Intro to BDR

Ron Privett | Customer Success Solution Engineer

September 30, 2021

EDB

What brings us here today?

- BDR History & Overview
- Core BDR features
- Conflict Resolution concepts
- Conflict-free replicated datatypes
- Demo
- Q&A

2021 Copyright © EnterpriseDB Corporation All Rights Reserved

BDR History & Overview

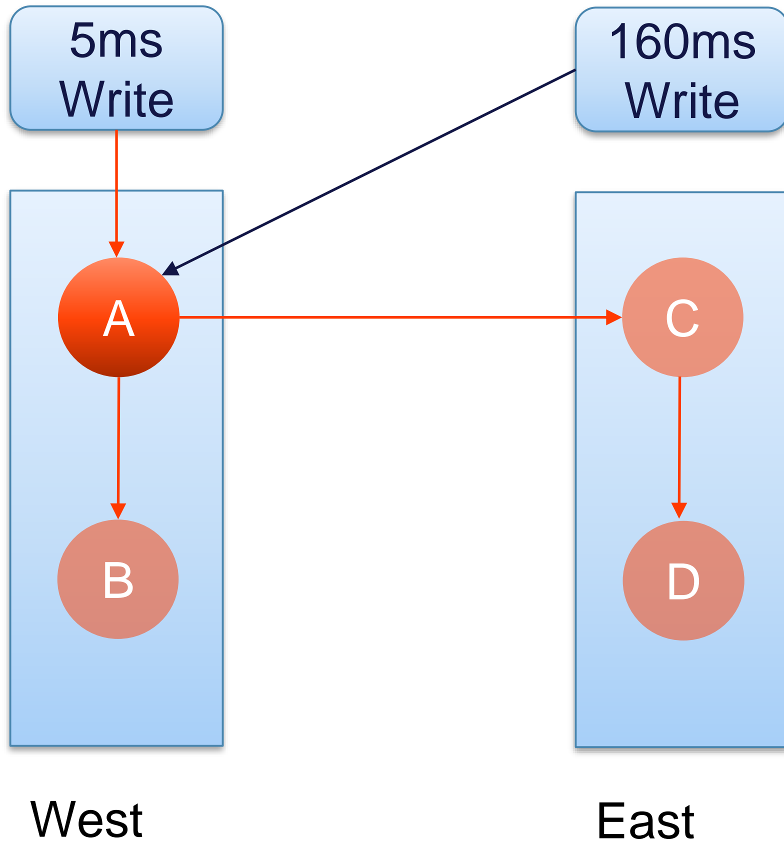
EDB

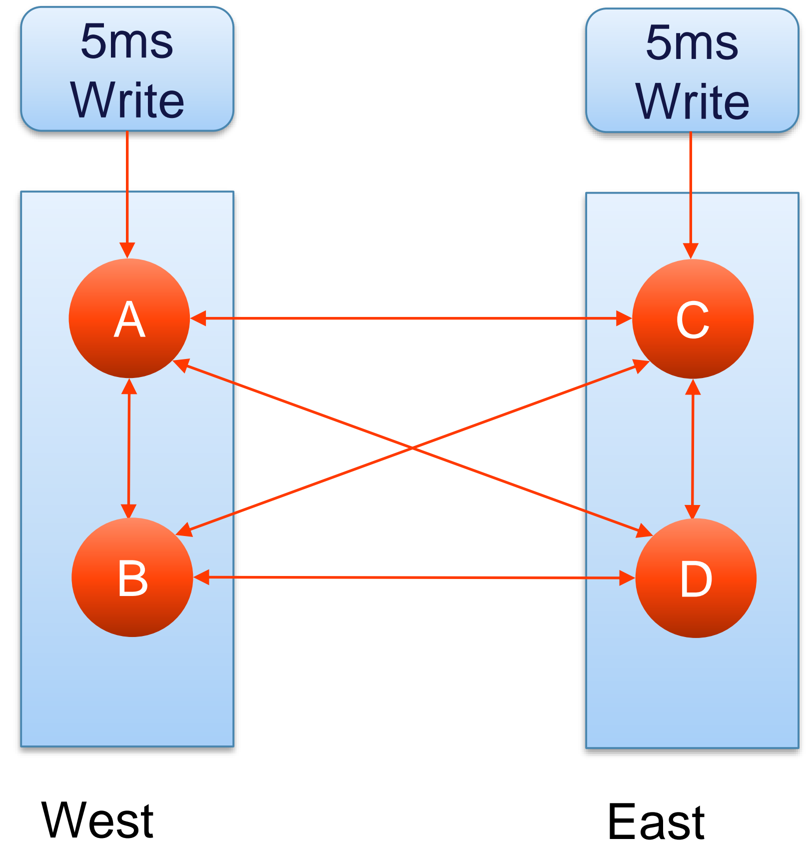
BDR Introduction

What is BDR?

