Postgres-BDR by 2ndQuadrant,

Third Generation Advanced Clustering & Replication

Whitepaper

28 Feb 2020

2ndQuadrant®
PostgreSQL

Overview

Bi-Directional Replication (BDR) from 2ndQuadrant, also known as Postgres-BDR, is a technology stack that provides advanced replication, very high availability, scalability, and multi-master replication options for PostgreSQL.

This white paper gives an in-depth overview of BDR3, the third generation of Postgres-BDR, including its features and their benefits, use cases, and deployment options.

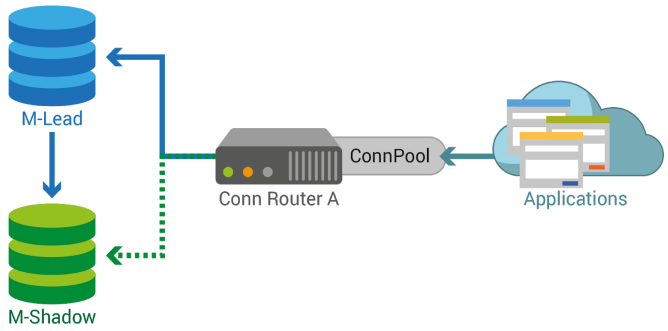
Table of Contents

Postgres-BDR v3 Architecture (“BDR3”, or simply “BDR”)

BDR3 runs as a plugin to core PostgreSQL 10, 11 and above, using the EXTENSION framework designed and implemented by 2ndQuadrant in core PostgreSQL.

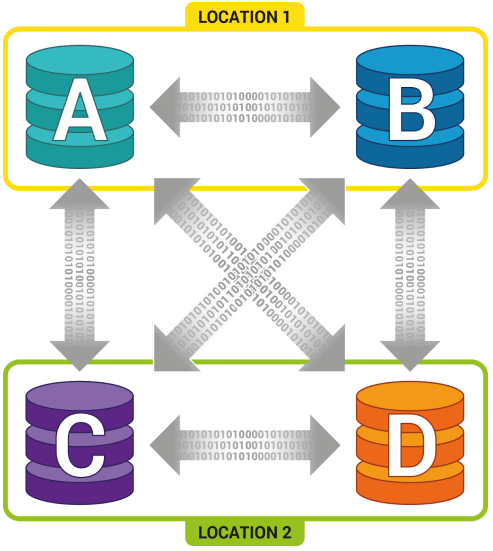
The purpose of BDR is to extend PostgreSQL (aka Postgres) with advanced replication technology that can be used to implement very high availability and multi-master replication. BDR3 builds upon earlier architectures to provide a very wide range of features and requirements. BDR has been in continuous development since 2012 and has been in production since 2014. Over this time, 2ndQuadrant has learnt many things about distributed systems - where they fail, what works, what doesn't work, and requirements for many future features.

The recommended configuration for BDR3 is to have multiple Logical Master nodes, each one protected by one or more Standby/Replica nodes. Each Logical Master forms a local cluster, while together these local clusters form one large global cluster.



High Availability can be further enhanced by having a Lead Master protected by a Shadow Master. This uses a minimum of 4 nodes - 2 logical nodes and their physical standby nodes (not shown). This configuration can provide Very High Availability, as discussed in the next section.

Various other architectures are supported, providing a range of possibilities for combining high availability with geographically distributed master nodes.



BDR supports "Commit At Most Once" functionality, ensuring that the most valuable transactions are neither dropped nor entered twice once committed by the application - a key feature for high value data.

BDR supports at least 48 logical nodes, however that is not a hard limit. A full BDR cluster including standby nodes could be 10s or 100s of nodes and would be capable of supporting Terabytes or Petabytes of data.

Many parts of BDR technology has been folded slowly into core PostgreSQL, so the majority of the BDR stack is already part of available PostgreSQL versions. Extensions, background workers, logical decoding, logical protocols, and logical replication are all features that have been developed and submitted successfully by 2ndQuadrant into core PostgreSQL. Further features will be fed back into core PostgreSQL when accepted. We have further features pending for inclusion into later versions of PostgreSQL.

