Unraveling Multi-Master Replication in PostgreSQL: Architectures and Solutions

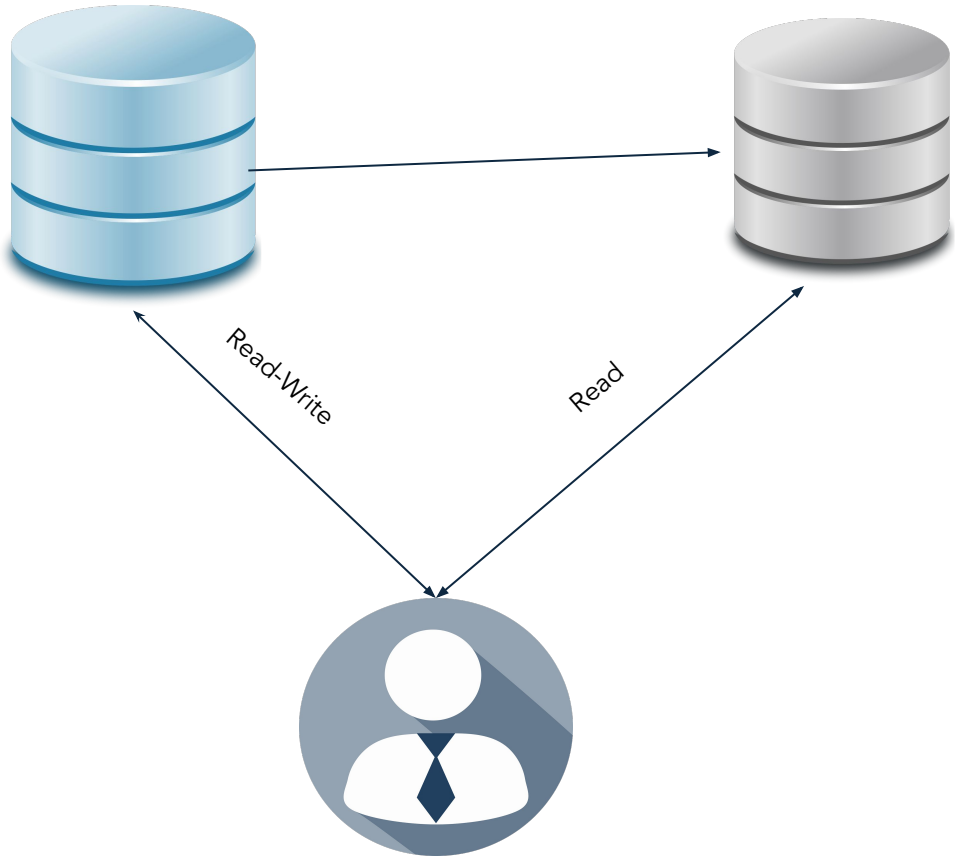
Principal Engineer

Ibrar Ahmed

Postgres
Conference 2024
San Jose / United States
April 18, 2024

pgEdge

Replication



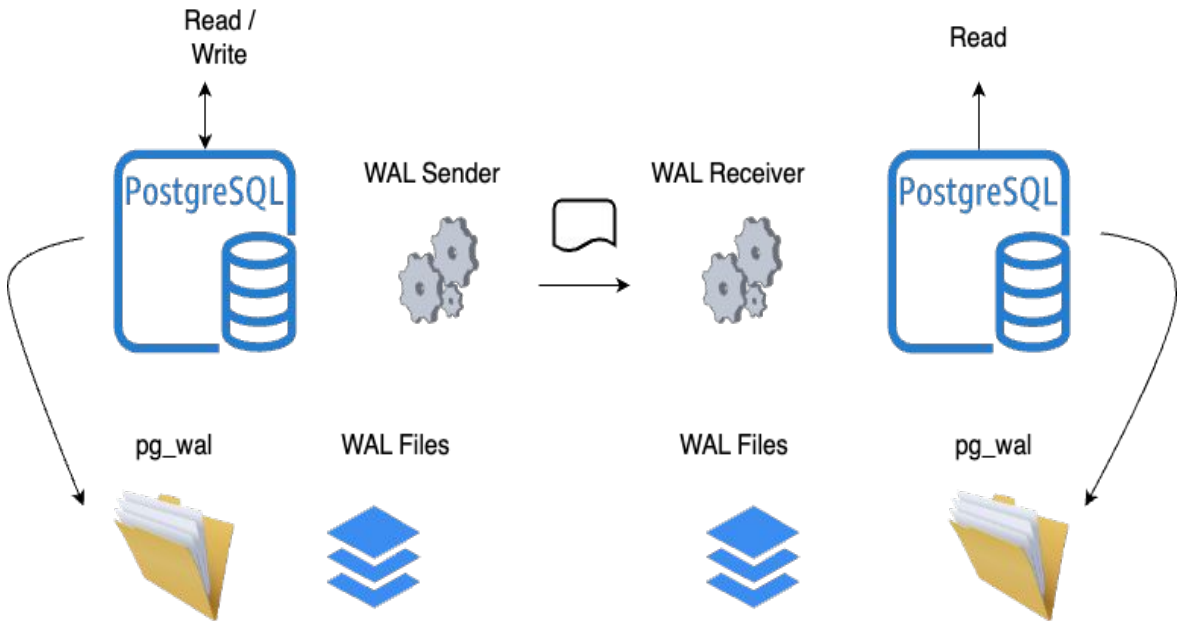
PostgreSQL supports several replication methods, including logical and streaming each catering to different requirements and use cases.

