BDR for DBAs

Martín Marqués

2ndQuadrant

October 23, 2015

2ndQuadrant
Professional PostgreSQL

Content

2ndQuadrant
Professional PostgreSQL

Trigger based Replication

Trigger based Replication

- Trigger based

Trigger based Replication

- Trigger based (Slony, Londiste)

Trigger based Replication

- Trigger based (Slony, Londiste)
  - Used since 7.3

2ndQuadrant
Professional PostgreSQL

Trigger based Replication

- Trigger based (Slony, Londiste)
  - Used since 7.3
  - Trigger based (double writes on upstream)

Trigger based Replication

- Trigger based (Slony, Londiste)
  - Used since 7.3
  - Trigger based (double writes on upstream)
  - Needs external daemon

2ndQuadrant
Professional PostgreSQL

Trigger based Replication

- Trigger based (Slony, Londiste)
  - Used since 7.3
  - Trigger based (double writes on upstream)
  - Needs external daemon
  - No DDL replication

Stream Replication

- Stream Replication

2ndQuadrant
Professional PostgreSQL

Stream Replication

- Stream Replication
  - Available since 9.0

2ndQuadrant
Professional PostgreSQL

Stream Replication

- Stream Replication
  - Available since 9.0
  - One way replication

2ndQuadrant
Professional PostgreSQL

Stream Replication

- Stream Replication
  - Available since 9.0
  - One way replication
  - Hot Standby with 9.1

2ndQuadrant
Professional PostgreSQL

Stream Replication

- Stream Replication
  - Available since 9.0
  - One way replication
  - Hot Standby with 9.1
  - Can’t replicate per-DB

2ndQuadrant
Professional PostgreSQL

Replication Slots

- Logical decoding

Replication Slots

- Logical decoding
  - Available since 9.4

Replication Slots

- Logical decoding
  - Available since 9.4
  - Good option for trigger based replication tools

2ndQuadrant
Professional PostgreSQL

Replication Slots

- Logical decoding
  - Available since 9.4
  - Good option for trigger based replication tools
  - BDR, UDR

2ndQuadrant
Professional PostgreSQL

Content

2ndQuadrant
Professional PostgreSQL

Bi-Directional Replication

2ndQuadrant
Professional PostgreSQL

Bi-Directional Replication

- Per-database replication

2ndQuadrant
Professional PostgreSQL

Bi-Directional Replication

- Per-database replication
- Replication sets

Bi-Directional Replication

- Per-database replication
- Replication sets
- Eventually consistent

2ndQuadrant
Professional PostgreSQL

Bi-Directional Replication

- Per-database replication
- Replication sets
- Eventually consistent (asynchronous replication)

2ndQuadrant
Professional PostgreSQL

Bi-Directional Replication

- Per-database replication
- Replication sets
- Eventually consistent (asynchronous replication)
- Conflict resolution

2ndQuadrant
Professional PostgreSQL

Bi-Directional Replication

- Per-database replication
- Replication sets
- Eventually consistent (asynchronous replication)
- Conflict resolution $\Rightarrow$ last update wins

2ndQuadrant
Professional PostgreSQL

Bi-Directional Replication

- Per-database replication
- Replication sets
- Eventually consistent (asynchronous replication)
- Conflict resolution $\Rightarrow$ last update wins $\Rightarrow$ user-defined conflict handlers

Bi-Directional Replication

- Per-database replication
- Replication sets
- Eventually consistent (asynchronous replication)
- Conflict resolution $\Rightarrow$ last update wins $\Rightarrow$ user-defined conflict handlers
- DDL replication

2ndQuadrant
Professional PostgreSQL

Bi-Directional Replication

- Per-database replication
- Replication sets
- Eventually consistent (asynchronous replication)
- Conflict resolution $\Rightarrow$ last update wins $\Rightarrow$ user-defined conflict handlers
- DDL replication
  - Keeps schema syncd

2ndQuadrant
Professional PostgreSQL

Bi-Directional Replication

- Per-database replication
- Replication sets
- Eventually consistent (asynchronous replication)
- Conflict resolution $\Rightarrow$ last update wins $\Rightarrow$ user-defined conflict handlers
- DDL replication
  - Keeps schema syncd
  - ... even objects not in replication sets

