ANTS AASMA

pgconf.eu 2025

CYBERTEC
POSTGRESQL SERVICES & SUPPORT

Hello

About me

What this talk is about

Why do we replicate?

- Hardware fails.

Why do we replicate?

- Hardware fails.
- Storage devices are hardware.

Why do we replicate?

- Hardware fails.
- Storage devices are hardware.
- We want our data to still be there after the failure.

Why do we replicate?

- Hardware fails.
- Storage devices are hardware.
- We want our data to still be there after the failure.
- RAID is a form of replication, but with a single point of failure.

Durability and consistency

- Two sides of the same coin.

Durability and consistency

- Two sides of the same coin.
- Both mean: “Will my transaction still be there?”

Durability and consistency

- Two sides of the same coin.
- Both mean: “Will my transaction still be there?”
- Consistency: “If I ask a replica”

Durability and consistency

- Two sides of the same coin.
- Both mean: “Will my transaction still be there?”
- Consistency: “If I ask a replica”
- Durability: “After there is a failure”

There are failures and Failures

- How much is lost, for how long and under what circumstances.
- Some failures you really can't ignore.
- In some cases, restoring from backup is acceptable.

There are failures and Failures

- How much is lost, for how long and under what circumstances.
- Some failures you really can’t ignore.
- In some cases, restoring from backup is acceptable.
- Or you classify it as a “does not happen” problem:

English > National

G-Drive Fire Destroys 125,000 Officials' Data

No Backup Available for Government Cloud System, Recovery Uncertain

It’s about waiting

- Synchronous replication is all about waiting.

It’s about waiting

- Synchronous replication is all about waiting.
- Get enough confirmations that data loss “can’t happen”.

It’s about waiting

- Synchronous replication is all about waiting.
- Get enough confirmations that data loss “can’t happen”.
  - For some value of “can’t”

It’s about waiting

- Synchronous replication is all about waiting.
- Get enough confirmations that data loss “can’t happen”.
  - For some value of “can’t”
- Almost no extra work needed.

It’s about waiting

- Synchronous replication is all about waiting.
- Get enough confirmations that data loss “can’t happen”.
  - For some value of “can’t”
- Almost no extra work needed.
- Response times get longer.

It’s about waiting

- Synchronous replication is all about waiting.
- Get enough confirmations that data loss “can’t happen”.
    - For some value of “can’t”
- Almost no extra work needed.
- Response times get longer.
- May need much more concurrency to get same throughput.

But how does it work?

What we need to know

- PostgreSQL logs all data changes into WAL.

What we need to know

- PostgreSQL logs all data changes into WAL.
  - Writing changes to one place is easier than doing it in dozen places.

What we need to know

- PostgreSQL logs all data changes into WAL.
  - Writing changes to one place is easier than doing it in dozen places.
- Order of the changes encodes dependencies.

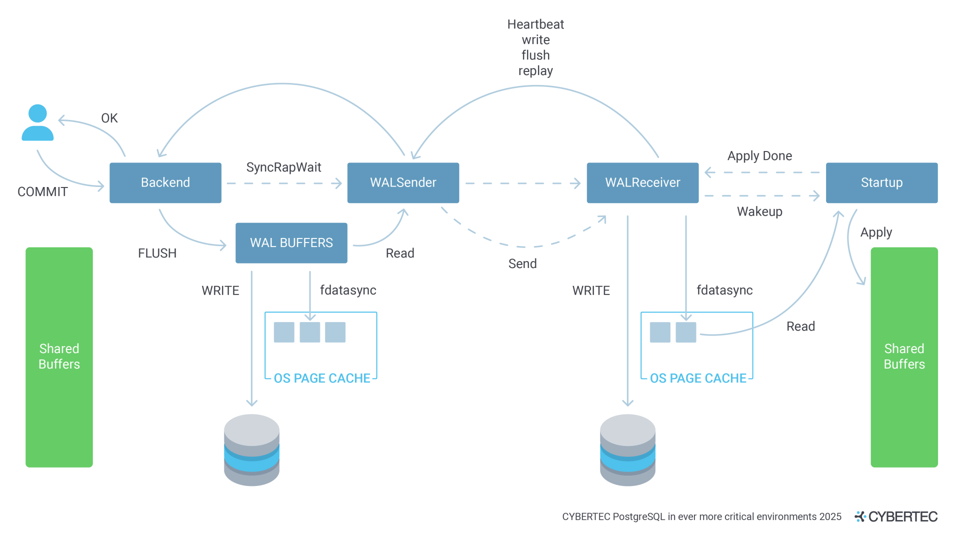
What we need to know

- PostgreSQL logs all data changes into WAL.
  - Writing changes to one place is easier than doing it in dozen places.
- Order of the changes encodes dependencies.
  - Logical - can’t allow withdraw if deposit didn’t happen.

