Citus: Orange You Glad It's Postgres

Presented by Zak Tedder

Preview

- What we will cover:
    - Citus Overview
    - Installation
    - Configuration
    - Execution Across Nodes
    - Viewing Shard Information
    - Adding a Node
    - Removing a Node
    - Tenant Isolation
    - Replication & Backups

What is Citus?

- Citus is a PostgreSQL extension
- Citus distributes PostgreSQL across multiple nodes to reduce overhead and load on a per node basis
- Citus allows horizontal scaling way beyond single node limitations
- Citus increases performance and query speed
- Citus functions as if connecting to a single PostgreSQL database

When to Use Citus?

- Multi-Tenant SaaS
    - Have a dimensional fit that naturally groups data together using a tenant_id
- Real-Time Analytics
    - Delivers sub-second queries with 1000’s of concurrent users, while still being able to ingest real-time data
- Time-Series Data
    - Allows larger volumes of monitoring data to maintain sub-second response time

COMMAND PROMPT, INC.

Scalability

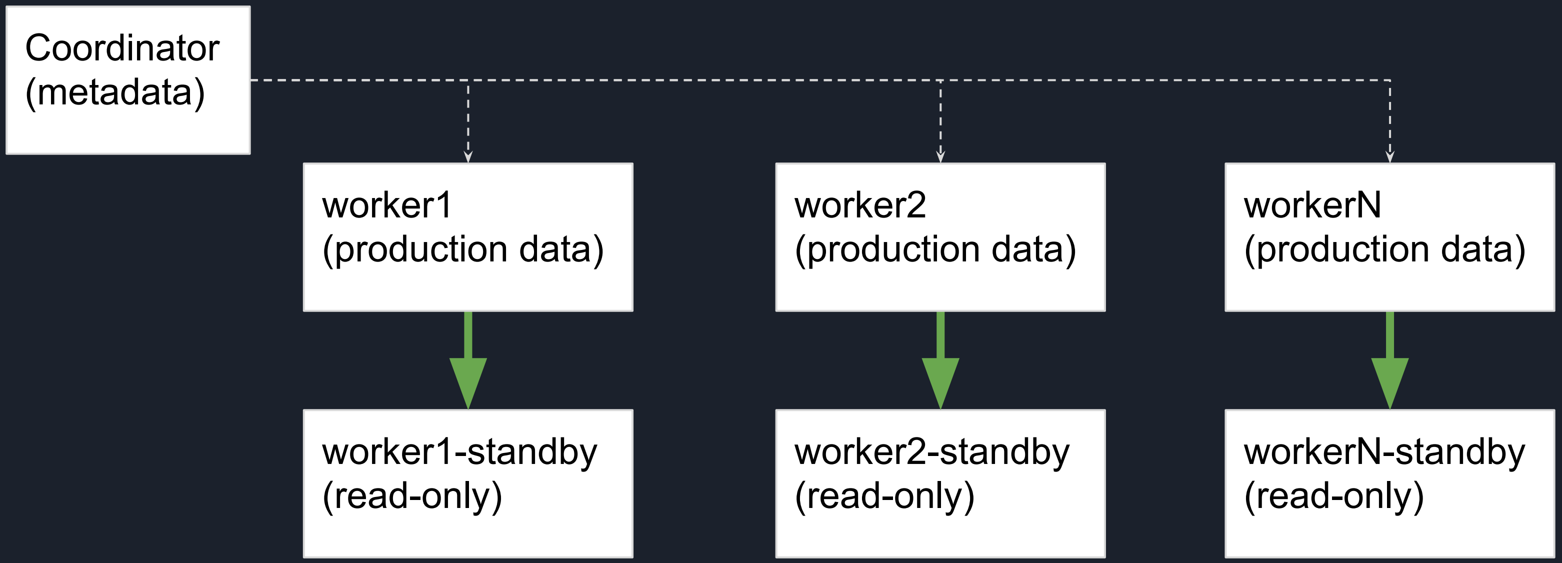
- One Citus client has 80 billion updates per day with a 20-node cluster and 80 TB of data, with plans for a 45-node cluster
- Another Citus client has 700+ billion events with a 100-node cluster and 1.4 PB of data
- Citus provides a turn-key operation with advanced capabilities for table and query distribution
- Citus uses logical replication to move data around with no downtime

Structure

- Citus isolates metadata to the coordinator node, which relays queries to the worker nodes and performs final aggregations on data
- Production data is distributed to the worker nodes, which perform computation on data
- Citus uses a distribution key/column to determine where to place blocks of data, called shards
- Distribution keys are used to group related data together on the same physical worker nodes
- The distribution keys help prevent cross-node traffic by focusing queries to one node (ideally)

COMMAND PROMPT, INC.