- **Streaming Replication:** A popular method for real-time replication, streaming replication involves a primary server sending data changes to one or more standby servers. This method is helpful for high availability and load balancing.
- **Synchronous vs. Asynchronous Replication:** In synchronous replication, transactions must be confirmed by both the primary and standby servers before being committed, ensuring data consistency but potentially affecting performance. Asynchronous replication, while faster, does not guarantee immediate consistency across servers.
- **Logical Replication:** Allows selective data replication at the table level, allowing the flexibility to replicate only specific tables or rows. It's beneficial for upgrading systems with minimal downtime and integrating data across different PostgreSQL versions.

Replication Use Cases

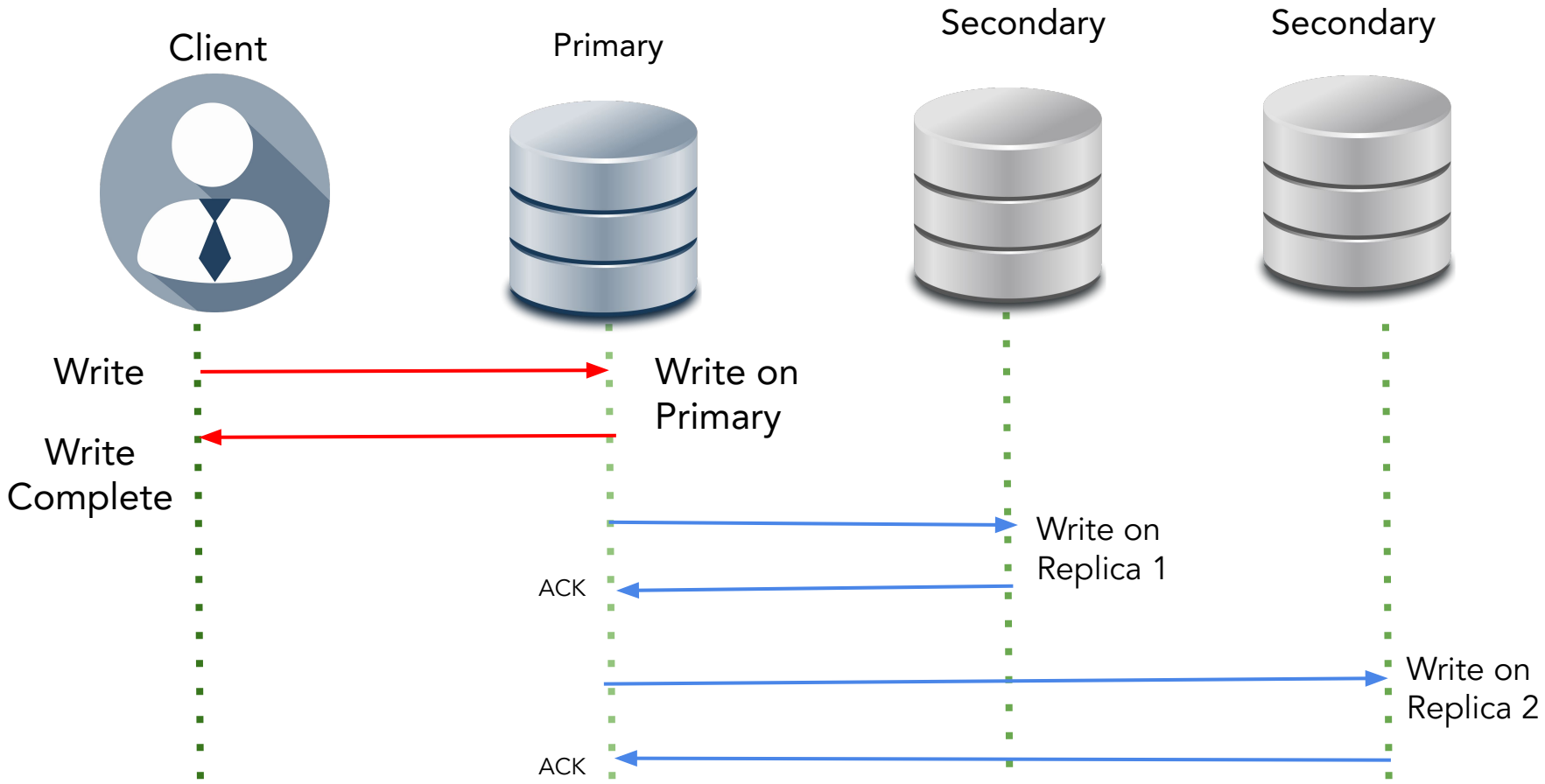
- **High Availability:** Ensuring continuous operation of the database by automatically failing over to a standby database if the primary database fails.
- **Data Latency**: Minimizing the delay in data being replicated across geographical locations to ensure timely access to data.
- **Data Residency**: Replicating data to servers in specific locations to comply with legal or policy requirements regarding where data is stored.
- **Near Zero Downtime for Major Upgrades Across Version**: Upgrading PostgreSQL versions with minimal service interruption by replicating data to a newer version instance and then switching operations to it.
- **Load Balancing / Query Routing**: Distributing read queries among several database replicas to optimize performance and resource utilization.
- **Data Migration**: Moving data from one database to another, potentially across different storage systems, locations, or database schemas.
- **ETL** (**Extract**, **Transform**, **Load**): Using replication to extract data from operational databases, transform it as needed, and load it into a data warehouse or analytical system.
- **Data Warehousing**: Aggregating data from various sources into a central repository to support business intelligence and reporting activities.

