pg_paxos: Table Replication through Distributed Consensus

Marco Slot

marco@citusdata.com

Distributed consensus

Example consensus problems:

- I have N servers, and need exactly one of them to do something.
- N replicas receive changes concurrently, need to agree on order.

Impossible to always reach consensus under arbitrary failure.

Paxos is a probabilistic algorithm for reaching consensus.

Paxos

The Part-time Parliament (Leslie Lamport, 1998) abstract:

“Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament’s protocol provides a new way of implementing the state-machine approach to the design of distributed systems.”

Paxos made Simple (Leslie Lamport, 2001) abstract:

“The Paxos algorithm, when presented in plain English, is very simple”

Paxos

paxos(key,value) is a function that returns the same value on all nodes in a group, and the value is one of the inputs.

Runs in two phases:

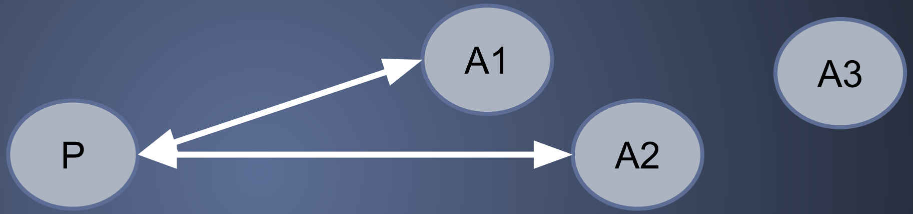
1. Proposer asks nodes to *prepare* for a new proposal
   Majority has to *promise* to participate
2. Proposer *requests acceptance* of a value
   Majority has to *accept*

If majority accepts, Paxos completes, otherwise… retry.

Paxos: Phase 1

Proposer to majority:

“Please don’t accept proposals with a lower number than i”



Acceptor:

- “Ok”
- “I already *received* a competing proposal j > i”

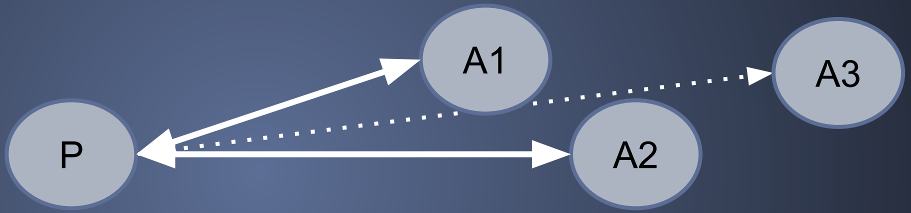
  → Proposer sets i > j and starts over
- “I already *accepted* value x from proposal j < i”

  → Proposer uses the *value* with highest j instead of input

Paxos: Phase 2

Proposer to acceptors:

“Please accept value x for proposal i”



Acceptor:

- “Ok”
- “I already received a competing proposal j > i”
  ⟶ Proposer starts over with i > j

Finally, inform all nodes of consensus (if possible).

Why does it work?

If a majority accepts, that means no other proposal has completed phase 1 since you did.

Otherwise, at least one node would have rejected your proposal.

Thus, it is guaranteed that:

- other proposals will see your value when they complete phase 1
- yours is the highest proposal number that got accepted, since it was higher than any other proposal that completed phase 1 and no other node has completed phase 1 since.

Thus nodes will always use your value.

Paxos State Machine (Multi-Paxos)

State machine implemented on a set of nodes using Paxos.

State is determined by a sequence of inputs (writes).

Nodes run Paxos for each write using increasing round numbers:

```
paxos(0, 'set x = 6')
paxos(1, 'set y = 7')
paxos(2, 'set y = 9')
```

Once a node knows rounds 0 to k were accepted by the majority, they can be applied to the local state.

