2ndQuadrant®
PostgreSQL

# Sharding in PostgreSQL

*28th February 2020*

**Sachin Kotwal**
Senior PostgreSQL DBA, 2ndQuadrant

**Hari Kiran**
PostgreSQL Consultant, 2ndQuadrant

# Why Sharding ?

- What is sharding? We all know that.
- Why do you have to think about sharding?
- Application level sharding
- Database level sharding
- Which sharding is better? Why?

# Sharding in PostgreSQL

- Existing sharding solutions
- Sharding with inheritance and declarative table partitioning
- Auto partitioning/sharding
- Use cases and Benefits of sharding

# Existing PostgreSQL Sharding solutions

## PL/Proxy

- PL/Proxy allows to do remote procedure calls between PostgreSQL databases, with optional sharding.
- More details at: https://plproxy.github.io/
- It is complex to setup and maintain

## Postgres-XL

- Postgres-XL successor of Postgres-XC is an all-purpose fully ACID open source scale-out SQL database solution.
- More details at: https://www.postgres-xl.org/
- Regular database maintenance operations are yet not that easy

# Existing PostgreSQL Sharding solutions

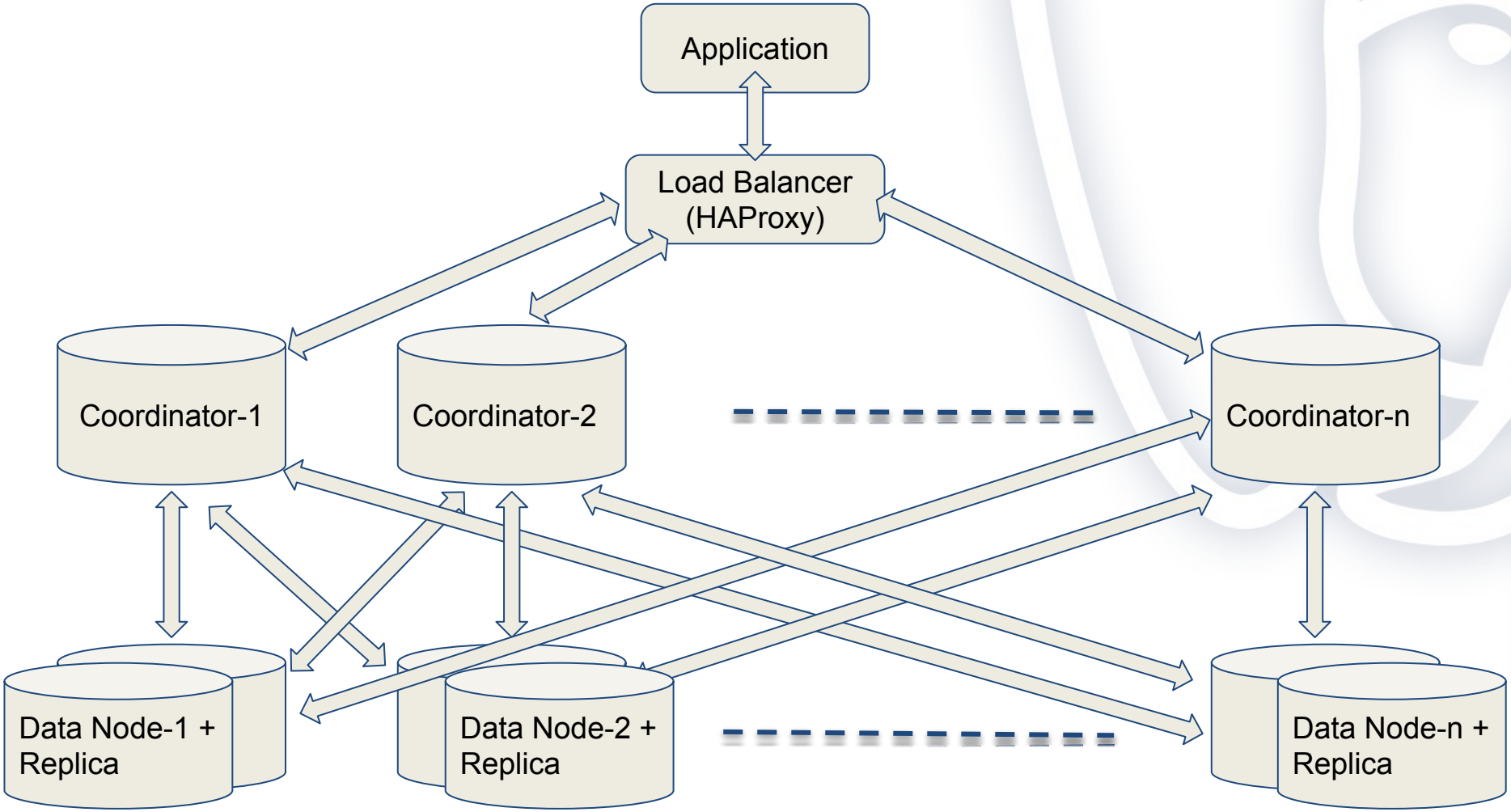
## BDR3

- Enterprise-grade solution for sharding
- Applications do not need heavy modifications
- High Availability in-built
- 24x7 Support ensures peace of mind