2ndQuadrant
Professional PostgreSQL

Bi-Directional Replication

- Per-database replication
- Replication sets
- Eventually consistent (asynchronous replication)
- Conflict resolution $\Rightarrow$ last update wins $\Rightarrow$ user-defined conflict handlers
- DDL replication
  - Keeps schema syncd
  - ... even objects not in replication sets
  - DDL locks

2ndQuadrant
Professional PostgreSQL

Bi-Directional Replication

- Per-database replication
- Replication sets
- Eventually consistent (asynchronous replication)
- Conflict resolution $\Rightarrow$ last update wins $\Rightarrow$ user-defined conflict handlers
- DDL replication
  - Keeps schema syncd
  - ... even objects not in replication sets
  - DDL locks
  - Global lock will cancel queries

2ndQuadrant
Professional PostgreSQL

Bi-Directional Replication

- Per-database replication
- Replication sets
- Eventually consistent (asynchronous replication)
- Conflict resolution ⇒ last update wins ⇒ user-defined conflict handlers
- DDL replication
  - Keeps schema syncd
  - ... even objects not in replication sets
  - DDL locks
  - Global lock will cancel queries
  - Not all DDLs are replicated

2ndQuadrant
Professional PostgreSQL

Bi-Directional Replication

- Replicates in both directions

Bi-Directional Replication

- Replicates in both directions
- Mesh topology

2ndQuadrant
Professional PostgreSQL

Bi-Directional Replication

- Replicates in both directions
- Mesh topology
- All bdr objects will be stored in the **bdr** schema

2ndQuadrant
Professional PostgreSQL

BDR postgresql.conf options

- shared_preload_libraries = 'bdr'
- wal_level = 'logical'
- track_commit_timestamp = on
- max_wal_senders = 10
- max_replication_slots = 10
- max_worker_processes = 10

2ndQuadrant
Professional PostgreSQL

BDR extensions

```
CREATE EXTENSION btree_gist;
CREATE EXTENSION bdr;
```

2ndQuadrant
Professional PostgreSQL

Content

2ndQuadrant
Professional PostgreSQL

Creating BDR node

On the first node at **Santa Fe** we create the BDR group

```
SELECT bdr.bdr_group_create(
          node_name := 'santafe',
          node_external_dsn := 'dbname=personal host=192.168.5.18 user=postgres port=5432',
          replication_sets := ARRAY['default']);
```

2ndQuadrant
Professional PostgreSQL

Creating BDR node

On the first node at **Santa Fe** we create the BDR group

```
SELECT bdr.bdr_group_create(
          node_name := 'santafe',
          node_external_dsn := 'dbname=personal host=192.168.5.18 user=postgres port=5432',
          replication_sets := ARRAY['default']);


SELECT bdr.bdr_node_join_wait_for_ready();
```

2ndQuadrant
Professional PostgreSQL

Creating BDR node

We can now join nodes with **bdr_init_copy**

Creating BDR node

We can now join nodes with **bdr_init_copy**

```
$ hostname
rosario
$ bdr_init_copy -D ~postgres/9.4-bdr/data -n rosario \
  -d "dbname=personal host=192.168.5.18 user=postgres port=5432" \
  --local-dbname="dbname=personal host=192.168.5.19 user=postgres port=5432"
```

2ndQuadrant
Professional PostgreSQL

Creating BDR node

... and on some other geographiclly remote DC

Creating BDR node

... and on some other geographiclly remote DC

```
$ hostname
cordoba
$ bdr_init_copy -D ~postgres/9.4-bdr/data -n cordoba \
  -d "dbname=personal host=192.168.5.18 user=postgres port=5432" \
  --local-dbname="dbname=personal host=192.168.5.20 user=postgres port=5432"
```

