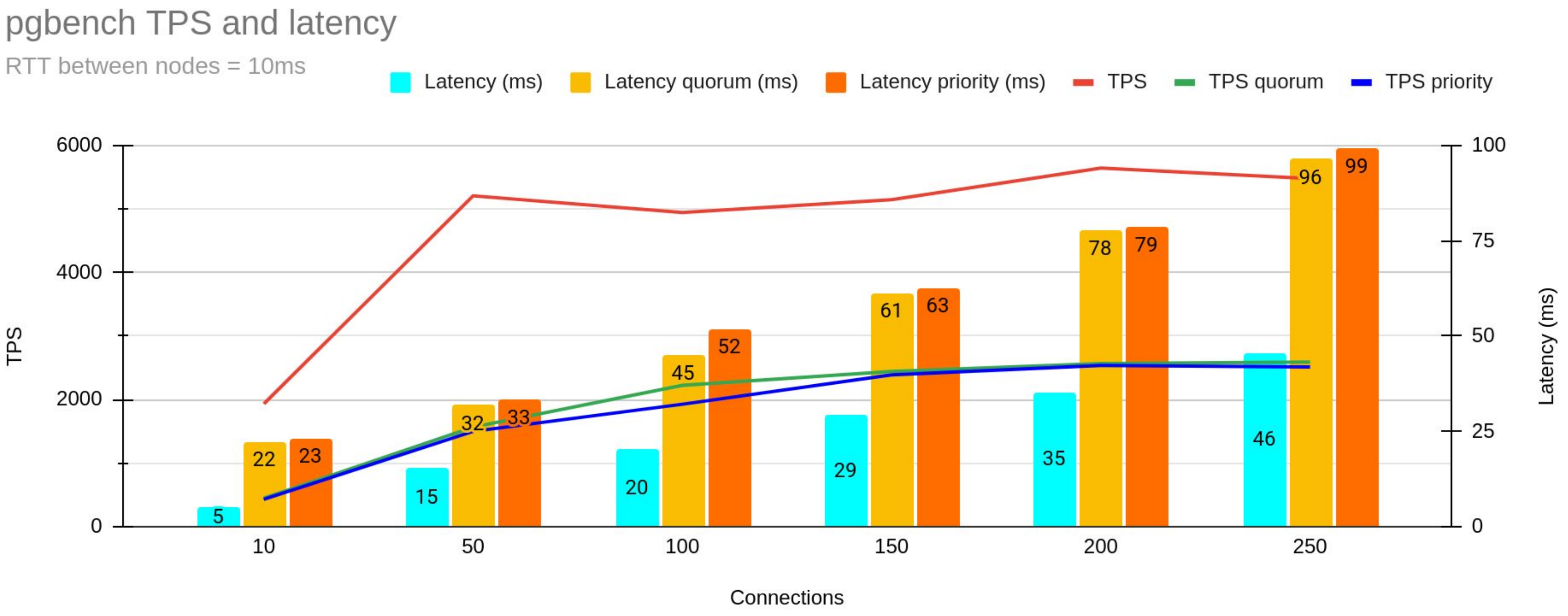
How RTT influence TPS and latency



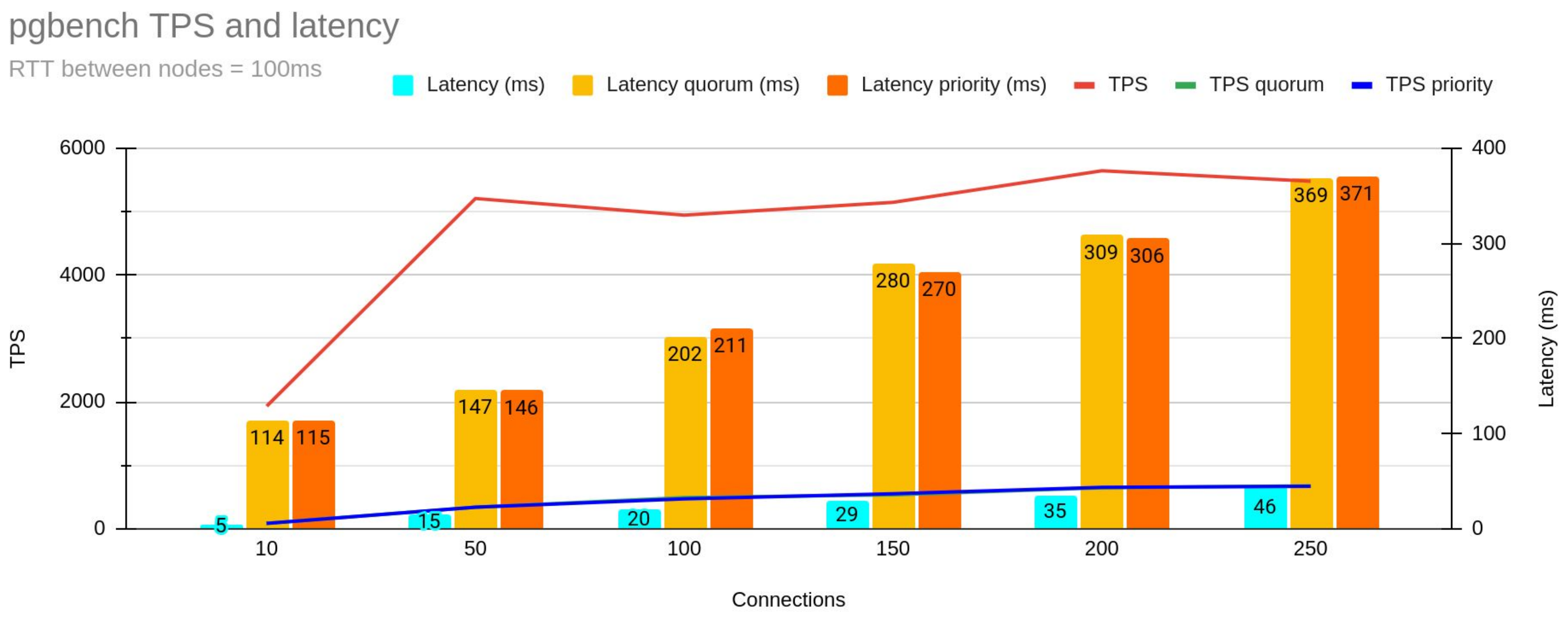
How RTT influence TPS and latency



How RTT influence TPS and latency



How RTT influence TPS and latency

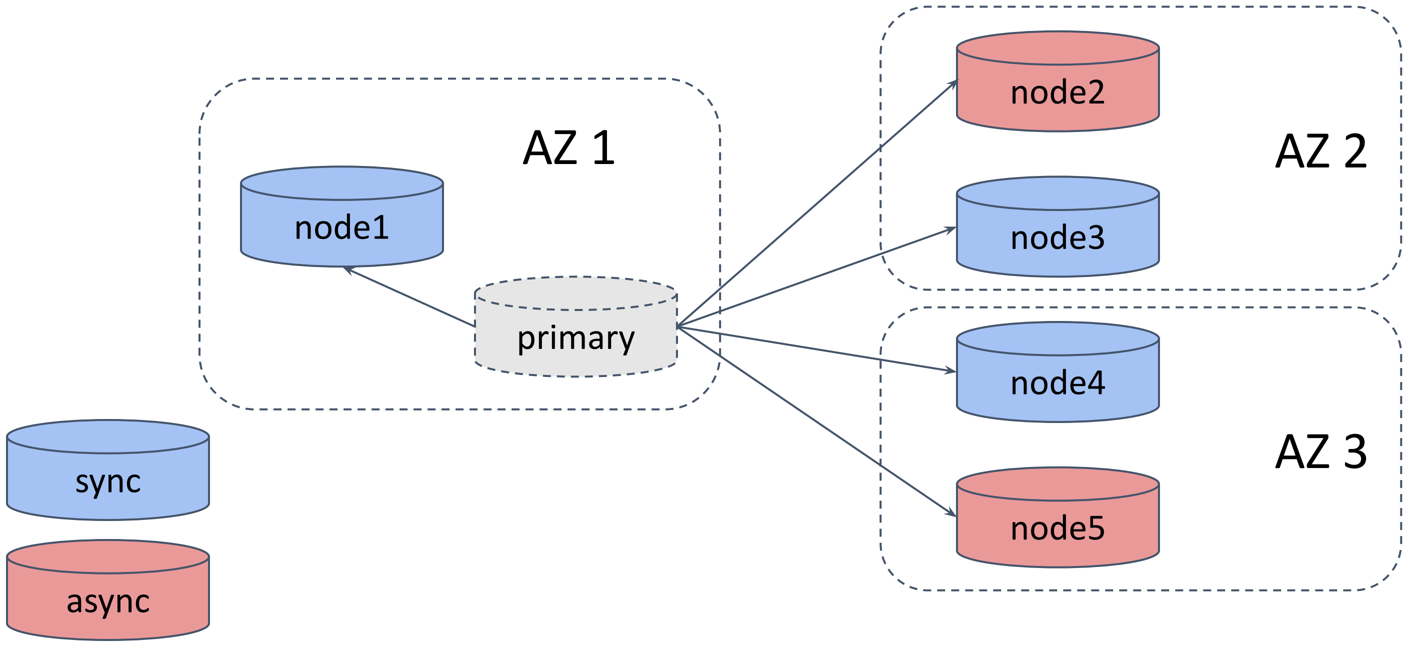


Truth

- Depends on hardware and on RTT between nodes
  - Don’t run synchronous nodes between continents!