Robustness, High Availability, and Disaster Recovery

BDR3 runs on PostgreSQL, which provides the robustness for each single node. BDR3 addresses the main issues impacting availability in distributed systems, listed below:

- **Node loss.** Switchover/failover
- **Cluster Expansion/Contraction.** Elastically scalable managed service
- **Rolling Software Upgrades.** Cross-version replication
- **Rolling Schema Change/Migration.** Cross-schema replication
- **User Error.** Backup/recovery

For maximum availability, the recommended configuration for BDR is to have multiple Logical Master nodes, each one protected by either a logical standby or a physical standby node. Data passes in both directions between logical nodes using logical streaming replication. Logical replication and physical replication can run concurrently, so the cluster is protected by two kinds of replication, making it much more resilient.

In the event of a node loss, a switchover to the local Standby can be performed. Replication can be enhanced with synchronous replication to avoid any data loss.

In this architecture, a switchover can happen instantaneously - as soon as any delay or suspected failure of the Lead Master occurs, it can switch immediately to the Shadow Master. If it is a false alarm and the Lead Master stays up, any changes are synced and replication continues - no time is lost waiting to find out the status of the server, allowing the application to continue without downtime. If the Lead Master goes down, the cluster switches over to a standby node and then does a resync. As long as the application is BDR complaint, there are no drawbacks to the frequency of responses from false alarms.

As a result of all of these factors, BDR is rated as capable of true five nines availability, that is 99.999% availability, or less than 26s of downtime per month.

The BDR3 cluster can add or remove nodes whilst remaining fully online. Nodes can be defined as either SendAndReceive (Master) or ReceiveOnly (Standby) nodes, allowing their use for reporting or software testing.

The logical master nodes do not need matching configurations, CPU architectures, or even software versions. This allows the BDR nodes to support **Rolling Upgrades** across major PostgreSQL or BDR software versions without cluster downtime.

Schema changes via DDL are propagated transparently across nodes. Fine grained locks have been implemented so that locking is minimized to only the required objects and only with a minimal lock appropriate to the request. In addition to this, BDR supports the ability to replicate across dissimilar schemas, which allows to add or drop columns of replicated tables without interrupting replication. This ability allows the user to perform **Rolling Schema Change** to achieve full platform high availability.

Many database maintenance commands can be executed on each logical master independently, so they have little effect on bandwidth or replication latency. Some DDL

commands are even allowed while nodes are down to allow administrators to proceed with certain maintenance actions, even with missing nodes.

BDR3 supports backup and recovery of each node through normal mechanisms. Clusters of more than 3 nodes can be backed up from any one node and yet still restored using Point-In-Time Recovery. Multiple logical nodes can perform backups, ensuring that whole-cluster backups are taken in different regions to offer disaster recovery capability.

Performance, Consistency, and Correctness

Tradeoff between Performance and Consistency

Brewer's CAP Theorem[1] states that it is impossible for a distributed data cluster to simultaneously provide more than two of the following: Consistency, Availability, or Partition-resilience. So, which two does Postgres-BDR choose?

The PACELC Theorem[2] adds to CAP, saying that in the general case a fully working system must choose between L - Latency and C - Consistency. What does this mean? Consistency between distributed nodes requires at least 3 round trips between nodes, so in a widely distributed system waiting for a transaction COMMIT to be consistent across nodes will force database response time (latency) to be very slow. If eager Consistency is not a requirement, improved Latency can be achieved as it is only necessary to write to the local node, making writes very fast. These are the only two choices - the 3rd generation of Postgres-BDR (BDR3) offers both

- Eventual consistency means ***fast*** database writes, with some complications
- Immediate ("eager") consistency means ***slow*** database writes

Eager consistency mode will take all changes made at COMMIT and confirm that each node has made those changes before a reply is sent to the user. This requires more work than the simplified mechanism of synchronous replication, which is already available in PostgreSQL - the RAFT protocol is used to achieve consistency. RAFT requires **three** network round-trips for a safe and consistent result. Across geographically separated systems, this might add >1s to transaction times, much in excess of the limit of human perception (100ms) and thus can be characterized in human terms as "very slow". If more nodes are added then these three trips will wait for the slowest node/link. Additionally, eager doesn't work well if a node is unavailable. While this is required in many use cases, some scenarios place efficiency at a higher priority. Along with eager consistency, Postgres-BDR also provides the ability to achieve consistency in a more efficient way.

