Multi-master replication for Postgres

K. Knizhnik, C. Pan, S. Kelvich

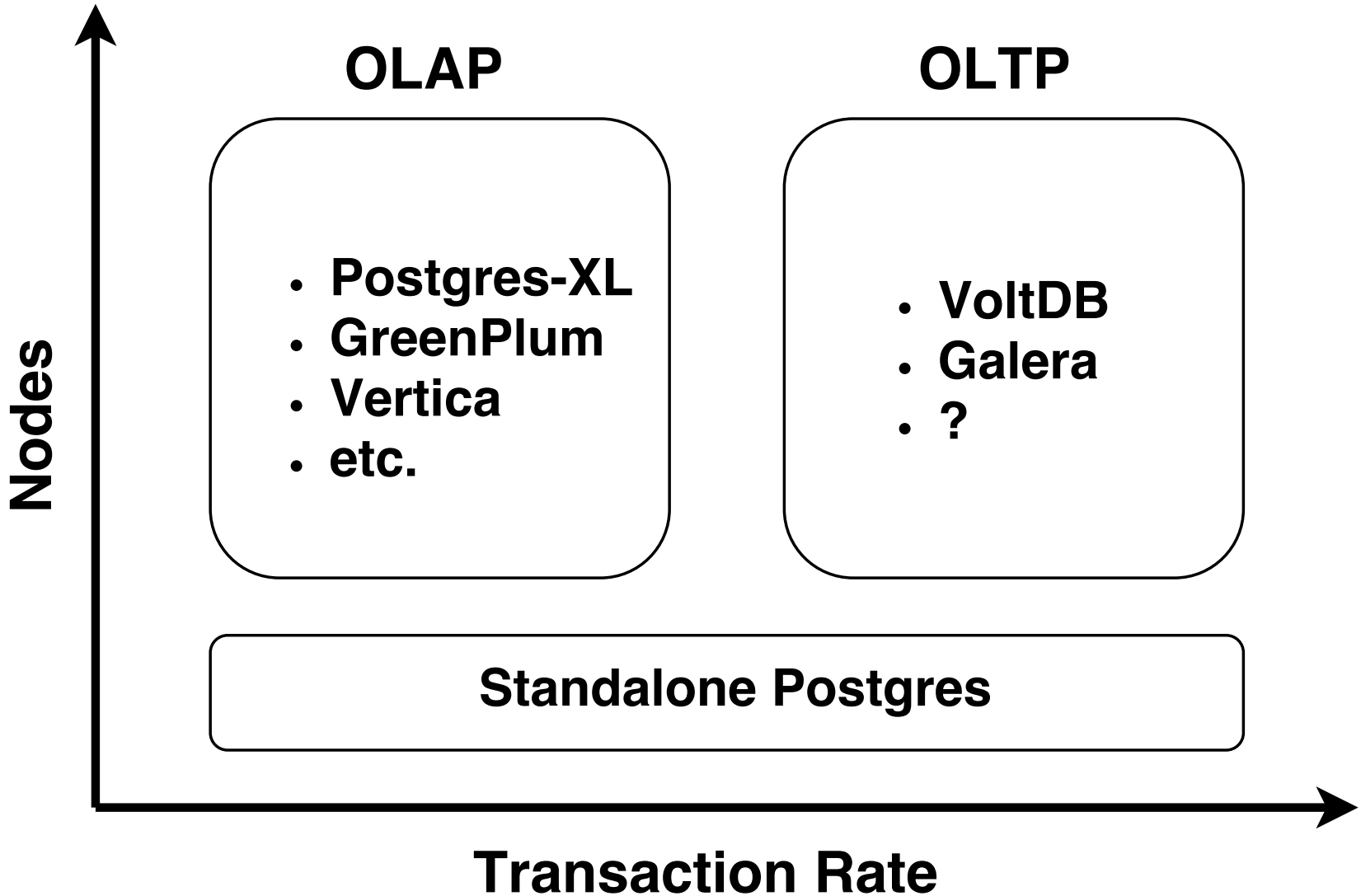
PROFESSIONAL Postgres

Postgres PROFESSIONAL

Contents

- Design objectives
- Implementation/internals
- Tests
- Configuration
- Roadmap

Design objectives



Design objectives

We want:

- Fault-tolerance in easy way
- OLTP-style load
- Compatibility with standalone postgres
- Possibility to reuse as metadata storage for sharded cluster

Design objectives

Replication:

- Identical replicated data on all nodes
- Possibility to have local tables
- Writes allowed to any node
  - => Easy to use
  - => We need to take care about proper isolation

Design objectives

Transaction manager. We want:

- Avoid single point of failure.
  - +: Spanner, Cockroach, Clock-SI
  - —: Pg-XL, ...
- Avoid network communication for Read-Only transactions
  - +: HANA, Spanner, Cockroach, Clock-SI
  - —: Pg-XL, ...