- BDR stands for Bi-Directional Replication (Logical Replication to all nodes)
- Benefits:
  - Scale-Out
  - High Availability
  - Low latency
  - Geo-Redundancy
  - Online upgrades and maintenance operations

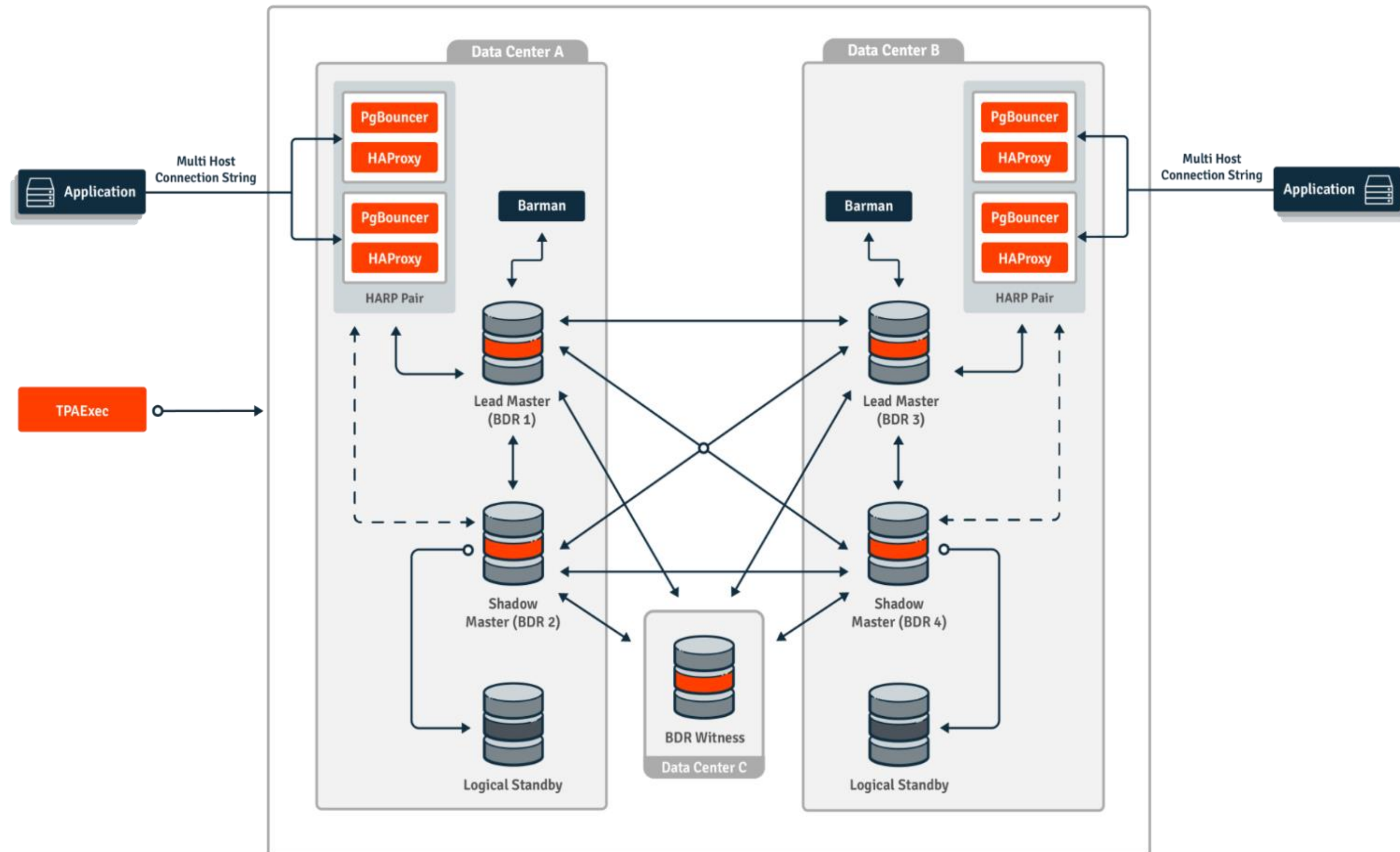
Single Master vs Multi Master





Very High Availability with BDR

- Multi-master/active-active
- Multiple data centers (DCs)
- Application access to Lead Node, fast failover to Shadow Node for **99.999% Availability**
- Tolerant of down nodes, automatically recovers from node outage and network partition