The Fast/Efficient path requires **zero** network round-trips for the writer, greatly improving application transaction latency. Yet it introduces the possibility of *potential* transaction conflicts, which must then be understood, controlled, and excluded by the developer. These are not random operational issues like disk corruption, they are deterministic issues that must be addressed by user application developers.

BDR3 provides both “eager” and “fast/efficient” modes (see Advanced Features), allowing the user to choose the best policy for their use case. The choice of which mode to use can be set as a default, or can be set for each transaction by setting a single parameter for write consistency.

A user can choose to write only to the local node, to all nodes, or some subset. We typically recommended to use 1 for data and ALL mode for metadata (DDL and reference info), remembering that each logical node is already highly available.

Practical Issues

Let’s be clear: In fast mode, eventual consistency can lead to conflicts and these can lead to **potential** anomalies. For many applications there will be no problems at all. For some, a few changes may be required, while for a small few there may be a need for major rework. In general, if an application runs well on multiple CPUs then it can work well with BDR.

Let’s start by looking at the most common case, which is also the easiest to understand:

INSERTs will cause a conflict unless a unique identifier is used for each row. In a distributed database, it is necessary to use ***globally unique identifiers*** to avoid conflicts; that is, they must be unique across the whole cluster. These identifiers might already be assigned externally, but if assigned by the database, then either naturally non-conflicting keys (such as UUIDs) must be used or some global service must be used to provide them. BDR3 provides a global sequence service that is transparent to the application using time-sharded identifiers, allowing values to be used in date range searches. BDR3 also allows globally allocated sequences (“galloc”) that rely on consensus to agree ranges of values for each node. Functions for time-based Key Sequenced UUIDs (KSUUIDs) also supplement normal UUID generation.

If two INSERTs do have uniqueness collisions then there is a conflict. Conflicts can also occur between two UPDATEs or between INSERTs/UPDATEs/DELETEs, in various combinations. Conflicts are logged and then a conflict handler will be invoked to resolve the issue. The conflict handler can be set for each table and can be changed transactionally.

Default handling is based on Last Update Wins logic, so clocks must be synchronized with reasonable accuracy. If this is not possible or desirable BDR also supports row version numbering as a deterministic approach to identifying conflicts, in which case there is no need for exact clock synchronization. Default conflict handling occurs for each row.

Each table can optionally be enabled for Column Level Conflict Detection (CLCD) and Resolution, minimizing conflicts from updates. BDR3 also ships with Conflict-Free Datatypes (CRDTs), which eliminate any update conflicts. CRDTs also allow delta changes to work correctly, so users on different nodes can increment a counter and still achieve the same globally correct outcome.

Foreign Keys can give problems in distributed systems with eventual consistency, so BDR provides special functions for the user to circumvent these issues and avoid anomalies.

Assessing Suitability

The 3rd generation of BDR is compatible with PostgreSQL, but not all PostgreSQL applications are suitable for use on distributed databases. Most applications are already, or can be easily modified to become, BDR compliant, and some packaged applications are already certified. Naturally high contention database designs, amongst others, need to be avoided in order to make an application compliant with Postgres-BDR.

2ndQuadrant provides a short BDR Compliance Assessment service that verifies if an application is BDR compliant. 2ndQuadrant has developed tools to ensure that this assessment is thorough and accurate based on many years of experience with successful production deployments of applications with Postgres-BDR.

Scalability

BDR3.7 will allow a distributed database to scale horizontally. Additional write nodes can be added without taking the cluster down, so the cluster is elastically expandable.

BDR allows Replication Sets, which allow groups of tables to be defined as:

- Replicated - every logical node has a full copy of the data, all rows are replicated
- Sharded - each logical node has a partial copy of the data, optionally copied to just a few other Logical Master nodes for redundancy

Replicated data can be used to scale read access from any node. Sharded data can be used to scale write access, so together with the ability to expand the cluster with zero downtime this makes BDR3 ***elastically scalable***.

In some distributed databases, data is automatically placed onto shards using hash algorithms. In Postgres-BDR, placement of the data is configurable; it can be placed near the user or a scheme such as hashing can be used to organize the data.