Physical Replication



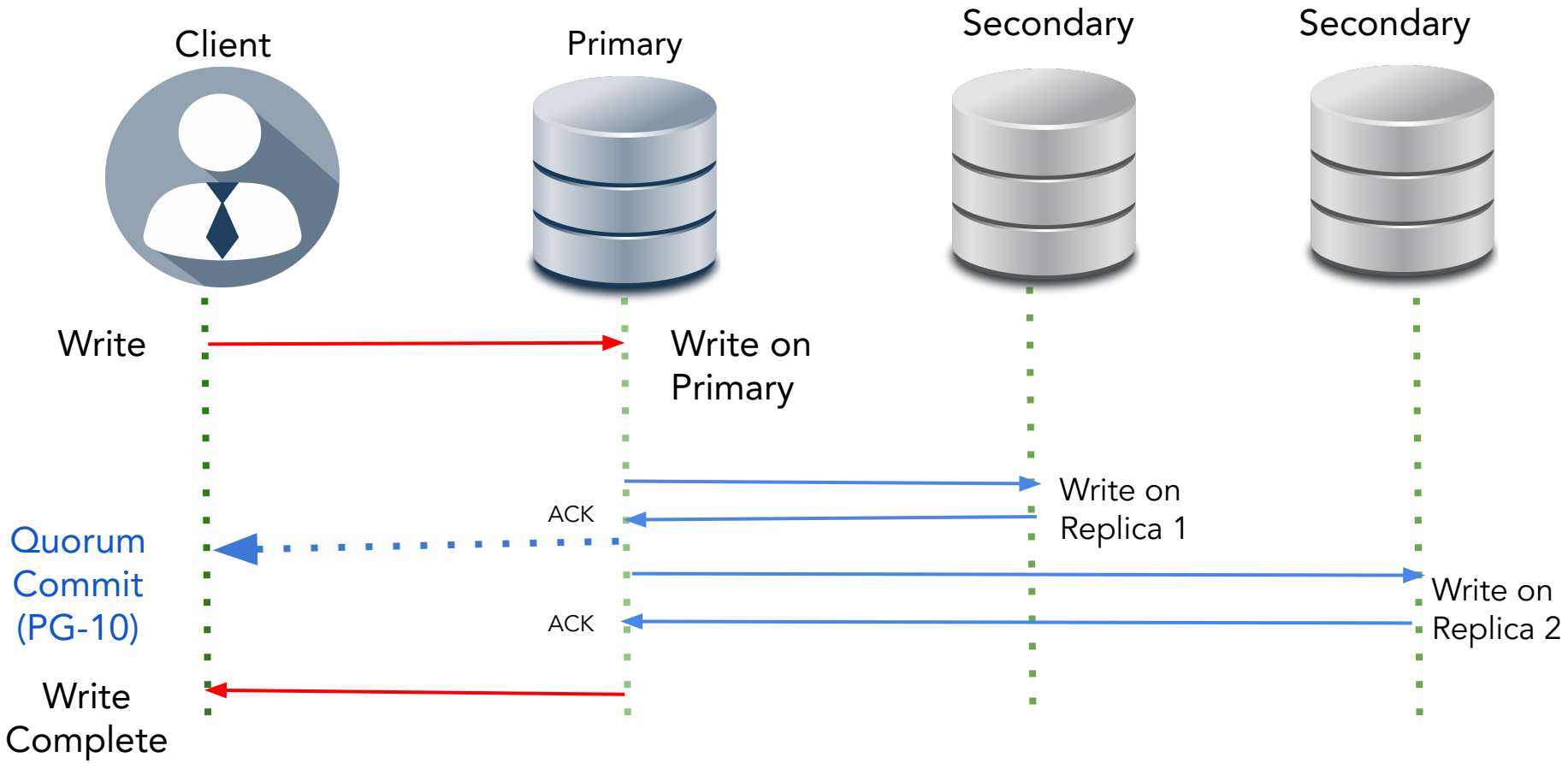
- Physical replication in PostgreSQL is the method of copying and synchronizing data from a primary server to standby servers in real-time.
- Real-time transfer of WAL records from primary to standby servers to ensure data consistency and up-to-date replicas.
- Standby servers can run in hot standby mode, allowing them to handle read-only queries while replicating changes.
- Configurable as either synchronous, for strict data integrity, or asynchronous, for improved write performance.
- Facilitates automatic failover by promoting a standby to primary in case of primary server failure.