EDB

Origins & History of Replication

- 2004 – Trigger-based Replication: Slony, Londiste
- 2006 – Physical File-based Replication PG8.2
- 2010 – Physical Streaming Replication PG9.0
- 2011 – BDR detailed design started
- 2012 – xDB 5.0 released
- 2014 – BDR 1 released, deprecated 2019
  - Fork of PostgreSQL 9.4
  - Invented logical replication without triggers, now in PostgreSQL 10 core
- 2017 – Logical Streaming Replication PG10
- 2018 – BDR 2 released, deprecated 2020
  - Extension of PostgreSQL 9.6
- 2019 – BDR 3 released
  - Major source code rewrite
  - Extension of PostgreSQL 10+
- 8/2021 – BDR 3.7 (current version)

EDB

BDR Editions (current)

Extensions on top of the pglogical 3 extension

Standard Edition

- On top of community Postgres 10+, and EDB Postgres Advanced Server (EPAS) v11+
- Essential multi-master replication
- Replication sets
- Conflict detection and resolution
- Monitoring capabilities

Enterprise Edition

- On top of EDB Postgres Extended (formerly called 2ndQPostgres 10+), and EDB Postgres Advanced Server (EPAS) v11+
- Includes everything in BDR Standard Edition
- Additional features for global workloads (see next slide)

EDB

Additional Postgres-BDR™ features

- Multi-origin PITR (Point In Time Recovery)
- CAMO (Commit At Most Once)
- Timestamp-based Snapshots
- Eager Replication
- CRDTs (Conflict-free Replicated Data Types)
- Column Level Conflict Detection and Resolution
- Transform Triggers
- Conflict Triggers

EDB

BDR Editions (future)

There used to be 2 Editions (Standard & Enterprise), but soon there will be just one

Postgres-BDR ™

- Full-featured replication solution providing the essential multi-master replication capabilities for PostgreSQL clusters with advanced conflict management and data-loss protection, and up to 5X faster throughput
- Enables application and database upgrades without requiring downtime
- Provides row-level last-update wins eventual consistency by default
- Tools to monitor operation and verify data consistency
- Guard application transactions with commit-at-most-once consistency even in the presence of node failures

BDR Releases 2021

BDR is now the strategic Postgres clustering technology from EDB, used by many customers

PostgreSQL

- BDR v3.6.27 runs on PostgreSQL v10-11 [Aug 2021]
- **BDR v3.7.12 runs on PostgreSQL v11-13** [Sep 2021]
- BDR v4.0.0 will run on PostgreSQL v12-14 [4Q2021]
- New major release annually

EDB Postgres Extended

- BDR v3.6.27 runs on Postgres Extended v11 [Aug 2021]
- **BDR v3.7.12 runs on Postgres Extended v11-13** [Sep 2021]
- BDR v4.0.0 will run on Postgres Extended v12-14 [4Q2021]
- New major release every 6 months

EDB Postgres Advanced

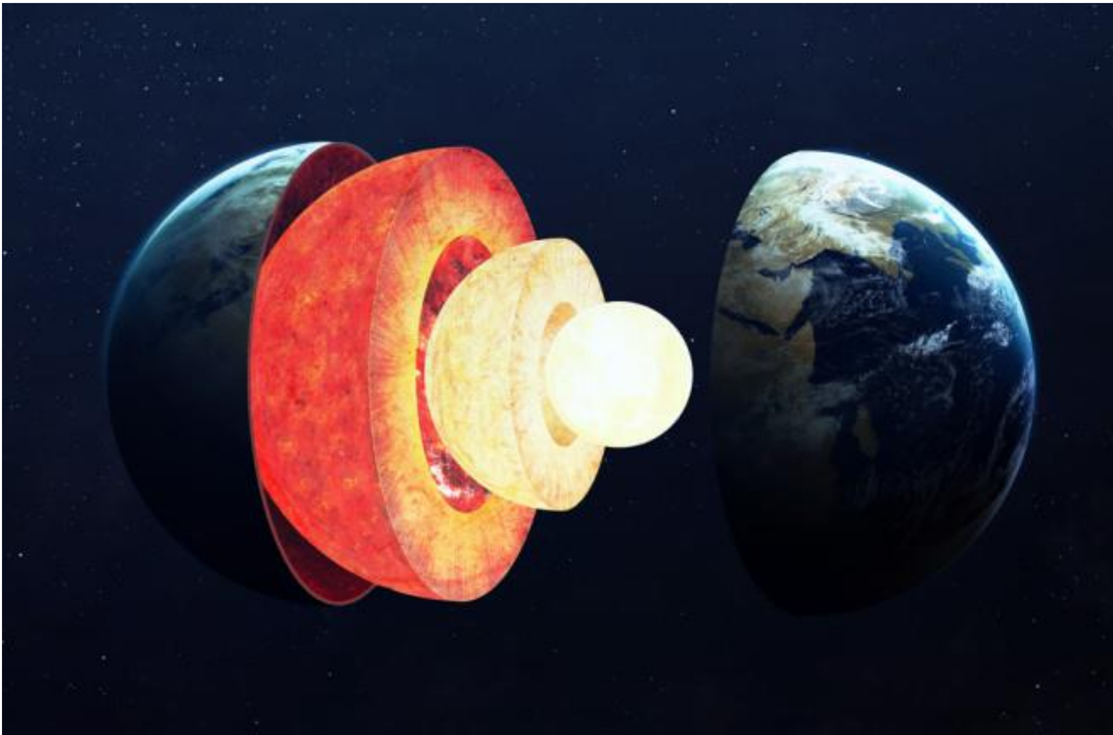
- **BDR v3.7.12 runs on Postgres Advanced v11-13** [Sep 2021]
- BDR v4.0.0 will run on Postgres Advanced v12-14 [4Q2021]
- New major release annually