Queries can be run on any node for replicated data. For sharded data, local data can be queried normally. Sharded data on remote nodes can be accessed using Foreign Data queries. Data across multiple remote nodes can also be queried similarly to the way queries would execute in a large data warehouse - but without the need to transport the data back and load it before queries can execute.

Deployment Options

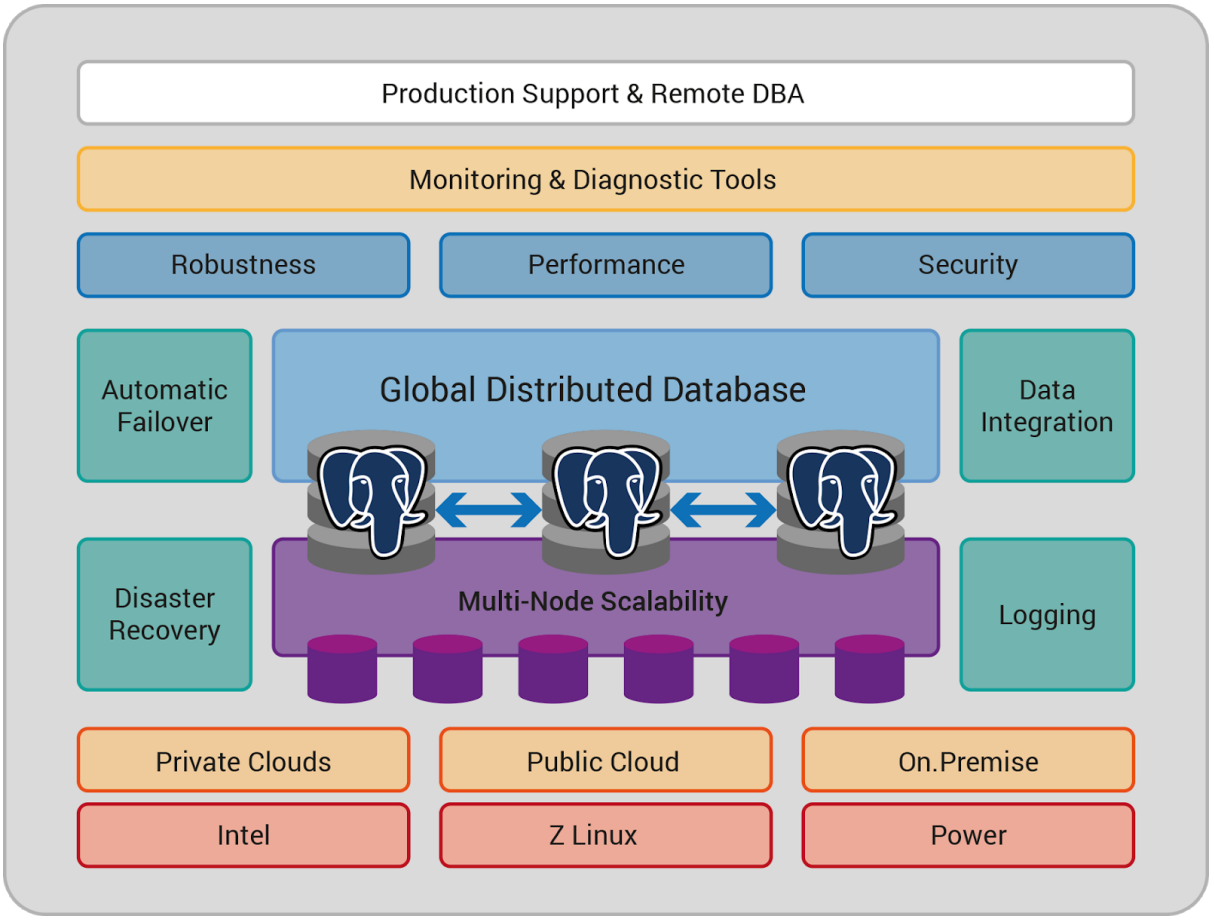
2ndQuadrant deploys clusters using Trusted PostgreSQL Architecture (TPA).