Asynchronous Replication



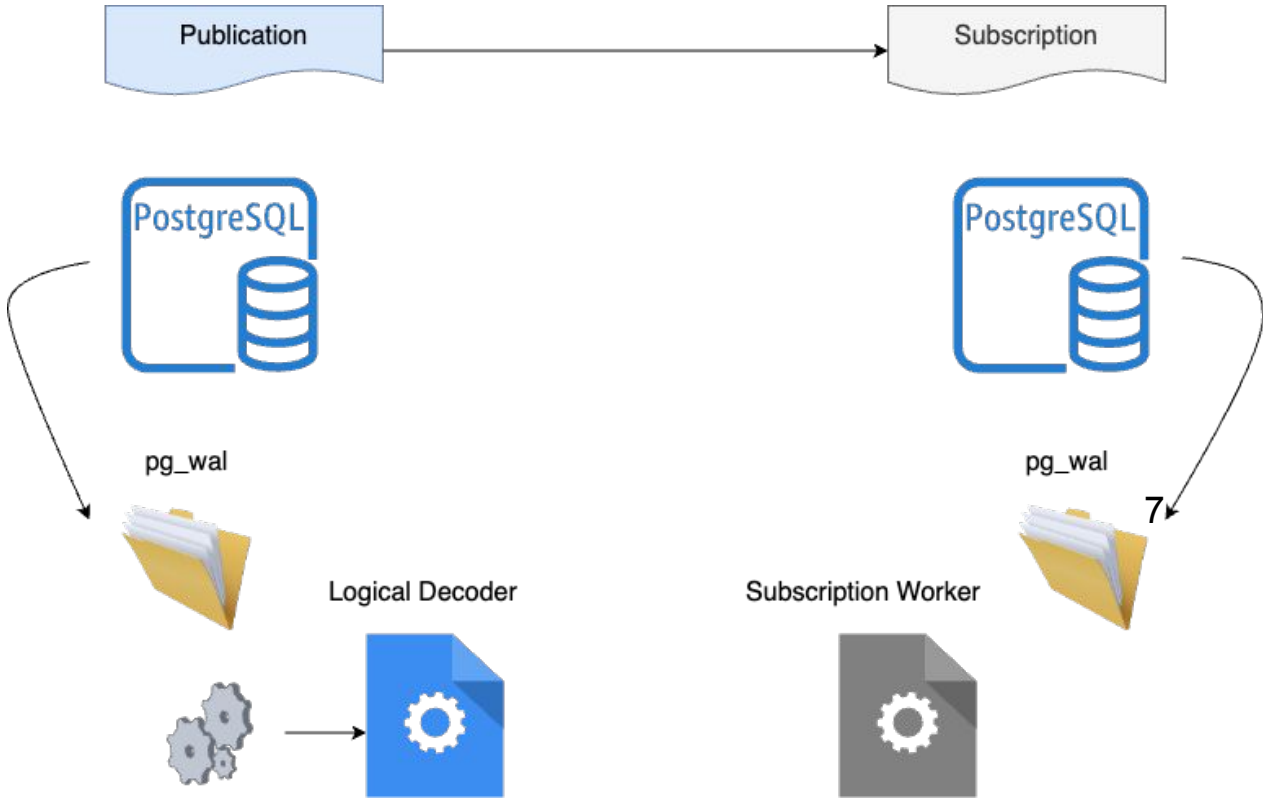
pgEdge

Synchronous Replication



pgEdge

Logical Replication



- Logical replication is the method of copying data objects and changes based on replication identity.
- Provides fine grained control over data replication and security.
- Publisher / Subscriber model - one or more subscriber nodes subscribe to one or more publisher nodes.
- Copy data in a format that can be interpreted by other systems using logical decoding plugins.
- Publication is set of changes generated from a table or group of tables.
- Subscription is the downstream end of logical replication.