2021 Copyright © EnterpriseDB Corporation All Rights Reserved

Core BDR Features

EDB

Core BDR Features



- Active-Active mesh network of nodes
- Asynchronous replication with row-level last-update wins eventual consistency
- Synchronous replication
- Automatic DDL and DML replication
- Rolling database upgrades
- Sub-groups with subscribe-only nodes
- Sequence management
- Tools to monitor and verify data consistency
- Supports v11, 12, & 13

EDB

BDR Essential Features

Feature	PostgreSQL	Postgres Extended	Postgres Advanced
Rolling application and database upgrades to address the largest source of downtime	✔	✔	✔
Asynchronous replication with origin based row-level last-update wins eventual consistency	✔	✔	✔
Synchronous replication (PostgreSQL core is not conflict-free)	✔	✔	✔
DDL replication with granular locking supports changes to application schema, ideal for use in continuous release environments	✔	✔	✔
Sub-groups with subscribe-only nodes enable data distribution use cases for applications with very high read scaling requirements *	✔	✔	✔
Sequence handling provides applications different options for generating unique surrogate ids that are multi-node aware	✔	✔	✔
Tools to monitor operation and verify data consistency	✔	✔	✔
Supports versions 11, 12, 13 *	✔	✔	✔

* New with 3.7

EDB

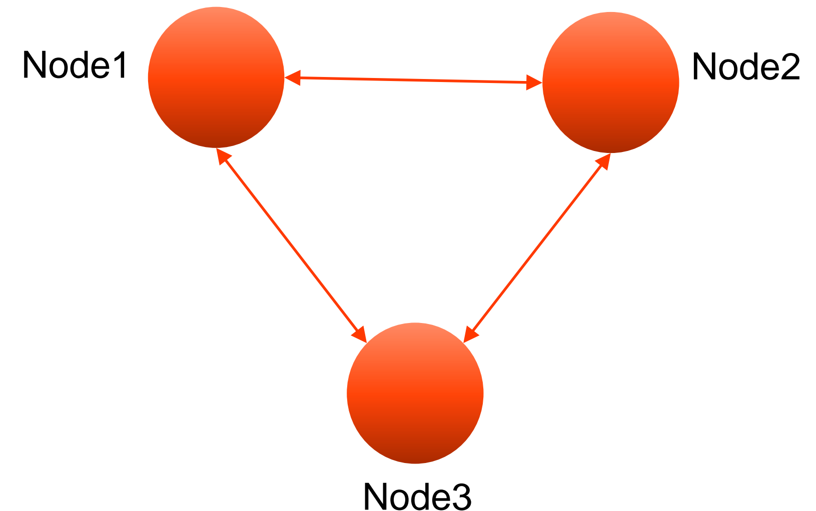
BDR Advanced Features

Feature	PostgreSQL	Postgres Extended	Postgres Advanced
Parallel apply allows multiple writer processes to apply transactions on the downstream node improving throughput up to 5X *	(v14+)	✔	✔
Conflict-free replicated data types (CRDTs) provide mathematically proven consistency in asynchronous multi-master update scenarios	(v14+)	✔	✔
Column-level conflict resolution enables per column last-update wins resolution to merge updates	(v14+)	✔	✔
Transform triggers execute on incoming data for modifying or advanced programmatic filtering	(v14+)	✔	✔
Conflict triggers provide custom resolution techniques when a conflict is detected	(v14+)	✔	✔
Eager replication provides conflict free replication by synchronizing across cluster nodes before committing a transaction		✔	(v14+)
Commit at most once (CAMO) consistency guards application transactions even in the presence of node failures		✔	(v14+)
Single decoding worker improves performance on upstream node by doing logical decoding of WAL once instead of for each downstream node *		v13+	(v14+)
Tooling to assess applications for distributed database suitability		✔	(v14+)