Design objectives

Fault tolerance.

- Paxos. Distributed consensus, low level.
- Raft. Complete state-machine replication solution with failure detector on timeouts and autorecovery. But all writes are proxied to one node.
- 2PC. Blocks in case of node and coordinator failure. Postgres already support 2pc.
- 3PC-like. Extra message between "P"and "C". 3PC, Paxos commit, E3PC.

Design objectives

Summary.

- No performance penalty for reads.
- Tx can be issued to any node.
- No special actions required in case of failure.

Implementation

github.com/postgrespro/postgres_cluster

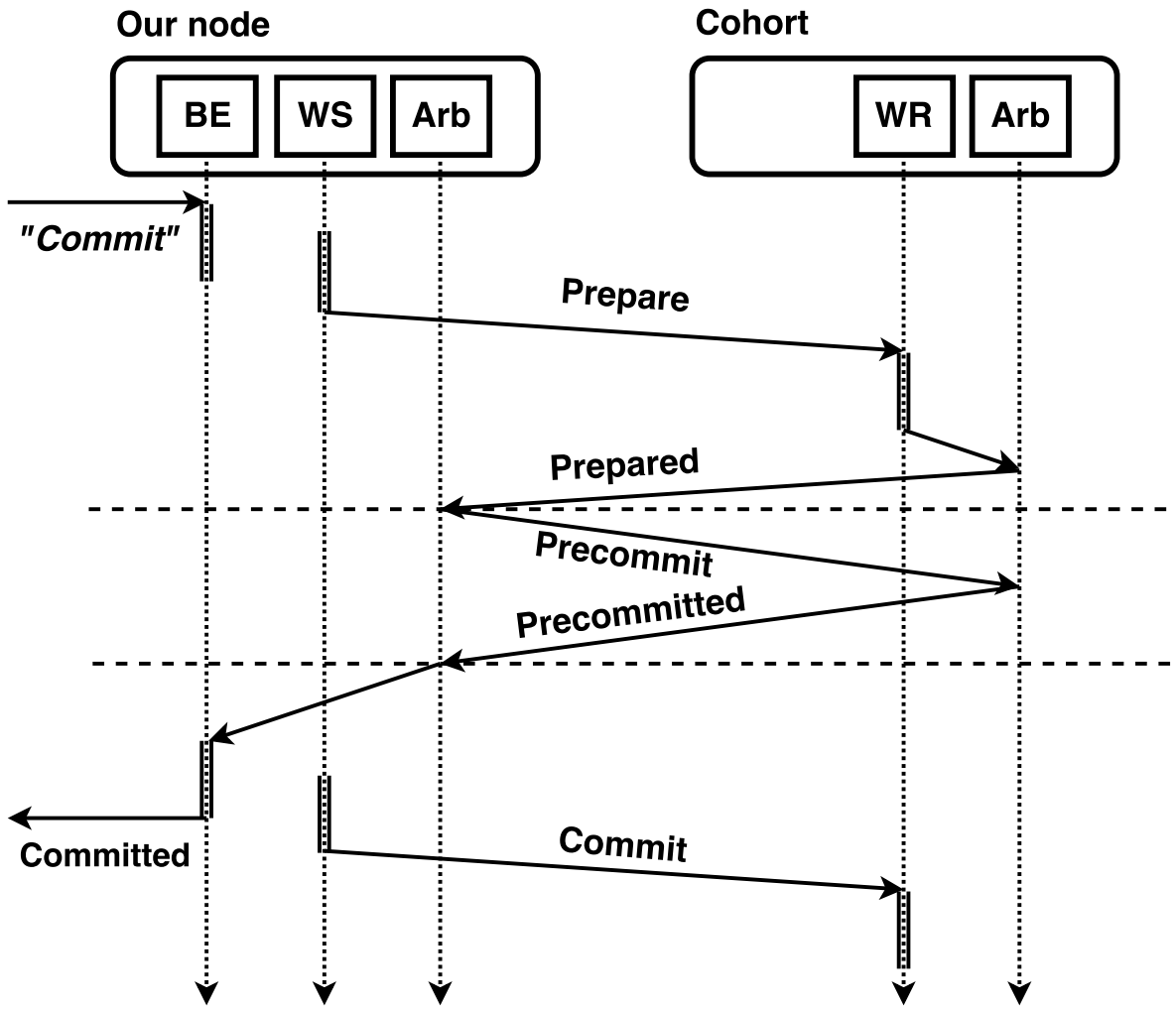
- Patched version of Postgres 9.6
  - Transaction Manager API + Deadlock detection API.
  - Logical decoding of 2PC transactions.
- Mmts extension.
  - Transaction Manager implementation (Clock-SI)
  - Logical replication protocol/client
  - Hooks on transaction commit and transforms it into 2PC.
  - Bunch of bgworkers.