What we need to know

- PostgreSQL logs all data changes into WAL.
  - Writing changes to one place is easier than doing it in dozen places.
- Order of the changes encodes dependencies.
  - Logical - can't allow withdraw if deposit didn't happen.
  - Physical - index pointers should not go into empty space.

Lifecycle of a commit



How long the happy path takes

- Flush WAL to disk:
  - Local NVMe: 15 µs - 250 µs average
  - Network storage: 500 µs - 2’000 µs average

How long the happy path takes

- Flush WAL to disk:
  - Local NVMe: 15 μs - 250 μs average
  - Network storage: 500 μs - 2’000 μs average
- Send data over network
  - Typical datacenter network: 100 - 500 μs
  - Same city: + couple hundred μs
  - Longer distance + 5 ms / 1’000 km

The very happy path

Assumptions: 50μs disk latency, 200μs ping, 20μs processing.

1. Flush to local disk: 50 μs
2. Walsender wakes up, reads block, it gets sent immediately: 20μs
3. Data traverses the network: 100μs
4. Packet arrives at replica, walreceiver wakes, writes to disk, flushes and sends feedback: + 70 μs
5. Feedback traverses the network: 100 μs
6. Walsender wakes up, processes feedback, wakes up backend: 20μs

Total: 360μs (vs 50μs local)

The cloud path

Assumptions: 500μs disk latency, 500μs ping (cross-AZ), 20μs processing.

1. Flush to local disk: 500 μs
2. Walsender wakes up, reads block, it gets sent immediately: 20μs
3. Data traverses the network: 250μs
4. Packet arrives at replica, walreceiver wakes, writes to disk, flushes and sends feedback: + 520 μs
5. Feedback traverses the network: 250 μs
6. Walsender wakes up, processes feedback, wakes up backend: 20μs

Total: 1’560 μs (vs 500μs local)

Why do we care about latency

- Surely nobody notices if something takes a couple milliseconds more?

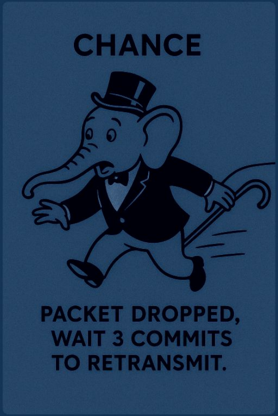
Why do we care about latency

- Surely nobody notices if something takes a couple milliseconds more?
- Highly contended workloads care (a.k.a. Admdahl's law):
  - If every commit waits for 5ms we can only manage 200 updates per second.

Why do we care about latency

- Surely nobody notices if something takes a couple milliseconds more?
- Highly contended workloads care (a.k.a. Admdahl's law):
  - If every commit waits for 5ms we can only manage 200 updates per second.
- Connection pools care.
  - Every millisecond extra latency means 1 more connection needed per 1'000 tps.

Hazards await



The (slightly) unhappy path

- Other backend is already flushing: +0.5ms flush delay

The (slightly) unhappy path

- Other backend is already flushing: +0.5ms flush delay
- Walsender's CPU is busy with other task: +1ms preemption delay

The (slightly) unhappy path

- Other backend is already flushing: +0.5ms flush delay
- Walsender's CPU is busy with other task: +1ms preemption delay
- Index build queued up 10MB of WAL: +10ms to write and +10ms to send

The (slightly) unhappy path

- Other backend is already flushing: +0.5ms flush delay
- Walsender's CPU is busy with other task: +1ms preemption delay
- Index build queued up 10MB of WAL: +10ms to write and +10ms to send
- Packet loss on the network: +3ms detection delay

The (slightly) unhappy path

- Other backend is already flushing: +0.5ms flush delay
- Walsender's CPU is busy with other task: +1ms preemption delay
- Index build queued up 10MB of WAL: +10ms to write and +10ms to send
- Packet loss on the network: +3ms detection delay
- Walreceiver CPU busy: +1ms preemption delay

The (slightly) unhappy path

- Other backend is already flushing: +0.5ms flush delay
- Walsender's CPU is busy with other task: +1ms preemption delay
- Index build queued up 10MB of WAL: +10ms to write and +10ms to send
- Packet loss on the network: +3ms detection delay
- Walreceiver CPU busy: +1ms preemption delay
- Walreceiver flushing previous packet: +0.5ms flush delay

The (slightly) unhappy path

- Other backend is already flushing: +0.5ms flush delay
- Walsender's CPU is busy with other task: +1ms preemption delay
- Index build queued up 10MB of WAL: +10ms to write and +10ms to send
- Packet loss on the network: +3ms detection delay
- Walreceiver CPU busy: +1ms preemption delay
- Walreceiver flushing previous packet: +0.5ms flush delay
- Walsender is busy sending and doesn't notice the feedback: +1ms

The (slightly) unhappy path

- Other backend is already flushing: +0.5ms flush delay
- Walsender's CPU is busy with other task: +1ms preemption delay
- Index build queued up 10MB of WAL: +10ms to write and +10ms to send
- Packet loss on the network: +3ms detection delay
- Walreceiver CPU busy: +1ms preemption delay
- Walreceiver flushing previous packet: +0.5ms flush delay
- Walsender is busy sending and doesn't notice the feedback: +1ms
- . . .