* New with 3.7

Todays Demo Environment

- Ubuntu
- Docker (installed within the Linux host)
- TPAexec (used to deploy docker containers configured with BDR)
- BDR-Simple = 3 BDR nodes, 1 instance per container
- All nodes running EPAS 13



Concepts of Conflict Resolution

Inter-node conflicts arise as a result of sequences of events that could not happen if all the involved transactions happened concurrently on the same node. Because the nodes only exchange changes after the transactions commit, each transaction is individually valid on the node it committed on, but would not be valid if applied on another node that did other conflicting work at the same time.

Excerpt from the wonderful BDR Documentation
https://www.enterprisedb.com/docs/bdr/latest/conflicts/#how-conflicts-happen

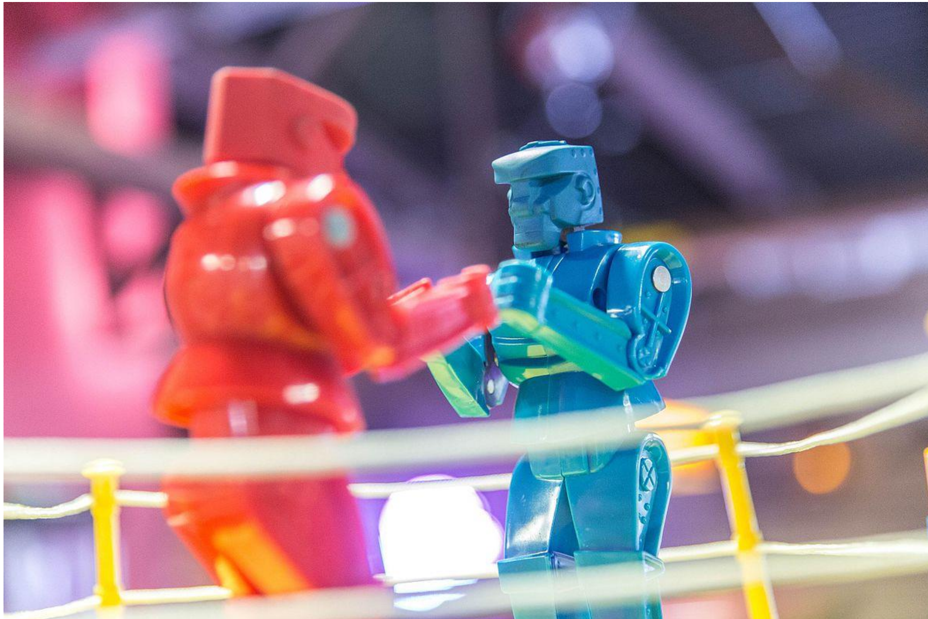
EDB

Basic Concepts of Conflict Resolution

- Conflicts are not errors
- Because BDR is Asynchronous (by default) conflicts happen
- BDR automatically detects conflicts, and resolves most conflicts automatically
- Conflicts are logged to the PostgreSQL log by default, they can also be logged into a table
- Resolution at the row level, by applying the last transaction on both sides
  - Requires `track_commit_timestamp = on` in `postgresql.conf`
  - Timestamp resolution requires clocks on all BDR servers in sync with each other
  - Resolution should end with all nodes storing the same values
- When appropriate for your BDR use case, conflicts can be avoided by using CRDT’s and Eager Replication

EDB

Conflict Handling



- Conflict handling is configurable
- Conflicts can be detected and handled differently for each table using Conflict Triggers (part of Column-Level Conflict Resolution)
- Possible values for `conflict_resolution` in `bdr.apply_log`:
  - skip = the processing of the row was skipped (no change has been made locally)
  - apply_remote = the remote (incoming) row has been applied
  - merge = a new row was created, merging information from remote and local row
  - user = user code (a conflict trigger) has produced the row that was written to the target table
- Determine your current behavior:
  - `SELECT * FROM bdr.node_conflict_resolvers;`