Paxos State Machine

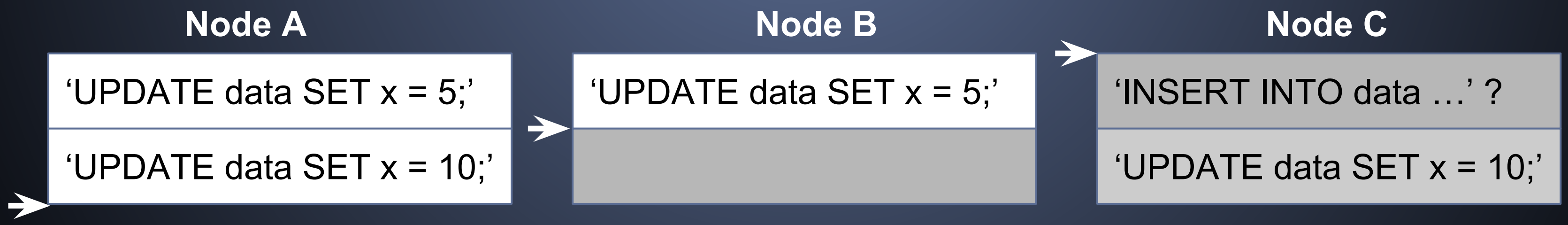
To write a value to the distributed log at position i:

```
while(paxos(round,query) != query) round++;
```

To perform a consistent read:

```
while(round < max_round()) paxos(round++,'');
```

Each node has its own copy of the log.



Node A	Node B	Node C
‘UPDATE data SET x = 5;’	‘UPDATE data SET x = 5;’	‘INSERT INTO data …’ ?
‘UPDATE data SET x = 10;’		‘UPDATE data SET x = 10;’

pg_paxos

pg_paxos is an extension for PostgreSQL that provides consistent, fault-tolerant table replication through Multi-Paxos

… with low throughput and high latency

- ✗ An alternative to streaming or logical replication.
- ✗ Magic Distributed PostgreSQL.
- ✓ A useful building block for distributed systems.

pg_paxos

Available on Github: https://github.com/citusdata/pg_paxos/

1. Basic implementation of Paxos and Multi-Paxos in PL/pgSQL using dblink.
2. Consistent table replication implemented using Multi-Paxos by automatically logging and executing DML statements.

Warning: Somewhat experimental

PL/pgSQL

Surprisingly suitable language for implementing Paxos:

- Transactional semantics come for free
- Managing data is easy
- Simple networking API: dblink
- Can do RPC by remotely calling a PL/pgSQL function
- Runs on managed PostgreSQL (Amazon RDS / Heroku)

CREATE EXTENSION pg_paxos

Metadata in pg_paxos:

`pgp_metadata.group`

Paxos groups in which server participates

`pgp_metadata.host`

Hosts in the Paxos group

`pgp_metadata.round`

The Multi-Paxos log with state of each proposal

`pgp_metadata.replicated_tables`

Tables that are automatically replicated using pg_paxos

pg_paxos internals

Functions in pg_paxos:

```
SELECT paxos(..., round_number, query)
```

Propose a query in a given round

or get a query by using `''`

```
SELECT paxos_apply_log(..., round_number)
```

Execute queries in the log up to a specified round number

```
SELECT paxos_apply_and_append(..., round_number, query)
```

Append a query to the log and execute preceding queries

Table replication

To replicate a table:

```
CREATE TABLE data (...);
SELECT paxos_create_group('pgcon','host=orig.server');
SELECT paxos_replicate_table('pgcon','data');
```

Queries on the data table are intercepted using executor hook.

Cluster set-up

To join a Paxos group:

```
SELECT paxos_join_group('pgcon','host=orig.server',
                                'host=new.server');
```

Joining clones the state of orig.server and then logs:

```
INSERT INTO pgp_metadata.host VALUES('new.server',5432,3);
```

Handling writes

When you run a DML/DDL query on a replicated table, e.g.:

```
  UPDATE data SET greeting = 'hello' WHERE object = 'world';
```

Then pg_paxos appends this query to the Multi-Paxos log.

```
  SELECT paxos_apply_and_append(..., query);
```

When it knows the position of the query in the log, it first executes all preceding queries in the log and then executes the UPDATE.

Handling reads

When you run a SELECT query on a replicated table, e.g.:

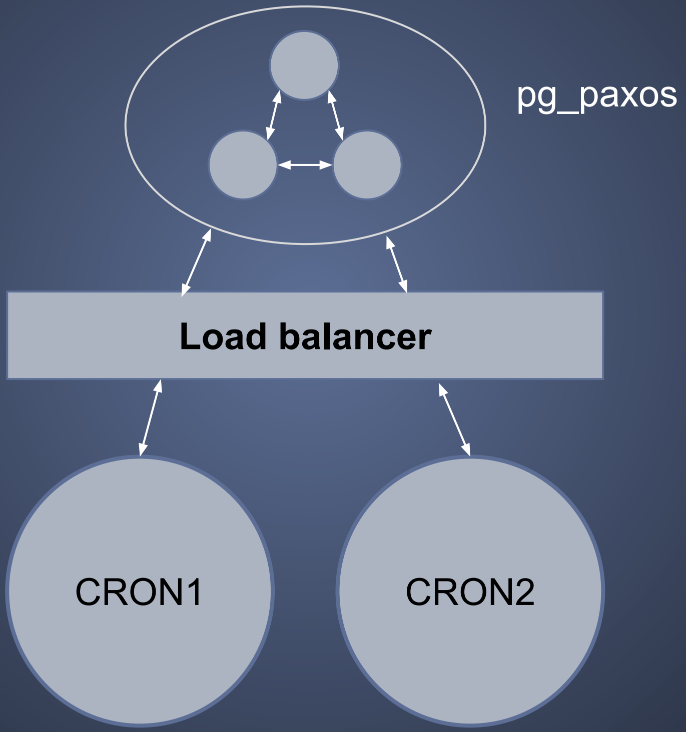
```
SELECT greeting FROM data WHERE object = 'world';
```

pg_paxos finds the highest accepted round number among a majority and executes preceding queries.

```
SELECT paxos_apply_log(..., paxos_max_group_round(...));
```

It knows that when the SELECT started, there was no consensus on higher round numbers.

Demo



Applications

Low read/write volume applications with strong consistency requirements, e.g.:

- Managing cluster membership
- Automated fail-over
- Job scheduler
- Data/schema migrations
- Source for metadata
- Distributed locks

Why not Raft?

Multi-Paxos:

- … can be implemented in PL/pgSQL
- … has a simpler minimal implementation
- … can be adapted to requirements
- … is mathematically very elegant

Short answer:

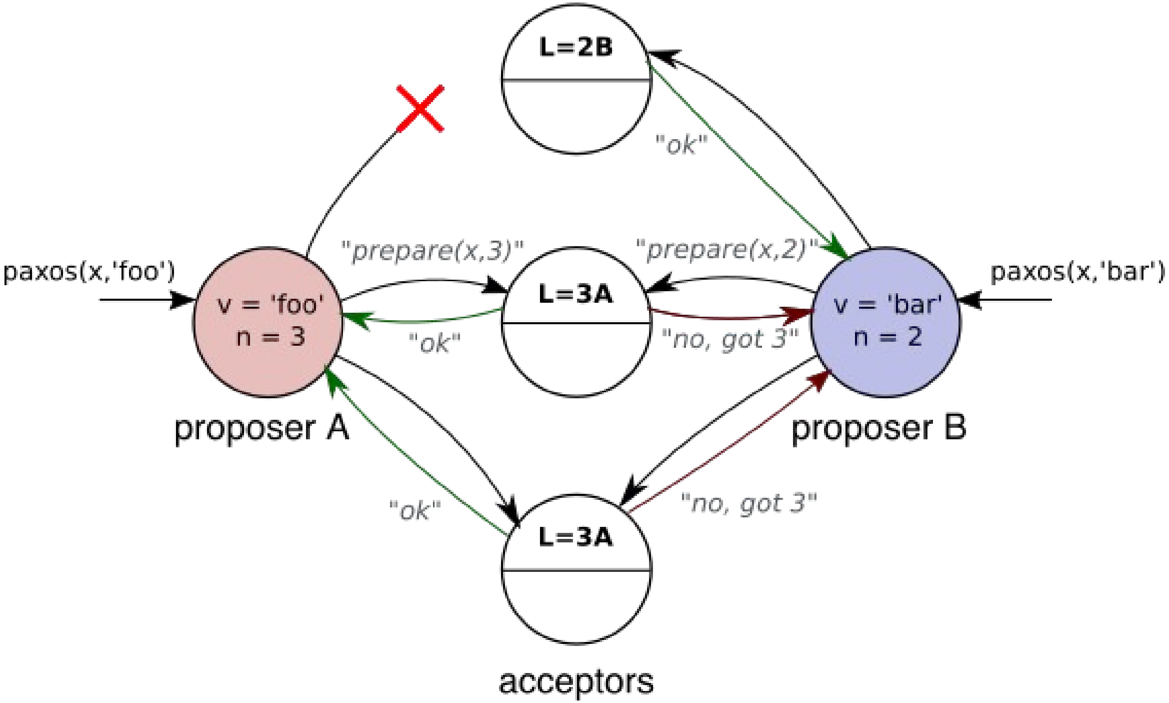
- I knew Multi-Paxos and PL/pgSQL

Questions?