## Built-in Sharding

- This is inbuilt sharding solution with **postgres_fdw** and partitioning, allows to push query execution to data nodes and combine result at coordinator node share result back to application..
- More info at: https://wiki.postgresql.org/wiki/Built-in_Sharding
- Additional subsystems like global transaction manager(GTM) and global snapshot manager(GSM) needs to be developed

# Food for thought Architecture



# Sharding with inheritance table partitioning

- On Data node:
  ```
  CREATE TABLE stocks_history_y2019m09(..CHECK (partition boundaries)..)
  ```

- On Coordinator nodes CREATE EXTENSION, SERVER and USER MAPPING.
  ```
  CREATE TABLE stocks_history (..);
  CREATE FOREIGN TABLE (..CHECK (partition boundaries)..) INHERITS (..)
  SERVER ..;
  CREATE OR REPLACE FUNCTION .. RETURNS TRIGGER AS $$ .. $$
  CREATE TRIGGER .. BEFORE INSERT ON .. PROCEDURE..();
  ```

- There is no automatic way to verify that all of the CHECK constraints are mutually exclusive
- Indexes apply to single tables and not to their inheritance children
- Update partition key needs handle separately
- Manual VACUUM or ANALYZE need to run on each child table individually

# Sharding with declarative partitioning

- Create partition table definition on Data node with appropriate partition boundaries using CHECK constraint on partition column.

- On Coordinator nodes CREATE EXTENSION, SERVER and USER MAPPING will be same as Inheritance partition sharding

```
CREATE TABLE ... PARTITION BY RANGE();

CREATE FOREIGN TABLE ... PARTITION OF () FOR
VALUES FROM (..) SERVER ..;
```

# PostgreSQL planner on Sharding

- Declarative and Inheritance based sharding have same planner behaviour

```
shard=# EXPLAIN ANALYZE SELECT * FROM stocks_history WHERE
buy_at>'2019-10-01 01:00:00' AND buy_at<'2019-11-01 01:00:00';

                                  QUERY PLAN
-------------------------------------------------------------------------------
-
 Append  (cost=100.00..221.25 rows=2 width=2020) (actual time=3.887..4.150
rows=30 loops=1)
   ->  Foreign Scan on stocks_history_y2019m10  (cost=100.00..110.62 rows=1
width=2020) (actual time=3.886..3.888 rows=30 loops=1)
   ->  Foreign Scan on stocks_history_y2019m11  (cost=100.00..110.62 rows=1
width=2020) (actual time=0.259..0.259 rows=0 loops=1)
 Planning Time: 0.119 ms
 Execution Time: 4.966 ms
(5 rows)
```

# PostgreSQL planner on Sharding

```
shard=# EXPLAIN ANALYZE SELECT * FROM stocks_history WHERE
buy_at BETWEEN '2019-10-01 01:00:00' AND '2019-11-01 01:00:00';
           QUERY PLAN
-------------------------------------------------------------------------------
 Append  (cost=100.00..221.25 rows=2 width=2020) (actual time=1.322..1.701 rows=32 loops=1)
   ->  Foreign Scan on stocks_history_y2019m10  (cost=100.00..110.62 rows=1 width=2020)
(actual time=1.320..1.323 rows=31 loops=1)
   ->  Foreign Scan on stocks_history_y2019m11  (cost=100.00..110.62 rows=1 width=2020)
(actual time=0.373..0.373 rows=1 loops=1)
 Planning Time: 0.211 ms
 Execution Time: 2.965 ms

shard=# EXPLAIN ANALYZE SELECT * FROM stocks_history WHERE
buy_at>'2019-10-01 01:00:00' AND buy_at<'2019-11-01 01:00:00' and stock_id=53;
         QUERY PLAN
-------------------------------------------------------------------------------
 Append  (cost=100.00..221.45 rows=2 width=2020) (actual time=0.819..1.071 rows=1 loops=1)
   ->  Foreign Scan on stocks_history_y2019m10  (cost=100.00..110.72 rows=1 width=2020)
(actual time=0.818..0.818 rows=1 loops=1)
   ->  Foreign Scan on stocks_history_y2019m11  (cost=100.00..110.72 rows=1 width=2020)
(actual time=0.251..0.251 rows=0 loops=1)
 Planning Time: 0.131 ms
 Execution Time: 2.074 ms
```

# Sharding concurrent update conflict

```
coord-1=# BEGIN;
BEGIN
coord-1=# UPDATE stocks_history SET stock_currency='Dollar' WHERE stock_id=1;
UPDATE 1

coord-2=# BEGIN;
BEGIN
coord-2=# UPDATE stocks_history SET stock_currency='Rupee' WHERE stock_id=1;
Waiting for lock release by other transaction.....

coord-1=# COMMIT;
COMMIT

coord-2:
ERROR:  could not serialize access due to concurrent update
CONTEXT:  remote SQL command: UPDATE public.stocks_history_y2019m09 SET
stock_currency = 'Rupee'::character(500) WHERE ((stock_id = 1))
NOTE: Need to rollback transaction.
```