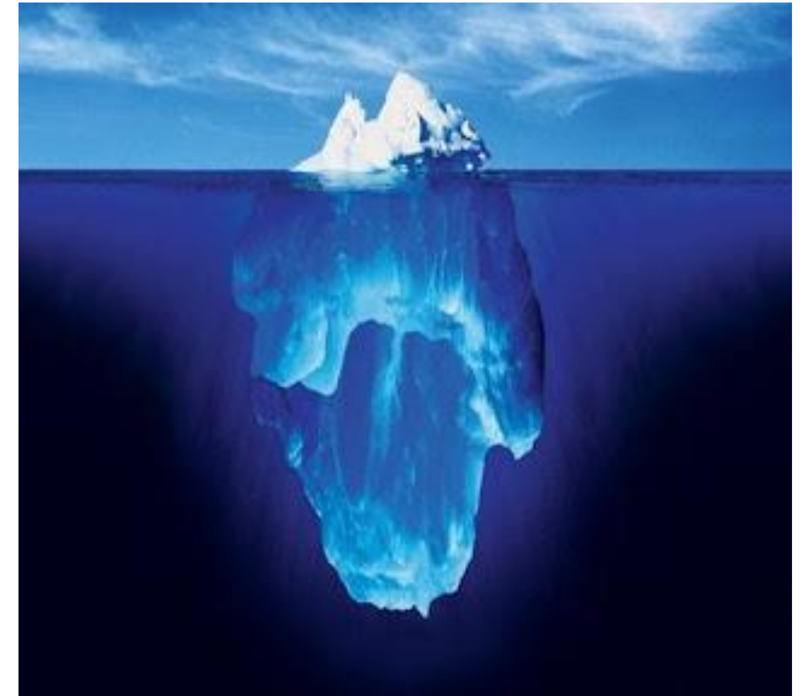
EDB

Be careful out there

There are other cases when conflicts can occur – such as:

- Multiple Unique/PK constraints
- Concurrent PK updates
- Node down / network issues / replication lag
- Vigorous Vacuum activity
- FK Constraint Conflicts
- Truncate Conflicts
- Exclusion Constraints Conflicts
- WAL logging of TOAST values
- ABA problem (https://en.wikipedia.org/wiki/ABA_problem)

Be aware some operations which work fine outside of replication, may cause issues when replication is enabled



Conflict-free replicated data types (CRDTs)

EDB

CRDTs – Use Case

Ensuring all Increments or Decrements are applied

- In this example, the following SQLs were run at the same time on different nodes :
  - AMT column for id=1 on all nodes = 100
  - Node 1: `UPDATE accounts SET amt = amt + 100 WHERE id=1;`
  - Node 2: `UPDATE accounts SET amt = amt + 75 WHERE id=1;`
- What would the new AMT be after this scenario? 275 (100+100+75), or something else?
- Without CDRTs – you might get 200, or 175 (depending) and not 275
- CRDTs ensure all increments (or decrements) are applied