The very unhappy path

- Multipath driver gets confused: +20s

The very unhappy path

- Multipath driver gets confused: +20s
- Storage fabric switches DoS themselves: +60s

The very unhappy path

- Multipath driver gets confused: +20s
- Storage fabric switches DoS themselves: +60s
- Standby dies: . . .

Different levels of syncing

- synchronous_commit has levels
  - off - hopes and prayers mode
  - local - can survive a crash
  - remote_write - can survive a failover
  - on - can survive a crash and failover
  - remote_apply - read after write consistency

Head of line blocking

- If one commit is blocked, then all commits after it are also blocked.
- Replica that is busy syncing to disk can't confirm transactions.

Consistency issues

- You can cancel a wait for the commit.
- Commit will become visible immediately.
- But commit is not durable.
- Might be gone after failover.
- Have to be careful with retry loop.

How to see the problem

- Track total commit latency - pg_stat_statements

How to see the problem

- Track total commit latency - pg_stat_statements
- Wait events:

How to see the problem

- Track total commit latency - pg_stat_statements
- Wait events:
  - Io/WalSync - Syncing WAL to local disk

How to see the problem

- Track total commit latency - pg_stat_statements
- Wait events:
    - Io/WalSync - Syncing WAL to local disk
    - LWLock/WALWrite - Waiting on someone else to sync WAL

How to see the problem

- Track total commit latency - pg_stat_statements
- Wait events:
  - Io/WalSync - Syncing WAL to local disk
  - LWLock/WALWrite - Waiting on someone else to sync WAL
  - Ipc/SyncRep - Waiting for confirmation from standby

How to see the problem

- Track total commit latency - pg_stat_statements
- Wait events:
  - Io/WalSync - Syncing WAL to local disk
  - LWLock/WALWrite - Waiting on someone else to sync WAL
  - Ipc/SyncRep - Waiting for confirmation from standby
- pg_wait_sampling to get high resolution.

How to see the problem

- Track total commit latency - pg_stat_statements
- Wait events:
  - Io/WalSync - Syncing WAL to local disk
  - LWLock/WALWrite - Waiting on someone else to sync WAL
  - Ipc/SyncRep - Waiting for confirmation from standby
- pg_wait_sampling to get high resolution.
- `strace -p` or `perf record -p $walreceiver -e 'syscalls:sys_*'`

How to see the problem

- Track total commit latency - pg_stat_statements
- Wait events:
  - Io/WalSync - Syncing WAL to local disk
  - LWLock/WALWrite - Waiting on someone else to sync WAL
  - Ipc/SyncRep - Waiting for confirmation from standby
- pg_wait_sampling to get high resolution.
- `strace -p` or `perf record -p $walreceiver -e 'syscalls:sys_*'`
  - github.com/cybertec-postgresql/perf-analysis to find outliers.

perf-analysis

```
$ perf-analysis.py syscalls-*.script.zst \
  --include=fdatasync,pwrite64,epoll_wait,fsync \
  --stat --base=10
latency [ms] pwrite64 fdatasync epoll_wait fsync
       0.000                             1     3
       0.001    12213                  179    34
       0.010     9114
       0.100       19                  148
       1.000        2     11529        132    74
      10.000                 16          1    37
```

Throwing hardware at the problem

- Buy better storage.

Throwing hardware at the problem

- Buy better storage.
- Quorum commit can hide tail latencies.

Throwing hardware at the problem

- Buy better storage.
- Quorum commit can hide tail latencies.
  - Needs at least 2 replicas.

Quorum commit

- With 2 replicas it is unlikely that both have problem simultaneously.
- `synchronuos_standby_names = 'ANY 1 (node2, node3, node4)'`
- Primary still needs to flush before replication can begin.
  - PostgreSQL could improve on this.

Picking the right replica

- If the primary fails then `quorum` other nodes have the latest commit.
- Need to reach `num_replicas` - `quorum` + 1 nodes to be sure to see at least one of them.
- Look at `pg_last_wal_receive_lsn()` and `pg_last_wal_replay_lsn()`
- Or use Patroni `synchronous_mode: quorum`

Summary

- Figure out if you can afford to lose transactions.
- If you have to use it, know that every performance issue is amplified.
- Be prepared to use more connections.
- Add replicas to hide bad latencies.
- Automatically managing quorum commit is hard, use existing tools.

Our partners at PGConf.EU

readyset

redgate

OpenSourceDB
Your Trusted Data Partner

Microsoft

PROVENTA

Open Alliance

For PostgreSQL Education

Your Pathway to Verified PostgreSQL Skills

Scan for Updates

oapg-edu.org

Vienna

17 September 2026

PGDay Austria returns in 2026

Scan for updates

pgday.at