PostgreSQL Logical Replication Slots

- PostgreSQL logical replication slots ensure efficient replication by retaining changes until confirmed by all subscribers.
- They play a crucial role in managing database changes in replication systems, allowing for seamless data streaming.
- Logical replication slots are essential for maintaining data integrity and consistency across multiple subscriber nodes.
- Monitor logical replication slots by querying 'SELECT * FROM pg_replication_slots;' to check slot status.

```
SELECT * FROM pg_stat_replication;
```

- Regular monitoring of logical replication slots is crucial for ensuring seamless replication and data consistency.

PostgreSQL Logical Replication Slots

- When deleting logical replication slots in PostgreSQL, with caution to prevent potential data loss

```
SELECT pg_drop_replication_slot(slot_name);
```

- Ensure no active subscriptions rely on the slot before deletion to avoid disrupting the replication process.
- Considerations when deleting slots include verifying all subscribers have received and applied the changes to maintain consistency.

PostgreSQL Logical Replication

First, ensure your PostgreSQL instance is configured to support logical replication. Key settings in the `postgresql.conf` file include:

- Set `wal_level` to `logical`.
- Set `max_replication_slots` to a number that accommodates your replication slots.
- Set `max_wal_senders` to a number that accommodates your replication connections.

```
wal_level = logical

max_replication_slots = 4

max_wal_senders = 4
```

Note: After modifying `postgresql.conf`, restart your PostgreSQL server for the changes to take effect.

Create a Publication

On the primary server (the source of the data), create a publication. A publication is a set of database changes that can be replicated to subscribers. You can publish all of the tables in a database or a selected subset.

Examples:

To publish all tables:

```
CREATE PUBLICATION my_publication FOR ALL TABLES;
```

To publish specific tables:

```
CREATE PUBLICATION my_publication FOR TABLE table1, table2;
```

Create a Subscription

On the subscriber database (the destination for the data), create a subscription to the publication.

This establishes the connection to the publisher and starts the replication process.

Example:

```
CREATE SUBSCRIPTION my_subscription

CONNECTION 'host=source_host port=5432 dbname=source_db user=replicator
password=secret'

PUBLICATION my_publication;
```

PostgreSQL Logical Replication in Retrospect

- **PostgreSQL 10**
  - Logical Replication was added
- **PostgreSQL 11**
  - Reduced memory usage
  - Truncate replication
- **PostgreSQL 12**
  - Allowing copying of replication slots.
- **PostgreSQL 13**
  - Replicate partitioned table
  - max_slot_wal_keep_size parameter (limiting WAL storage for rep slots)
- **PostgreSQL 14**
  - Replication slot activity reporting
  - Streaming in-progress replication
  - Support data transfer in binary mode
  - Allow decoding of prepared transactions
- **PostgreSQL 15**
  - Replication of prepared transactions
  - Allow publication of all tables in schema
  - Allow logical replication to run as owner of subscription

Allow logical decoding on the stand-by

- With this PG-16 feature, creation of a replication slot and logical decoding is possible from the stand-by server.
- Thanks to this feature, you can create subscriptions to a standby server.
- This has been in the works with the PostgreSQL community for some time now.
- Prior to this feature while creating logical replication slot at stand-by:

```
ERROR:  logical decoding cannot be used while in recovery
```

- Prior to this feature creating a subscription to a stand-by would produce :

```
ERROR:  could not create replication slot "mysub": ERROR:  logical
decoding cannot be used while in recovery
```

- This feature reduces the load on the primary server.

Monitoring and Managing Replication

PostgreSQL provides several functions and views to monitor and manage logical replication, such as:

- `pg_stat_replication` view provides information about the current replication connections.
- `pg_publication_tables` view shows which tables are part of a publication.
- `pg_subscription` view shows subscription information on the subscriber side.