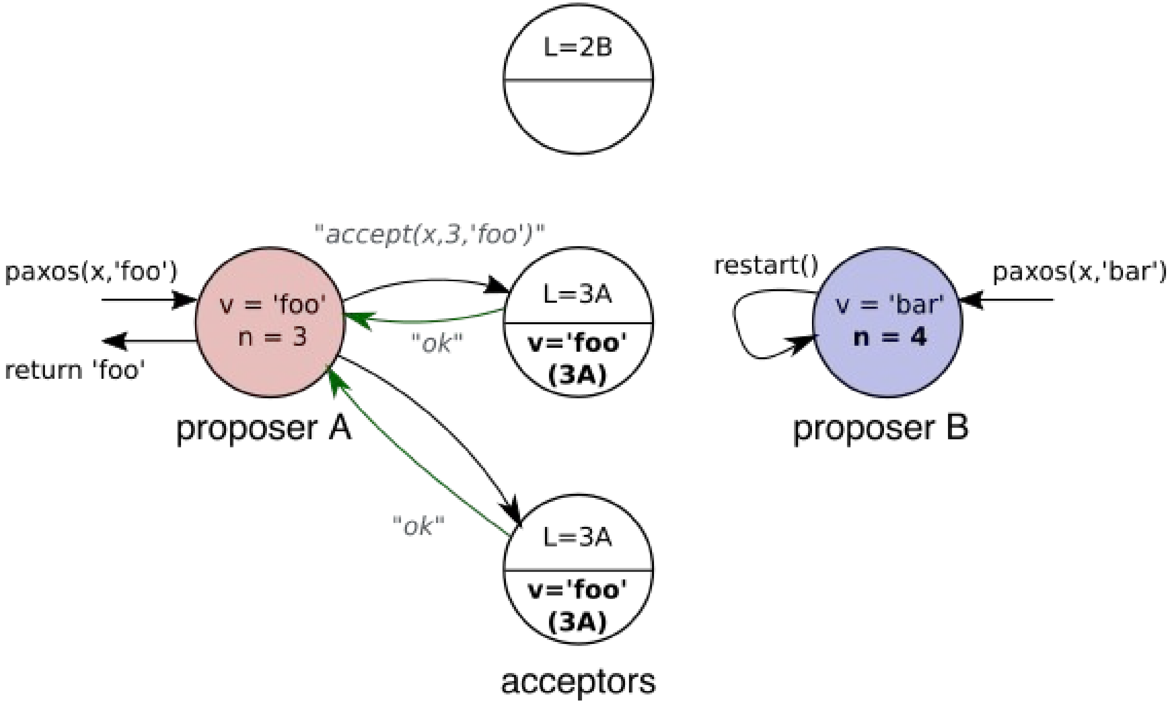
marco@citusdata.com

https://github.com/citusdata/pg_paxos/

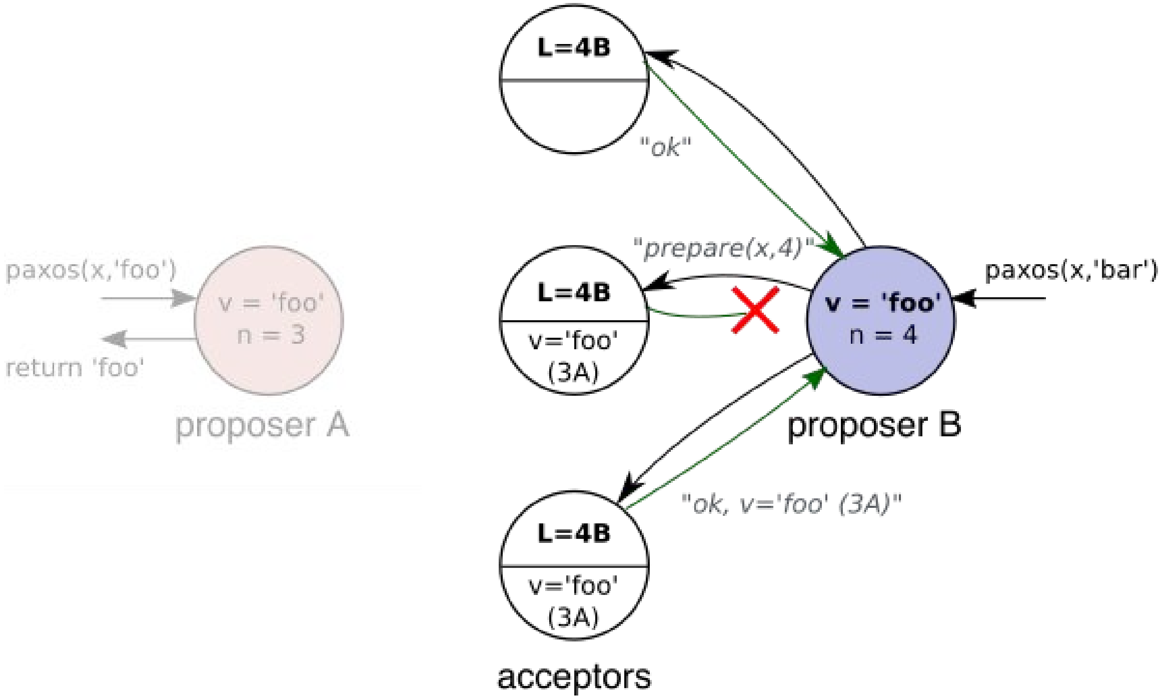
Proposer A completes phase 1, Proposer B is rejected