Implementation

Mmts uses logical replication/decoding.

- In-core support and extension by 2ndQuadrant.
- Very flexible:
  - Can skip tables
  - Replication between different versions
  - Logical messages

Implementation



BE – backend, WS – Walsender, Arb – Arbiter, WR – Walreceiver

Implementation

Transaction Manager.

- Clock-SI algorithm (MS research)
- Make use of CSN instead of running lists. (we track xid-csn correspondence in extension, but there is ongoing work to have CSN in-core by Heikki and Alexander)

Implementation

DDL replication.

- Statement-based.
- Happily, postgres support 2PC for almost all DDL (alter enum already fixed in -master)
- CREATE TABLE AS, CREATE MATVIEW, etc – tricky, mixes DDL and DML.
- Temp tables are tricky – shouldn't be replicted.
- Depends on environment (search_path, auth, etc.)

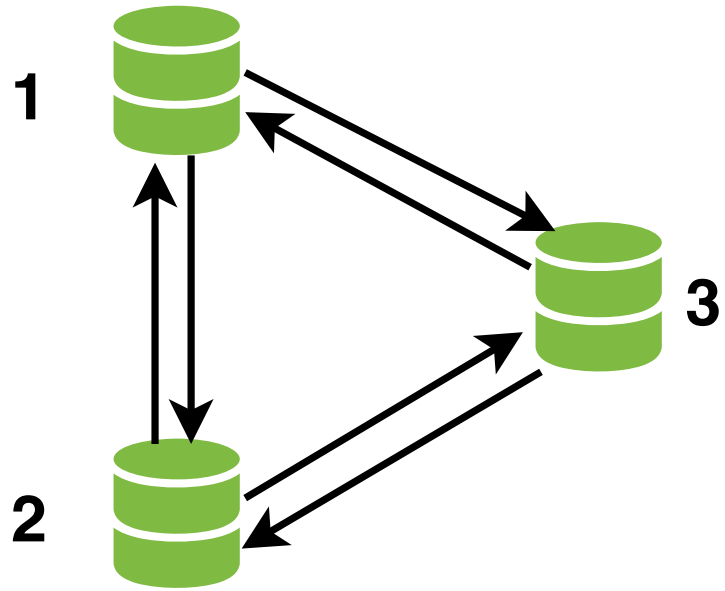
Implementation

Postgres compatibility.

- almost FULLY compatible with pg.
- 162 of 166 regressions tests pass as is.
- 1 test is using prepared statement inside CREATE TABLE AS (CTA).
- 3 tests are using CTA(CTA(TEMP TABLE)).
- Some obvious way to abuse statement based replication, e.g. write function that create table with name based on current timestamp.
- Also sequences can add pain.