EDB

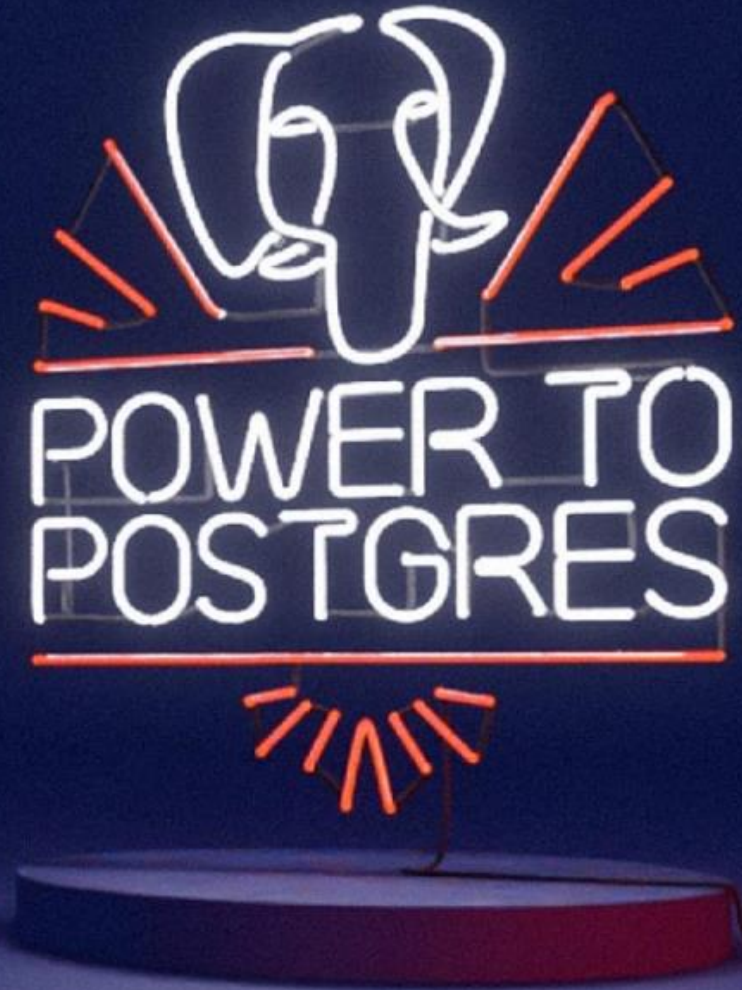
CRDTs - Details

- Traditional conflict resolution discards one of the rows
  - `conflict_resolution` = `skip` or `apply_remote`
- CRDTs merge values from concurrently modified rows
  - `conflict_resolution = merge`
- Requires Column-Level Conflict Resolution enabled on the table
  - Column-level conflict resolution requires the table to have `REPLICA IDENTITY FULL`. The bdr.alter_table_conflict_detection function does a check, and will fail with an error otherwise.
- Several data types, for example `bdr.crdt_delta_counter`, which behaves like BIGINT
  - All types are `BIGINT` or `NUMERIC`
- List of CRDTs:
  - bdr.crdt_gcounter, bdr.crdt_gsum,
  - bdr.crdt_pncounter, bdr.crdt_pnsum,
  - bdr.crdt_delta_counter, bdr.crdt_delta_sum

Demos

Demos

Q&A



Thank you!

Ron.Privett@enterprisedb.com

EDB

EDB

Basic Usage - Demo

Replicate basic DDL and DML in a BDR cluster

- Create table on node 1 and 2:
  - Node1: `CREATE TABLE tbln1 (id INTEGER PRIMARY KEY, msg TEXT);`
  - Node2: `CREATE TABLE tbln2 (id INTEGER PRIMARY KEY, msg TEXT);`
- Check tables on all nodes to confirm creation:
  - `\dt public.*`
  - `\d+ tbln1`
  - `\d+ tbln2`
- Replicate Alter Table statement:
  - Node1: `ALTER TABLE tbln1 RENAME COLUMN msg TO cnt;`

EDB

Basic Usage - Demo

Replicate basic DDL and DML in a BDR cluster

- Check that the column name was changed on all nodes:
  - `\d+ tbln1`
- Run DML statement with non-concurrent transactions
  - Node1: `INSERT INTO tbln1 VALUES (1, 'N1_AAA');`
  - Node1: `SELECT * FROM tbln1;`
  - Node2: `INSERT INTO tbln1 VALUES (2, ‘N2_BBB');`
  - Node2: `SELECT * FROM tbln1;`
- Both rows should be on all servers

EDB

Basic Usage - Demo

Replicate basic DDL and DML in a BDR cluster

- Run DML statement with eventual consistency
  - Node1: run and keep transaction open

    ```
    BEGIN;INSERT INTO tbln1 VALUES (3,
    'N1_BBB');SELECT * FROM tbln1;
    ```

  - Node2: verify insert has not replicated yet

    ```
    SELECT * FROM tbln1;
    ```

  - Node1: commit the transaction

    ```
    COMMIT;
    ```

  - Node2: verify the insert has been replicated

    ```
    SELECT * FROM tbln1;
    ```

- Bonus - Check row origin on all nodes:

  ```
  SELECT t.*, o.roname
  FROM tbln1 t
  LEFT JOIN pg_replication_origin o
  ON o.roident =
  bdr.pg_xact_origin(t.xmin);
  ```

EDB

Basics of Conflict Resolution – Demo

Scenario: Update of same row on two different servers

- On node1, create a table:
```
CREATE TABLE tbl (id INTEGER PRIMARY KEY, msg TEXT);
```
- On node1, start a transaction, insert 1 row, don’t commit:
```
BEGIN;
INSERT INTO tbl VALUES (1, 'N1_AAA');
SELECT * FROM tbl;
```
- On node2, insert a new row with the same PK value:
```
INSERT INTO tbl VALUES (1, 'N2_BBB');
SELECT * FROM tbl;
```

Basics of Conflict Resolution – Demo

Scenario: Update of same row on two different servers

- Check node3 to confirm replicated row
- On node1, verify that the insert from node2, which was done after the transaction was opened, is NOT visible from the current transaction
- The transaction from node2 is waiting on a row-level lock on node1
- Commit the transaction on node1 – What Happened?

```
bdrdb=# commit;
COMMIT
bdrdb=# select * from tbl;
 id |   msg
----+---------
  1 | N1_AAA
(1 row)
```

```
bdrdb=# select * from tbl;
 id |   msg
----+---------
  1 | N1_AAA
(1 row)
```

```
bdrdb=# select * from tbl;
 id |   msg
----+---------
  1 | N1_AAA
(1 row)
```