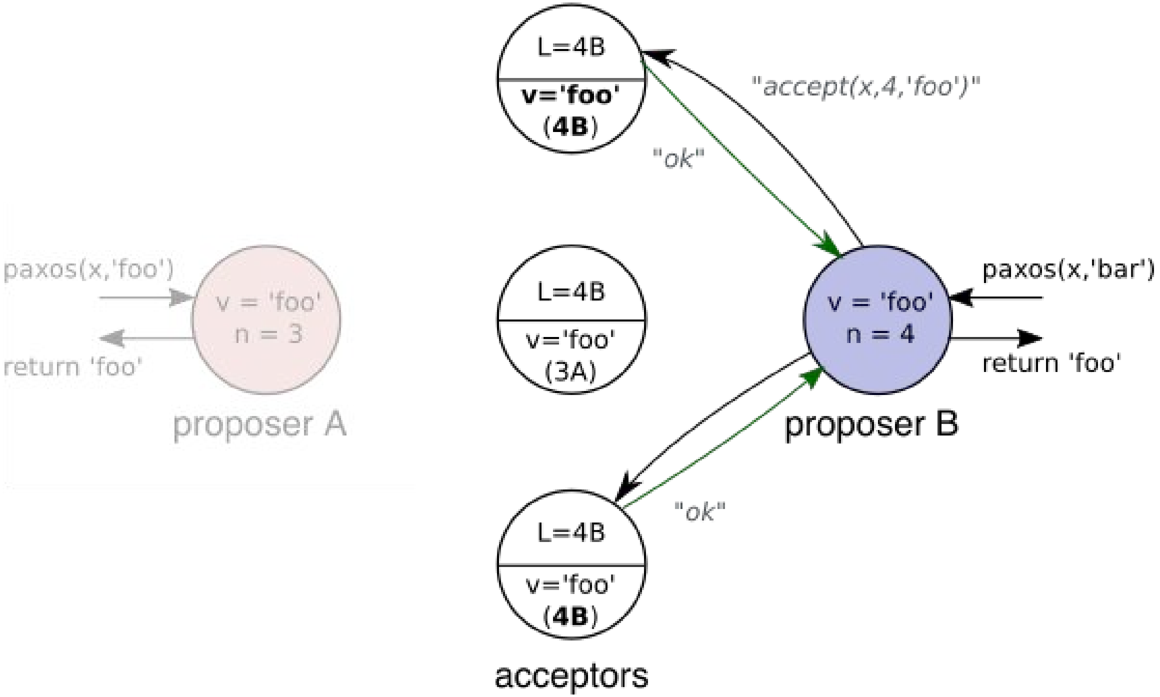
Proposer A completes phase 2, Proposer B restarts



Proposer B completes phase 1, changes value to 'foo'



Proposer B completes phase 2 with value 'foo'


L=4B
v='foo'
(4B)
"accept(x,4,'foo')"
"ok"
paxos(x,'foo')
v = 'foo'
n = 3
return 'foo'
proposer A
L=4B
v='foo'
(3A)
v = 'foo'
n = 4
paxos(x,'bar')
return 'foo'
proposer B
"ok"
L=4B
v='foo'
(4B)
acceptors