To monitor replication status, you can query these views. For example, to see the status of subscriptions:

```
SELECT * FROM pg_subscription;
```

Active-Standby / Active-Active

Active - Standby

- One primary and one or more stand-by servers
- Write traffic on primary and read traffic load balancing on read replicas using external tools
- Synchronous / Asynchronous / Quorum commit choices
- Load Balancing / High Availability
- Automatic Failover using external tools
- Low chance of data loss

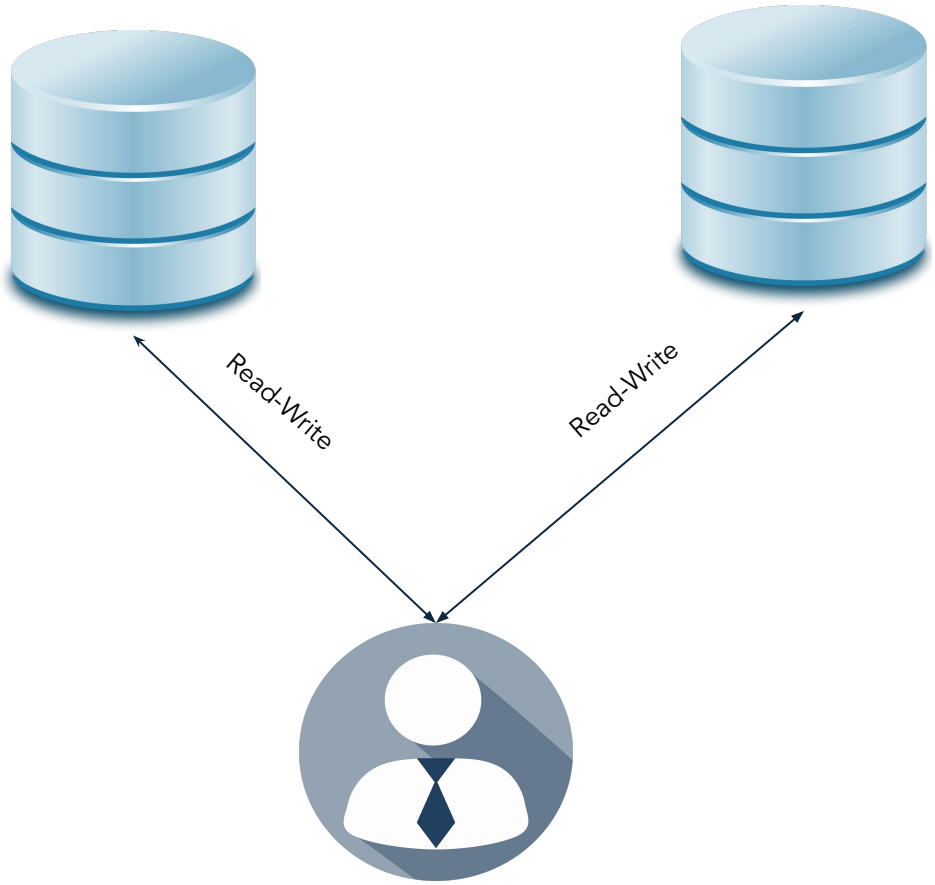
Active-Active

- One or more primary/active servers replicating between each other.
- Not part of core PostgreSQL, implemented using 3rd party tools and extension
- Requires conflict detection and resolutions
- Use cases are High Availability, data residency, data latency, and near zero downtime upgrades

Managing Conflicts

- Logical replication does not handle conflict resolution automatically. If there are conflicting changes made in the subscriber database, you'll need to manage these conflicts manually or through application logic.
- Limited Conflict Detection: PostgreSQL's built-in logical replication does not inherently detect conflicts at the row level when the same record is modified in different replicas at the same time. Conflict detection and resolution are typically handled at the application level or with external tools.
- Application-Level Resolution: Logical replication expects conflicts to be resolved by the application or the administrators. This means designing the application in a way that minimizes conflict possibilities (e.g., by segmenting data so that each replica writes to a distinct set of rows) or having a mechanism to detect and resolve conflicts post-replication.

Multimaster Replication



- **Simultaneous Data Writing:** Multi-master replication allows multiple database instances to handle write operations simultaneously, enhancing the database's write availability and scalability.
- **Conflict Resolution**: Incorporates mechanisms to handle conflicts that arise when the same data is modified at different nodes, ensuring data consistency across all nodes.
- **Load Distribution**: Distributes both read and write loads across several nodes, effectively balancing the load and improving overall system performance.
- **Improved Fault Tolerance**: Increases the database system's fault tolerance by allowing the system to remain operational even if one of the master nodes fails, thereby reducing potential downtime.
- **Real-Time Data Synchronization**: Ensures real-time or near-real-time synchronization between nodes, keeping the databases up-to-date and consistent with each other.
- **Geographical Distribution**: Supports geographical distribution of database nodes, which can reduce latency for globally distributed applications by allowing users to interact with the nearest database node.