- Additional latency due to clients waiting until sync standbys confirmed that they received/flushed/applied transaction
  - Lower TPS with the same amount of connections
- You can scale TPS by increasing connections
  - Final TPS will be lower!

Bonus: quorum commit is not AZ-aware!

synchronous_standby_names = 'ANY 3 (node1, node2, node3, node4, node5)

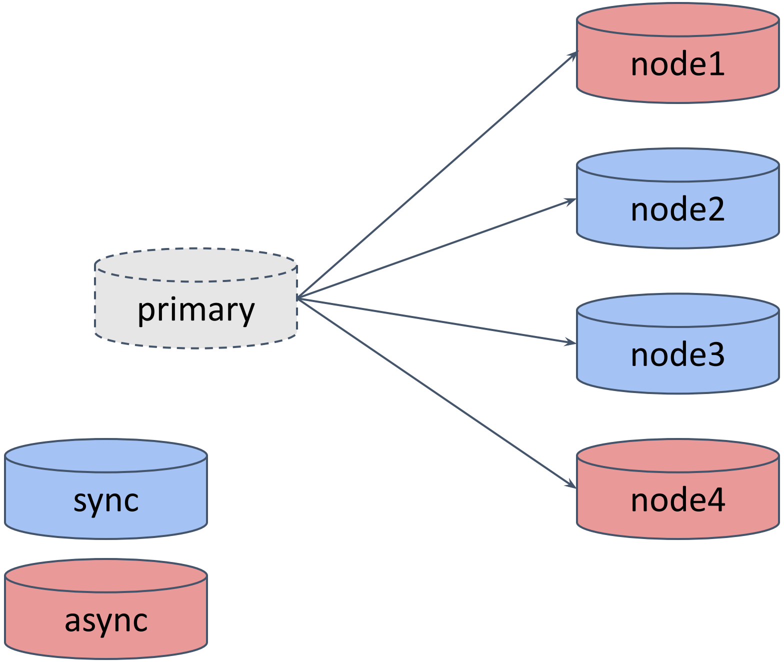


Bonus: what to do on failover

- **synchronous_standby_names = 'N (node1, …, nodeN)'**
  - Pick any node. However, better to choose the most up-to-date
- **synchronous_standby_names = 'N (node1, …, nodeM)'**
  - Need to get responses from **M-N+1** nodes to find the synchronous

Bonus: quorum-based failover (example)

synchronous_standby_names = 'ANY 2 (node1, node2, node3, node4)



We need to see at least 3 nodes to find at least 1 synchronous among them!

Questions?