Structure Visualized



Installation

- Multi-node installation requires steps to be performed on the coordinator node and each worker node
- Add the repository on all nodes:
  - `curl https://install.citusdata.com/community/deb.sh | sudo bash`
- Install PostgreSQL 15 and Citus with one command:
  - `sudo apt-get -y install postgresql-15-citus-11.2`
- You will now have a running PostgreSQL instance on each node this is performed on. The next step is configuring the nodes

Multi-Node Configuration

- Changes to postgresql.conf:
  - `listen_addresses = '*'`
  - `wal_level = logical`
  - `shared_preload_libraries = 'citus'`
- Addition to pg_hba.conf. Allow access from worker nodes:
  - `host    all      all         10.0.0.0/8      scram-sha-256`

Multi-Node Configuration (cont.)

- Ensure a password is set for the postgres database user:
  - `psql -c "\password"`
- Create a database for the Citus cluster:
  - `psql -c "CREATE DATABASE demo"`
- Load Citus into the cluster:
  - `psql -d demo -c "CREATE EXTENSION citus"`

Multi-Node Configuration (cont.)

- As the postgres user, create and restrict permission to .pgpass file:
  - `touch .pgpass`
  - `chmod 600 .pgpass`
- Configure .pgpass
  - `hostname:port:database:username:password`
  - `*:*:*:postgres:<password>`

Coordinator Node Configuration

- From the coordinator node, assign the node to be the coordinator:
  - `SELECT citus_set_coordinator_host('192.168.0.1', 5432);`
- From the coordinator node, assign other nodes to be workers:
  - `SELECT * FROM citus_add_node('192.168.0.2', 5432);`
  - `SELECT * FROM citus_add_node('worker2', 5432);`
- Confirm assignments:
  - `SELECT * FROM citus_get_active_worker_nodes();`

COMMAND PROMPT, INC.

Choosing a Distribution Column

- Perhaps the most important aspect of planning a Citus cluster
- Grouping related data together on the same physical node makes queries fast, and adds support for all SQL features
- Co-location is the practice of tactfully dividing data to keep related data on the same machine (node in this case)
- Multi-Tenant apps use the term *tenant_id*
    - Examples: Company, account, organization, and customer
- Real-Time apps use the term *entity_id*
    - Examples: Users, hosts, or devices

Distribution Column: Best Practices

- Queries should ideally retrieve data from one node at a time
- We want small meaningful blocks of data, with columns frequently used in GROUP BY or JOIN queries
- We want even-ish distribution of data among worker nodes at all times
- If multiple tenants share a small table, you can distribute the table to each node as a *reference table*

Distribution Column: Real-Time & Time-Series

- Do not use *timestamp* as distribution column
- Potentially leads to skewed data and skewed workload
- Citus functions alongside PostgreSQL standard partitioning to group related data together by time ranges

Understanding Your Schema

```
CREATE TABLE sales (

     transaction_id SERIAL NOT NULL,

     total_cost MONEY NOT NULL,

     date_time TIMESTAMP NOT NULL,

     store_id INT NOT NULL,

     PRIMARY KEY(transaction_id)

);
```

Example: Simple table to track sales details across a number of different stores

Understanding Your Schema

```
CREATE TABLE sales (

    transaction_id SERIAL NOT NULL,

    total_cost MONEY NOT NULL,

    date_time TIMESTAMP NOT NULL,

    store_id INT NOT NULL,

    PRIMARY KEY(transaction_id)

);
```

Which column will group the data best?

- transaction_id - Would not group enough data together very well. It is a unique primary key that separates each sale, but it will not group transactions at the same store or in any other meaningful way.

Understanding Your Schema

```
CREATE TABLE sales (

    transaction_id SERIAL NOT NULL,

    total_cost MONEY NOT NULL,

    date_time TIMESTAMP NOT NULL,

    store_id INT NOT NULL,

    PRIMARY KEY(transaction_id)

);
```

Which column will group the data best?

- total_cost - Will not group data together well. There might be some occurrences of the same cost, but it will not likely group the data together meaningfully.

Understanding Your Schema

```
CREATE TABLE sales (

    transaction_id SERIAL NOT NULL,

    total_cost MONEY NOT NULL,

    date_time TIMESTAMP NOT NULL,

    store_id INT NOT NULL,

    PRIMARY KEY(transaction_id)

);
```

Which column will group the data best?

- date_time - Won’t likely work well. This is a timestamp type, so it would be possible to group the data using time partitioned data. However, at any point in the day or week, data will only be ingested by one node, leading to a skewed workload. Additionally, any queries across time will cross nodes.

Understanding Your Schema

```
CREATE TABLE sales (

    transaction_id SERIAL NOT NULL,

    total_cost MONEY NOT NULL,

    date_time TIMESTAMP NOT NULL,

    store_id INT NOT NULL,

    PRIMARY KEY(transaction_id, store_id)

);
```

Which column will group the data best?