2ndQuadrant
Professional PostgreSQL

Creating BDR node

Check the nodes in bdr.bdr_nodes

```
personal=# select * from bdr.bdr_nodes;
-[ RECORD 1 ]------+----------------------------------------------------
node_sysid         | 6182129686351217602
node_timeline      | 1
node_dboid         | 16389
node_status        | r
node_name          | santafe
node_local_dsn     | dbname=personal host=192.168.5.18 user=postgres port=5432
node_init_from_dsn |
-[ RECORD 2 ]------+----------------------------------------------------
node_sysid         | 6182167093031105444
node_timeline      | 2
node_dboid         | 16389
node_status        | c
node_name          | rosario
node_local_dsn     | user=postgres host=192.168.5.19 port=5432 dbname=personal
node_init_from_dsn | user=postgres host=192.168.5.18 port=5432 dbname=personal
-[ RECORD 3 ]------+----------------------------------------------------
node_sysid         | 6182181726884158214
node_timeline      | 2
node_dboid         | 16389
node_status        | r
node_name          | cordoba
node_local_dsn     | user=postgres host=192.168.5.20 port=5432 dbname=personal
node_init_from_dsn | user=postgres host=192.168.5.18 port=5432 dbname=personal
```

2ndQuadrant
Professional PostgreSQL

Content

2ndQuadrant
Professional PostgreSQL

Sequences

2ndQuadrant
Professional PostgreSQL

Sequences

Let's try using normal PG sequences

Sequences

Let's try using normal PG sequences
On node 1:

```
CREATE TABLE my_rep_table (
  id   SERIAL PRIMARY KEY,
  val  TEXT NOT NULL
);

INSERT INTO my_rep_table VALUES (default,'some value');
```

2ndQuadrant
Professional PostgreSQL

Sequences

On node 2:

```
SELECT * FROM my_rep_table;
 id |     val
----+------------
  1 | some value
(1 row)
```

2ndQuadrant
Professional PostgreSQL

Sequences

On node 2:

```
SELECT * FROM my_rep_table;
 id |    val
----+------------
  1 | some value
(1 row)


INSERT INTO my_rep_table VALUES (default,'some other value');
ERROR:  duplicate key value violates unique constraint «my_rep_table_pkey»
DETAIL:  Key (id)=(1) already exists.
```

2ndQuadrant
Professional PostgreSQL

Global Sequences

2ndQuadrant
Professional PostgreSQL

Global Sequences

- Sequences on each node have a chunk of values to use

Global Sequences

- Sequences on each node have a chunk of values to use
- Chunks are all disjoint

2ndQuadrant
Professional PostgreSQL

Global Sequences

- Sequences on each node have a chunk of values to use
- Chunks are all disjoint
- Don’t expect ascending values

2ndQuadrant
Professional PostgreSQL

Global Sequences

- Sequences on each node have a chunk of values to use
- Chunks are all disjoint
- Don't expect ascending values (max(id) is not the last inserted value from the sequence)

2ndQuadrant
Professional PostgreSQL

Global Sequences

- What happens when all values from the chunk are used?

2ndQuadrant
Professional PostgreSQL

Global Sequences

- What happens when all values from the chunk are used?
  - Voting

2ndQuadrant
Professional PostgreSQL

Global Sequences

- What happens when all values from the chunk are used?
  - Voting (which needs quorum of half + 1 nodes)

2ndQuadrant
Professional PostgreSQL

Global Sequences

- What happens when all values from the chunk are used?
  - Voting (which needs quorum of half + 1 nodes)
  - New chunk of values not used will be assigned to the sequence

2ndQuadrant
Professional PostgreSQL

Global Sequences

- What happens with a disconnected node?

2ndQuadrant
Professional PostgreSQL

Global Sequences

- What happens with a disconnected node?
  - When node reconnects, voting information is replicated

2ndQuadrant
Professional PostgreSQL

Global Sequences

- What happens with a disconnected node?
  - When node reconnects, voting information is replicated
  - If it needs values for the same seq a new voting will take place

2ndQuadrant
Professional PostgreSQL

Content

2ndQuadrant
Professional PostgreSQL

Prohibited DDL

- `CREATE TABLE AS`

Prohibited DDL

- `CREATE TABLE AS`
- `ALTER TABLE ... ADD COLUMN ... DEFAULT ...`

2ndQuadrant
Professional PostgreSQL

Prohibited DDL

- `CREATE TABLE AS`
- `ALTER TABLE ... ADD COLUMN ... DEFAULT ...`
- `CREATE MATERIALIZED VIEW ...`
  `REFRESH MATERIALIZED VIEW ...`