Challenges of MMR

Data conflicts can arise when multiple masters simultaneously update the same data, leading to inconsistencies and requiring conflict resolution.

- Ensuring consistency across multiple masters is challenging, as immediate synchronization may not always be feasible, potentially causing data discrepancies.
- Scalability issues may arise due to the increased complexity of managing multiple master nodes and the need for efficient communication and coordination.

Principles of MMR

- Multi-master replication in PostgreSQL is based on decentralized architecture with multiple nodes capable of accepting write operations independently.
- These principles influence distributed system designs by enabling parallel writes, conflict resolution mechanisms, and eventual consistency across nodes.
- High availability is achieved through load balancing, fault tolerance, and automated failover mechanisms, ensuring data reliability and system continuity.

PostgreSQL's MMR Architectural Overview

- Key components of PostgreSQL's MMR include logical decoding, replication slots, and transaction timestamps for conflict resolution.
- Logical decoding captures changes at a granular level, replication slots ensure data consistency, and timestamps aid in conflict detection.
- These components work together harmoniously to synchronize data across multiple master nodes, facilitating distributed database systems efficiently.

Existing Solutions in PostgreSQL Ecosystem

- pgEdge (Spock): A state-of-the-art solution offering seamless multi-master replication with high performance.
- BDR: Known for its robust replication capabilities, allowing for distributed databases but requiring a complex setup and configuration.
- Postgres-XL: Provides scalable horizontal database clustering, ideal for large-scale applications, yet may pose challenges in terms of maintenance and management.

pgEdge (Spock): Solution Overview

- pgEdge (Spock) offers conflict resolution capabilities, supporting bidirectional synchronization between multiple masters while maintaining data consistency.
- Its unique features include the ability to handle high transaction volumes, ensuring minimal latency and downtime for critical business applications.
- Compared to other solutions, pgEdge (Spock) excels in providing real-time data replication, scalability, and seamless integration with PostgreSQL's ecosystem.

BDR (Bi-Directional Replication)

- BDR allows simultaneous reads and writes on multiple database nodes, ensuring data consistency across all masters.
- With conflict resolution mechanisms, BDR resolves data conflicts arising from concurrent write operations on different nodes.
- Practical applications of BDR include geographically distributed databases, high availability scenarios, and workload scaling in PostgreSQL.

Postgres-XL: Scalability Solutions

- Postgres-XL ensures horizontal scalability by sharding tables across multiple nodes, distributing the workload efficiently for large-scale data processing.
- The solution's shared-nothing architecture minimizes contention, allowing for seamless expansion in distributed environments without sacrificing performance.
- Postgres-XL's ability to handle multiple active masters provides fault tolerance, high availability, and read and write scalability in complex infrastructures.

Conflict Resolution Strategies

- **Last Write Wins (LWW):** In this approach, the system resolves conflicts by accepting the data from the most recent write based on timestamp, effectively overriding earlier writes.
- **Version Vectors:** Utilize version vectors to keep track of the version history of each data item. This method allows the system to identify and resolve conflicts by comparing version histories and merging changes accordingly.
- **Operational Transformation (OT):** Commonly used in collaborative applications, this technique involves transforming operations in such a way that they can be applied in any order while still achieving a consistent state across all nodes.
- **Conflict-Free Replicated Data Types (CRDTs)**: Implement CRDTs which are data structures designed to handle data consistency in a decentralized manner, ensuring that all replicas can converge to the same state without needing to resolve conflicts.
- **Manual Override**: Provide mechanisms for manual conflict resolution, where conflicts that cannot be automatically resolved are flagged for administrative intervention, allowing database administrators to make the final decision.

Pros and Cons of Multi-Master Replication

- **High Availability**: If one master fails, another master can continue to handle updates and inserts, ensuring service continuity.
- **Geographical Redundancy**: Masters are located in different locations, significantly reducing the risk of simultaneous failures.
- **Scalable Writes**: Allows data updates on multiple servers, enhancing write scalability.
- **Traffic Distribution**: No need to route all traffic to a single master, allowing better load distribution and utilization.

- **Complexity**: Multi-master replication is inherently complex, making it challenging to manage.
- **Conflict Resolution**: Simultaneous writes on multiple nodes can lead to conflicts, which are often difficult to resolve.
- **Manual Intervention**: Conflicts may sometimes require manual intervention to resolve, adding to maintenance overhead.
- **Data Inconsistency**: There is a risk of data inconsistencies across different nodes due to the replication mechanism.