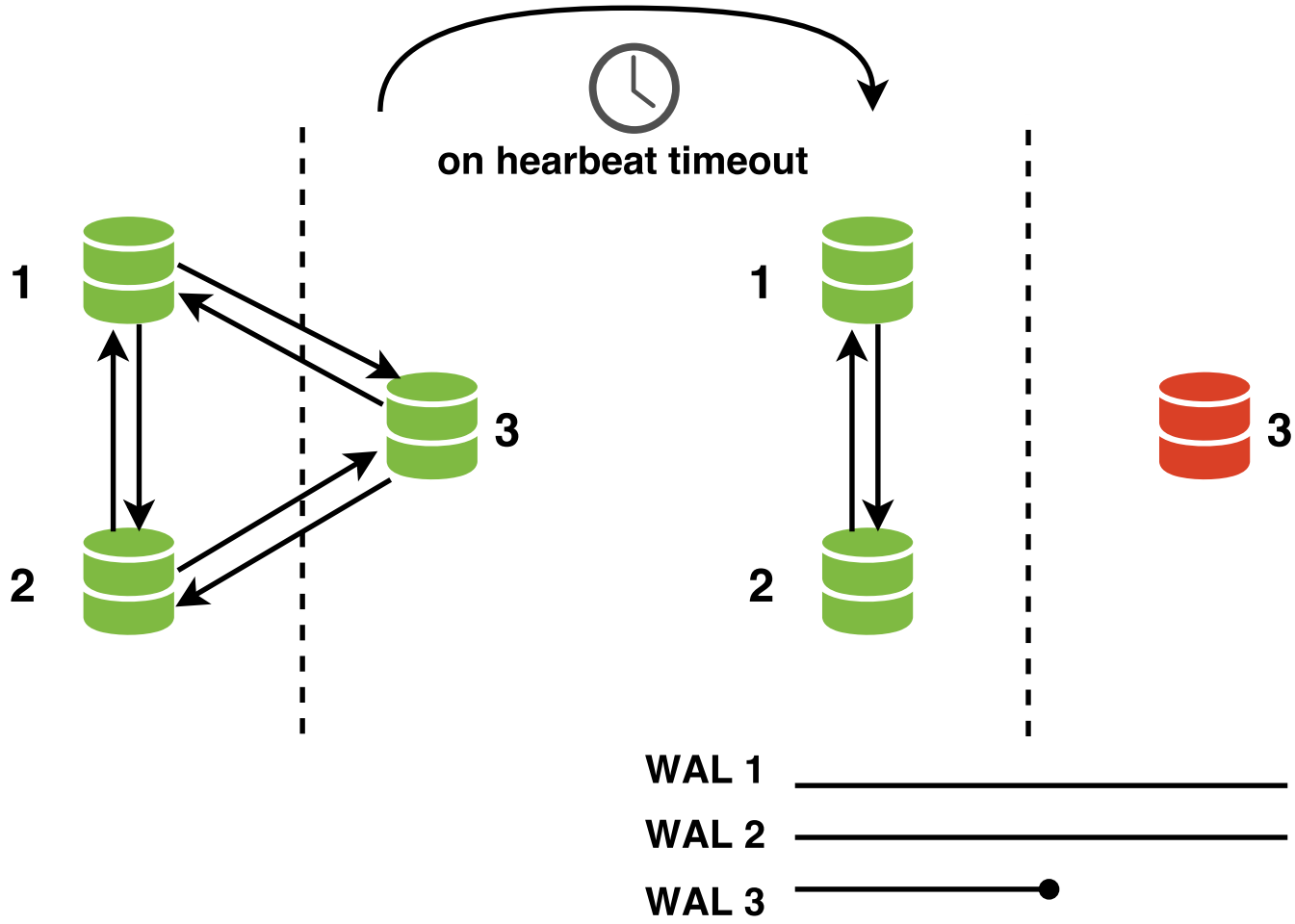
Implementation

Automatic recovery: normal work



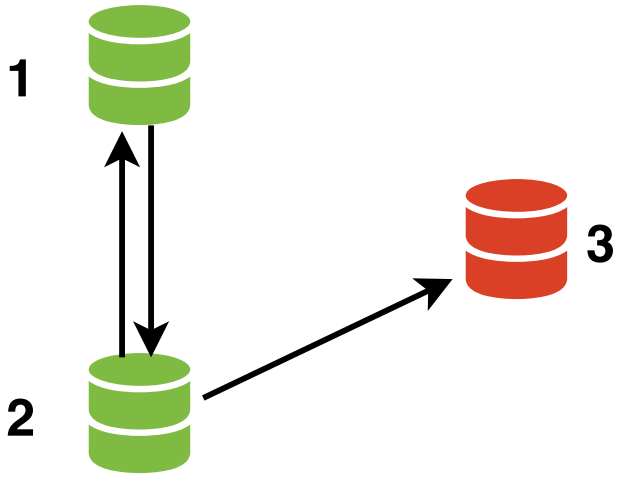
Implementation

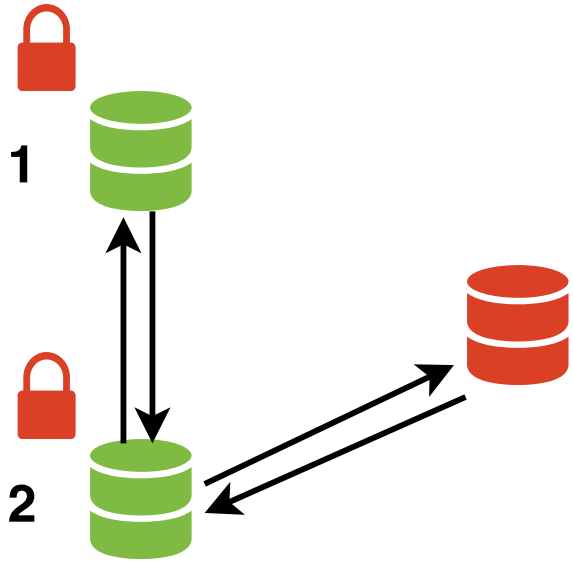
Automatic recovery: network split

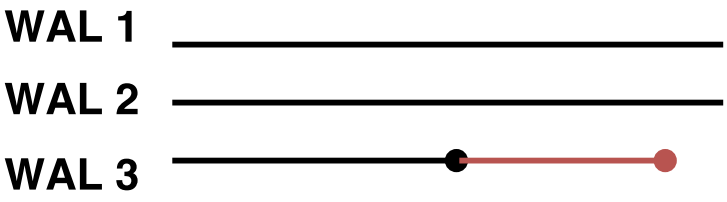


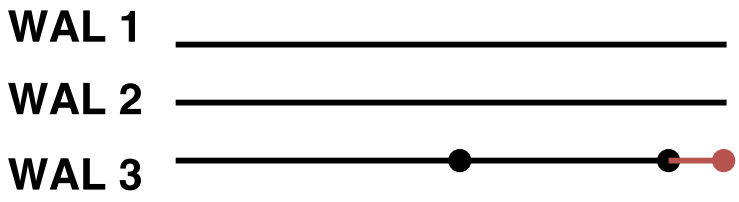
Implementation

Automatic recovery: recovery process



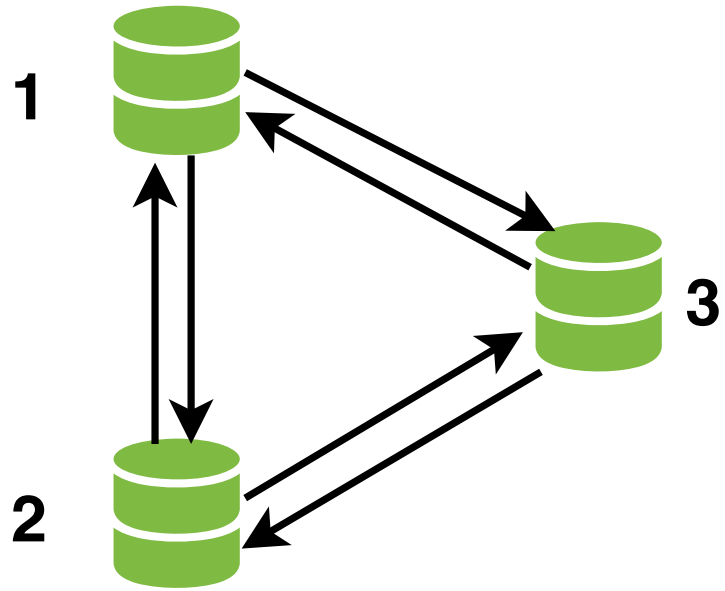






Implementation

Automatic recovery: normal work again



Configuration

Not that hard:

- Install mmts extension
- Postgres:
  - max_prepared_transactions
  - wal_level = logical
  - max_worker_processes, max_replication_slots, max_wal_senders
  - shared_preload_libraries = 'multimaster'
- Multimaster extension:
  - multimaster.node_id = ...
  - multimaster.conn_strings = '...'

Testing

We want:

- Test cluster liveness against network problems, restarts, timeshifts, etc.
- Sound like Jepsen. But unfortunately it uses ssh on precreated vm's/servers. That's okay for single test, but painful for CI.
- No sane way of testing network split with processes, i.e. postgres TAP test framework is not helpful with that.

Postgres PROFESSIONAL

Testing

So we are using python unittest with docker.

- 3-5 containers is _way_ faster to start than vm's.
- takes 10 seconds to compile mmts extension, init and start cluster.
- failure injection via docker.exec (iptables, shift time, etc).
- compatible with Travis-CI.

Testing

- Testing itself: attach clients to each node of cluster and start abusing nodes.
- Client: bank-like test case. Transfer money between accounts with concurrent total balance calculation.

Testing

Failures injected:

- Node stop-start
- Node kill-start
- Node in network partition
- Edge network split (a.k.a. majority rings)
- Shift time
- Change clock speed on nodes with libfaketime *

* – not yet implemented.

Testing

Performance.

- Read-only tx speed is the same as in standalone postgres.
- Commit takes more time (two net roundtrips).
- Logical decoding slows down big transactions – but that should be fixed, patch on commitfest.

Roadmap

- Release a public beta
- Try to commit twophase decoding patch to pg
- Try to commit transation manager patch to pg
- Raise discussion about replication/decoding of catalog content