**NOTE: Concurrent updates not allowed even on another column value of the same row.**

# Sharding/declarative partitioning limitations

- Declarative partitioning allows more flexibility but needs optimization in locking and planner side
- It is not allowed to CREATE FOREIGN TABLE as a partition of the parent table, if there are UNIQUE indexes on the parent table
- Not possible to use the CONCURRENTLY qualifier when creating a partitioned index on partitioned table
- Manual VACUUM or ANALYZE need to run on each partition table individually

# Auto Sharding

- Database tables get sharded automatically - Makes Everyone Happy!
- Maintenance of large set of sharded tables will be easy
- There are few solutions which provides auto sharding but creates dependency on product/vendor support
- What's the solution?
  We've an answer

# Auto Sharding workaround

Find the workaround at :
https://github.com/kotsachin/pg_auto_shard

- Used postgres_fdw and dblink PostgreSQL extensions
- A shard_config table for data node connection details
- Trigger function - CREATE SERVER and USER MAPPING for each data node after each entry in shard_config table
- EVENT TRIGGERS to table definition on remote data node and respective foreign table definition on the coordinator

# Auto Sharding workaround

```
CREATE TABLE IF NOT EXISTS shard.shard_config(sid int primary key,sname varchar not null,
host varchar, port int not null, dbname varchar not null, password varchar default ''
,unique(sname,host,port,dbname));

CREATE OR REPLACE FUNCTION shard.foreign_server_trigger()
RETURNS TRIGGER AS $$
...
BEGIN
 IF (TG_OP = 'DELETE') THEN
     server_name := (SELECT sname FROM shard.shard_config WHERE sid=OLD.sid);
     cmd := 'DROP USER MAPPING IF EXISTS FOR public SERVER ' || server_name;
     EXECUTE cmd ;
     cmd := 'DROP SERVER IF EXISTS ' || server_name;
     EXECUTE cmd;
     RETURN OLD;
....
$$
LANGUAGE plpgsql;
ALTER FUNCTION shard.foreign_server_trigger() OWNER TO postgres;

CREATE TRIGGER foreign_server_trigger BEFORE UPDATE OR INSERT OR DELETE
 ON shard.shard_config FOR EACH ROW EXECUTE PROCEDURE shard.foreign_server_trigger();
```

# Auto Sharding workaround

```
CREATE EVENT TRIGGER ddl_trigger_create ON ddl_command_end WHEN TAG in ('CREATE
TABLE') EXECUTE PROCEDURE shard.fn_ddl_trigger_create();
```

Fn_ddl_trigger_create:

- Creates table definition on remote host
- Creates respective foreign table to each table created on data nodes
- Creates trigger function to insert data into respective table partition
- Data distribution like Hash partitioning e.g. '(CHECK (' || part_col || ' % ' || shard_count || ' = ' || i '))'

```
CREATE EVENT TRIGGER ddl_trigger_drop ON sql_drop WHEN TAG in ('DROP TABLE')
EXECUTE PROCEDURE shard.fn_ddl_trigger_drop();
```

- Above event trigger function **fn_ddl_trigger_drop** does exactly reverse operation of **fn_ddl_trigger_create** i.e dropping of remote tables and respective database objects.
- This is currently done for inheritance based partition sharding, planning to convert it for declarative based partition sharding soon. Please give your feedback on my mail listed in one of below slide.

# Benefits Sharding in PostgreSQL

- Performance improvement with horizontal scalability
- PostgreSQL is one of the widely used open source RDBMS
- Required feature are under continuous development
- Strong community support for core features
- Easy to handle schema upgrades and migrations

# Some parameters...

**enable_partition_pruning (boolean)**

- Query planner's ability to eliminate a partitioned table's partitions from query plans.
- Generate query plans which allow the query executor to remove partitions

**enable_partitionwise_join (boolean)**

- Query planner's use of partition wise join
- Same data type and have exactly matching sets of child partitions

**enable_partitionwise_aggregate (boolean)**

- Query planner's use of partition wise grouping or aggregation
- GROUP BY clause is not included - only partial aggregation can be performed on a per-partition basis

# Use cases for Sharding

- Stocks history
- Weather forecasting history and analysis
- Population Census
- Patient  record history

and MANY more

2ndQuadrant® PostgreSQL

## Thank you!
## Questions??

**Sachin Kotwal**
**sachin.kotwal@2ndquadrant.com**

**Hari Kiran**
**hari.kiran@2ndquadrant.com**

2ndQuadrant PostgreSQL

# 2ndQuadrant
# Mission Critical Databases

**Website: https://www.2ndquadrant.com/**
**Blog: https://www.2ndquadrant.com/en/blog/**
**Email: info@2ndQuadrant.com**