- store_id - Will group data together the best. It will group data well across servers, as most queries will be isolated to data about individual stores. This will also be a consistent distribution column that can be added to other tables for colocation.

Executing Across Nodes

- Assign distributed table and distribution key:

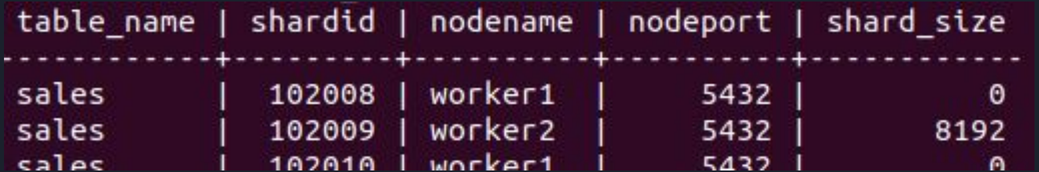
    -
      ```
      SELECT create_distributed_table('table_name',
      'distribution_key');
      ```
- Confirm data has distributed to each worker node:

    -
      ```
      SELECT * FROM table_name;
      ```
- Clean the coordinator of the original data, now distributed shards:

    -
      ```
      SELECT citus_drain_node('citus', 5432);
      ```

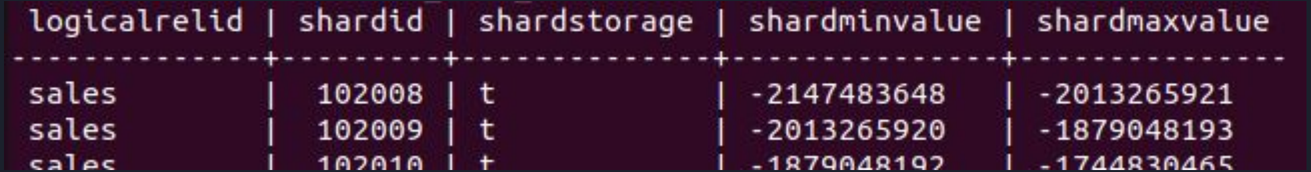
Understanding the Catalog

- `citus_shards` - View shard information:



```
table_name | shardid | nodename | nodeport | shard_size
-----------+---------+----------+----------+-----------
sales      |  102008 | worker1  |     5432 |          0
sales      |  102009 | worker2  |     5432 |       8192
sales      |  102010 | worker1  |     5432 |          0
```

- `pg_dist_shard` - How Citus chooses tenant distribution:



```
logicalrelid | shardid | shardstorage | shardminvalue | shardmaxvalue
-------------+---------+--------------+---------------+--------------
sales        |  102008 | t            | -2147483648   | -2013265921
sales        |  102009 | t            | -2013265920   | -1879048193
sales        |  102010 | t            | -1879048192   | -1744830465
```

COMMAND PROMPT, INC.

Adding a Node

- Assign another node to be a worker:
  - `SELECT * FROM citus_add_node('worker3', 5432);`
- We can review the rebalance plan before we execute:
  - `SELECT get_rebalance_table_shards_plan();`
- Rebalance the table shards to distribute some to the new worker:
  - `SELECT rebalance_table_shards();`

COMMAND PROMPT, INC.

Removing a Node

- To remove a node, we must first empty the node. The most likely case will require moving each shard off the node to another node. The easiest is to drain the node of data (or undistribute the table):
    - `SELECT citus_drain_node('worker2', 5432);`
    - `SELECT undistribute_table('sales');`
- Remove a worker node:
    - `SELECT * FROM citus_remove_node('worker2', 5432);`

Tenant Isolation

- We can isolate a tenant to a new shard with:
  - `SELECT isolate_tenant_to_new_shard('table_name', tenant_id);`
- It will return the new *shardid*. We use this to find the find the node holding the new shard:
  - `SELECT nodename, nodeport FROM citus_shards WHERE shardid = 102240;`
- Then we move the shard to our desired node:
  - `SELECT citus_move_shard_placement(102240, 'source_host', source_port, 'desination_host', destination_port);`

Replication and Standbys

- Each node must be backed up individually
- Logical replication is used to move the shards, streaming replication is a good choice to replicate nodes
- Binary replication can be initiated with pg_basebackup or pgBackRest
- Add a secondary node to the Citus metadata:

  -
    ```
    SELECT * FROM citus_add_secondary_node('new_node',
    5432, 'primary_node', 5432);
    ```

Summary

- We have covered:
  - Citus Installation
  - Configuration
  - Execution Across Nodes
  - Viewing Shard Information
  - Adding a Node
  - Removing a Node
  - Tenant Isolation
  - Replication & Backups

Thank you for you time! Questions?

COMMAND PROMPT, INC.