BDR has the following deployment models:

- On-Premise
  - Ansible
  - Docker
- Cloud
  - Kubernetes
  - AWS EC2

Trusted PostgreSQL Architecture (TPA)

TPA brings together a large number of standard components of the PostgreSQL ecosystem using best practices for implementation. It allows the deployment of highly available PostgreSQL clusters that can reliably recover from disaster. TPA can be used to deploy such clusters on a variety of platforms, in a virtual environment or bare metal, and in the cloud or on-premise. The figure below displays an example configuration.



Main Use Cases

Very High Availability

BDR can provide very high availability for the majority of applications and is a very popular use case for bi-directional replication.

Globally Distributed Database

Allow multiple departments in separate locations to access the same data with both read and write access, ensuring they operate as part of a global team on the same data. Direct user access to databases is possible on a global scale. For example, if a user wishes to check their remaining balance or current preferences, it is possible from any geographical location, most importantly from their closest database server, improving latency.

Orders taken in one region can be seen soon afterwards by other teams in other locations, allowing accurate reporting and control.

Retail sales data can be stored for local processing before being sent upwards. Product reference data can be sent down from central locations, with real-time price updates being applied across stores worldwide.

Data Privacy

Operate a global database where customer data is stored only in specific regions, with individual customers being able to restrict or allow their data to be accessible more widely.

With a server placed in each privacy jurisdiction it will be possible to operate a single application worldwide, yet at the same time respect each set of data privacy laws.

Internet of Things (IoT)

IoT applications collect measurement data from many small devices spread out globally. Sending all this data straight back to one central hub is neither practical nor useful; it is impractical because of the bandwidth and scalability requirements of the central server, and it is not useful because a single central server will be distant from many users.

Postgres-BDR allows a network of databases to be deployed in just the same way as a mobile telephone network is deployed - close to the user. Data can be uploaded quickly and easily, user access is close and responsive.

In this style of usage, the main hyper-detailed data collected is stored in only a few local masters. This is the only practical approach when expecting Petabytes of detailed data such as 5 second location sample. From there it can be semantically reduced, for example avoiding storage of repeated measurements when not moving. It can then be compressed, summarized and sent back up to more central servers.

Reference data can be managed centrally and replicated out to each node, allowing routing data, pricing tables, or map data to be sent down to the local nodes.

Development History

Development of Postgres-BDR started in late 2011, with a prototype presented at the PostgreSQL Developer Conference (PgCon) in May 2012. From there, 2ndQuadrant went on to develop BDR as full strength production software used worldwide.

BDR1 has been in production since 1 Sept 2014. BDR1 is a fork of PostgreSQL 9.4.
BDR2 has been in use since 2017. BDR2 runs as an extension on PostgreSQL 9.6.
BDR3 was newly released in 2018, utilizing improvements in PostgreSQL 10 and later.
BDR3 is the main code path for future developments.

Upgrade paths from BDR1 to BDR3 and from BDR2 to BDR3 are available, emphasising 2ndQuadrant's commitment to provide continuously available databases even across software releases.

Many parts of the BDR project are now part of core PostgreSQL, with further additions planned in the future. Many different developers, from within 2ndQuadrant as well as the wider PostgreSQL community, have been involved in the design, development, review, and testing of the various features of Postgres-BDR. Complex use cases from customers have driven the development of BDR far beyond the original feature set. This has resulted in a more robust technology than ever imagined. BDR3 is truly innovative and 2ndQuadrant is happy to deliver these features for wide adoption.

Advanced Features

The following advanced BDR features require an enhanced version of PostgreSQL available from 2ndQuadrant (2ndQPostgres), which contains production-quality features not yet accepted into core PostgreSQL. 2ndQPostgres 11 contains

- Commit-At-Most-Once (CAMO)
- Eager Consistency
- Multi-node Query Consistency
- Performance Improvements

References

1. Wikipedia contributors. (2018, March 19). CAP theorem. In *Wikipedia,* Retrieved 11:54, May 1, 2018, from https://en.wikipedia.org/w/index.php?title=CAP_theorem
2. Wikipedia contributors. (2018, January 16). PACELC theorem. In *Wikipedia,* Retrieved 11:55, May 1, 2018, https://en.wikipedia.org/w/index.php?title=PACELC_theorem