2ndQuadrant
Professional PostgreSQL

Workaround Prohibited DDL

```
CREATE TABLE my_new_table AS <query>
```

2ndQuadrant
Professional PostgreSQL

Workaround Prohibited DDL

```
CREATE TABLE my_new_table AS <query>

BEGIN;
SET LOCAL bdr.permit_unsafe_ddl_commands = true;
CREATE TABLE my_new_table AS <query> NO DATA;
END;
UPDATE my_new_table ...
```

2ndQuadrant
Professional PostgreSQL

Workaround Prohibited DDL

```
ALTER TABLE ... ADD COLUMN ... DEFAULT <some_value>
```

2ndQuadrant
Professional PostgreSQL

Workaround Prohibited DDL

```
ALTER TABLE ... ADD COLUMN ... DEFAULT <some_value>

BEGIN;
ALTER TABLE my_table ADD COLUMN my_col <type>;
END;
UPDATE my_table set my_col = <some_value>
```

2ndQuadrant
Professional PostgreSQL

Workaround Prohibited DDL

```
CREATE MATERIALIZED VIEW ...
REFRESH MATERIALIZED VIEW ...
```

2ndQuadrant
Professional PostgreSQL

Workaround Prohibited DDL

```
CREATE MATERIALIZED VIEW ...
REFRESH MATERIALIZED VIEW ...

BEGIN;
SET LOCAL bdr.permit_unsafe_ddl_commands = true;
SET LOCAL bdr.skip_ddl_replication = true;
SET LOCAL bdr.skip_ddl_locking = true;
CREATE MATERIALIZED VIEW ...
END;
```

2ndQuadrant
Professional PostgreSQL

Workaround Prohibited DDL

```
CREATE MATERIALIZED VIEW ...
REFRESH MATERIALIZED VIEW ...

BEGIN;
SET LOCAL bdr.permit_unsafe_ddl_commands = true;
SET LOCAL bdr.skip_ddl_replication = true;
SET LOCAL bdr.skip_ddl_locking = true;
CREATE MATERIALIZED VIEW ...
END;

PGOPTIONS='-c bdr.do_not_replicate=on \
           -c bdr.permit_unsafe_ddl_commands=on \
           -c bdr.skip_ddl_replication=on \
           -c bdr.skip_ddl_locking=on' \
ON_ERROR_STOP=1 psql -c "REFRESH MATERIALIZED VIEW ..."
END;
```

2ndQuadrant
Professional PostgreSQL

Content

2ndQuadrant
Professional PostgreSQL

Conclusions

2ndQuadrant
Professional PostgreSQL

Conclusions

- Trigger based replication is dead

2ndQuadrant
Professional PostgreSQL

Conclusions

- Trigger based replication is dead
- Prepare to execute again your queries

2ndQuadrant
Professional PostgreSQL

Conclusions

- Trigger based replication is dead
- Prepare to execute again your queries
- Doesn't scale writes

2ndQuadrant
Professional PostgreSQL

Conclusions

- Trigger based replication is dead
- Prepare to execute again your queries
- Doesn't scale writes
- You have to unlearn old habits when using Global Sequences

2ndQuadrant
Professional PostgreSQL

Conclusions

- Trigger based replication is dead
- Prepare to execute again your queries
- Doesn't scale writes
- You have to unlearn old habits when using Global Sequences
- Works very well on replication clusters with little or no DDL execution

2ndQuadrant
Professional PostgreSQL

Conclusions

- Trigger based replication is dead
- Prepare to execute again your queries
- Doesn't scale writes
- You have to unlearn old habits when using Global Sequences
- Works very well on replication clusters with little or no DDL execution
- Don't mix DDL execution with DML statements in the same transaction

2ndQuadrant
Professional PostgreSQL

Conclusions

- Trigger based replication is dead
- Prepare to execute again your queries
- Doesn't scale writes
- You have to unlearn old habits when using Global Sequences
- Works very well on replication clusters with little or no DDL execution
- Don't mix DDL execution with DML statements in the same transaction
- Be extremely careful with using **bdr.permit_unsafe_ddl_commands=on**

2ndQuadrant
Professional PostgreSQL

Content

2ndQuadrant
Professional PostgreSQL

Questions?

2ndQuadrant
Professional PostgreSQL