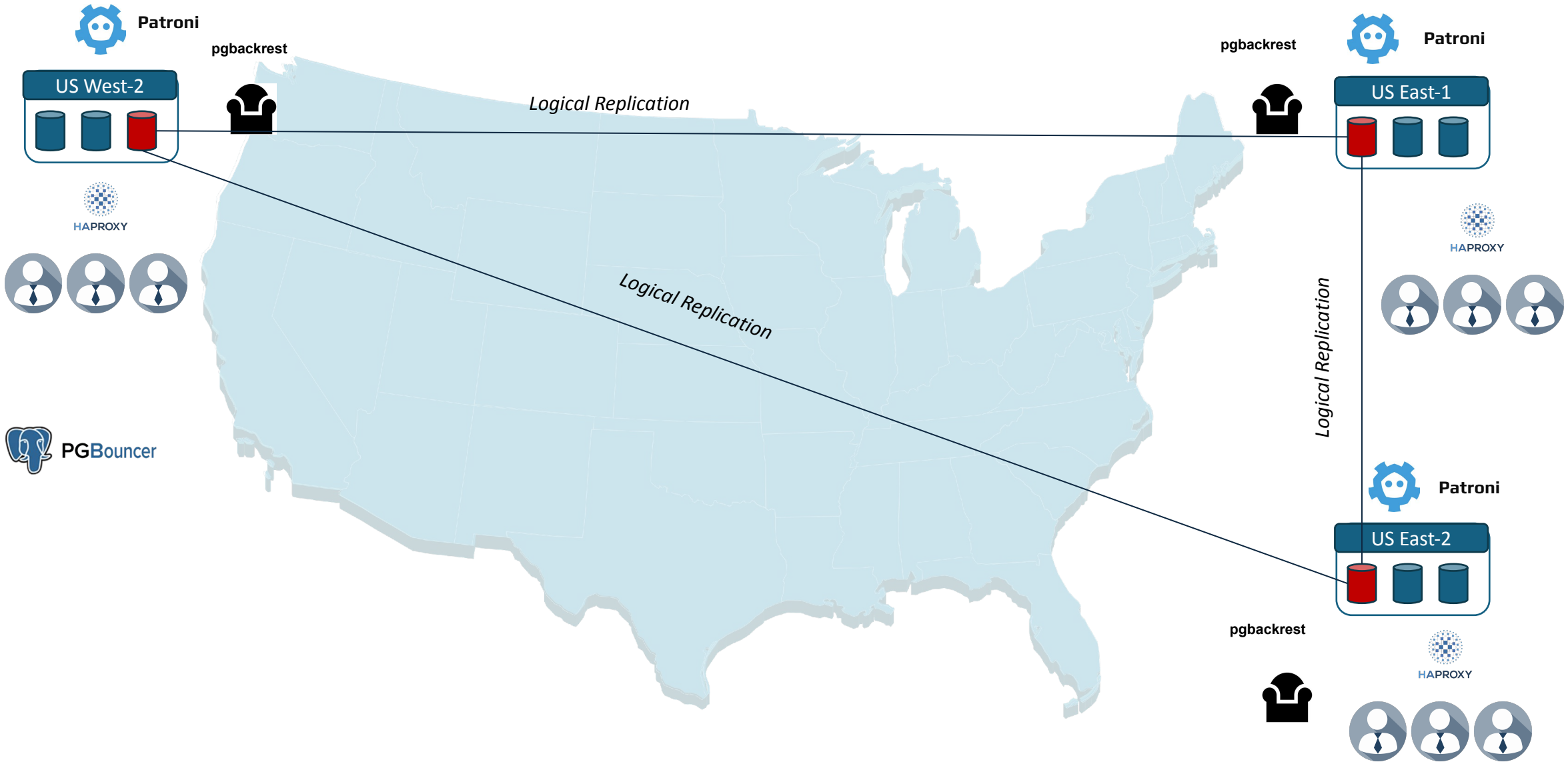
Performance Optimization in MMR

- Employ conflict resolution mechanisms to handle conflicting changes between multiple master nodes efficiently.
- Implement strict data consistency checks to ensure synchronization among all nodes in the replication setup.
- Utilize efficient data distribution strategies to balance the workload and prevent bottlenecks in multi-master replication systems.

Ensuring Data Integrity

- Ensuring data integrity in multi-master replication systems is crucial for maintaining consistency across nodes and preventing data conflicts.
- Best practices include implementing strong data validation mechanisms, performing regular audits, and utilizing checksums to detect anomalies.
- Tools like pgEdge's Spock, pglogical, Bucardo, and PostgreSQL's built-in features (like logical replication) are instrumental for validating data consistency in distributed nodes.

High Availability in PostgreSQL Using MMR



Questions

Code is like clay; in the hands of a skilled craftsman, it can be molded into something that stands the test of time. Remember, the art is not in writing code, but in crafting solutions that endure. Let's build not just for today, but for the future.

Ibrar Ahmed

pgEdge