Basics of Conflict Resolution – Demo

- A conflict happened on the PK, and the last committed transaction (from node1) won on all nodes
- Logs from node1:

Sep 30 00:06:37 node1 postgres[4651]: [117-1] 2021-09-30 00:06:37 UTC [enterprisedb@/pglogical writer 17072:148744716/bdrdb:4651]: [321] LOG: **CONFLICT: insert_exists on relation "public.tbl"; resolution: skip; resolver: update_if_newer.**

Sep 30 00:06:37 node1 postgres[4651]: [117-2] 2021-09-30 00:06:37 UTC [enterprisedb@/pglogical writer 17072:148744716/bdrdb:4651]: [322] **DETAIL: remote tuple origin=2,timestamp=2021-09-30 00:05:36.519732+00,commit_lsn=0/3CCF298**

Sep 30 00:06:37 node1 postgres[4651]: [117-3] 2021-09-30 00:06:37 UTC [enterprisedb@/pglogical writer 17072:148744716/bdrdb:4651]: [323] **CONTEXT: during apply of INSERT from remote relation public.tbl in xact with commit-end lsn 0/3CCF298 xid 8985 committs 2021-09-30 00:05:36.519732+00 (action #2) (effective sess origin id=2 lsn=0/3CCF298)**

- Logs from node2 are similar

EDB

Basics of Conflict Resolution – Monitoring

- Be aware of Log Files (or table) on each node:
    - The type of *conflict*: **insert_exists**
    - The *resolver*: **update_if_newer**
    - The *resolution*:
        - **skip** (on node1)
        - **apply_remote** (on node2)

EDB

CRDTs – Function overview – Demo

- Create a table on Node 1:
```
CREATE TABLE tbl_crdt (id INTEGER PRIMARY KEY, val1 bdr.crdt_delta_counter, val2 INTEGER);
```
- Enable Column-Level Conflict Resolution on the table:
```
ALTER TABLE tbl_crdt REPLICA IDENTITY FULL;
SELECT bdr.alter_table_conflict_detection(
  relation     := 'tbl_crdt',
  method       := 'column_modify_timestamp',
  column_name := 'cts'
);
```
- Check table schema:
```
bdrdb=# \d+ tbl_crdt
```
- The column `val1` is of type `bdr.crdt_delta_counter`. Also a new column `cts` of type `bdr.column_timestamps` was added, as well as a couple of triggers.

EDB

CRDTs – Function overview – Demo

- On Node 1 – Add an initial row to the table:
```
INSERT INTO tbl_crdt VALUES (2, 0, 0);
```
- Check current row:
```
SELECT id, val1, val2 FROM tbl_crdt;
id | val1 | val2
----+------+------
  2 |    0 |    0
(1 row)
```
- With this environment set up, test the functionality

EDB

CRDTs – Function overview – Demo

- On node1, open a transaction, run an UPDATE to change `val1`, and keep the transaction open:

```
BEGIN;
  UPDATE tbl_crdt SET val1 = val1 + 10 WHERE id = 2;
```

- On node2, run an UPDATE to change `val1` to a different value:

```
UPDATE tbl_crdt SET val1 = val1 + 20 WHERE id = 2;
```

- On node1, COMMIT:

```
COMMIT;
```

- On all nodes, you will see this:

```
SELECT id, val1, val2 FROM tbl_crdt;
 id | val1 | val2
----+------+------
  2 |   30 |    0
(1 row)
```

References:

- Conflicts - https://www.enterprisedb.com/docs/bdr/latest/conflicts/
- List of Conflict Types - https://www.enterprisedb.com/docs/bdr/latest/conflicts/#list-of-conflict-types
- List of Conflict Resolvers (see the table- https://www.enterprisedb.com/docs/bdr/latest/conflicts/#list-of-conflict-resolvers
- Default Conflict Resolvers - https://www.enterprisedb.com/docs/bdr/latest/conflicts/#default-conflict-resolvers
- CRDTs - https://www.enterprisedb.com/docs/bdr/3.7/crdt/
- Eager Replication - https://www.enterprisedb.com/docs/bdr/3.7/eager/
- Conflict logging - https://www.enterprisedb.com/docs/bdr/latest/conflicts/#conflict-logging

2021 Copyright © EnterpriseDB Corporation All Rights Reserved