Database Systems:

Module 13, Lecture 4 – Cassandra Architecture

Instructor: Alan Paradise

Cassandra

LESSON OBJECTIVES

- Become familiar with the Cassandra NoSQL database
- Describe, at a high level, Cassandra's "wide row column family" architecture
- Explain how the Cassandra software embodies many key components of the NoSQL database model

Cassandra

Cassandra is a NoSQL database

- Stores data in a key:value pair format within a wide-row, column-family structure

What Cassandra is NOT

- Cassandra is NOT Relational
- Cassandra does NOT store data in tables
- Cassandra does not use the SQL query language, but uses a very similar "CQL", (Cassandra query language)

Cassandra

What Cassandra is:

- Massively scalable, free, open source
  - Like Mongo, the community edition is free
  - Enterprise C* users can purchase support and add-on features through a vendor like DataStax
- Designed for High Performance and High Scalability
- Designed for High Availability, fault tolerant with no SPOF
- Abbreviated as C*

Cassandra

Cassandra's Heritage. Based on:

- Google Big Table which is the Core foundation for many Google services and is the foundation for Cassandra's internal storage model
- Amazon Dynamo: Which supports many of Amazon's core services and is the foundation for Cassandra's distributed backbone
- Facebook -- which developed and open-sourced Cassandra

Cassandra

Cassandra

Cassandra provides the benefits of these technologies

- But, It improves them for Cassandra's needs
- C* Data model is a partitioned row store (with roots in Google's "BigTable")
- Peer-to-peer distributed architecture (roots in Dynamo)

Cassandra

Cassandra Concepts

Although not really stored in a "table", data in Cassandra is

- Row-oriented: Each row is an aggregate with column families representing meaningful, related chunks of data within that aggregate.

- Column-oriented: Each column family defines a record with sets of related data. You then think of a row as a collection of related records in all column families.

Cassandra

Cassandra Terminology

- **Partition**: defines the mandatory part of the primary key all rows in Cassandra must have. All queries should supply the partition key in the query.
- **Row**: contains a collection of columns identified by a unique primary key made up of the partition key and optionally additional column clustering keys.
- **Column**: A single piece of data which belongs to a row.

Cassandra

Cassandra Concepts

- Cassandra uses "CQL" – Cassandra Query Language
- Allows users a familiar row-column-based, SQL-like language
- BUT – no joins

Cassandra

C* is a peer-to-peer, fully distributed system where

- All nodes are equal (no masters)
- Data is partitioned (replicated & sharded) among multiple nodes in a cluster
- Sharding and replication are configurable
- No node is a single point of failure (SPOF)
- Any node may be read from or written to

Cassandra

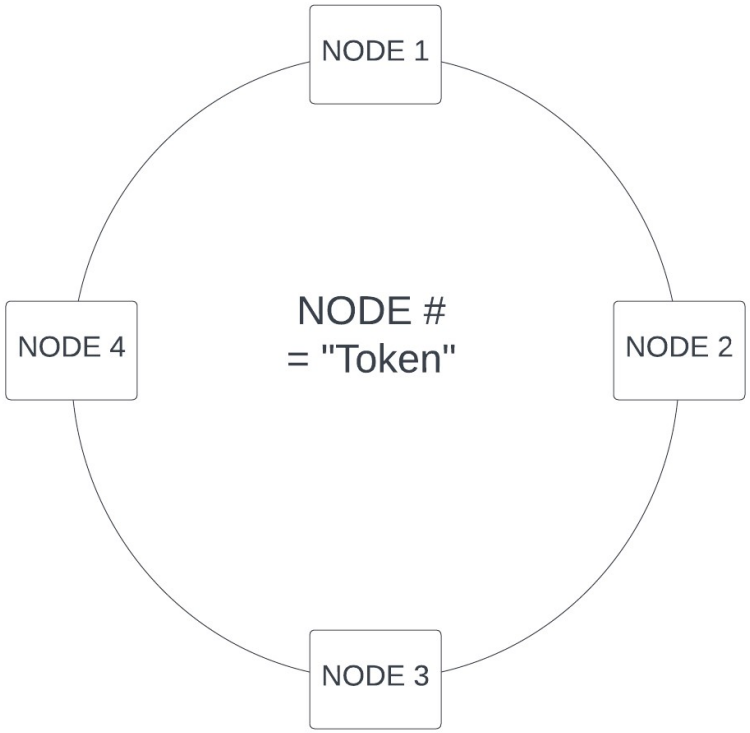
C* "Instance" = A collection of independent nodes running C*

- Configured into a cluster
- All nodes are peers (i.e. they all serve the same role)
- Data is distributed across all nodes in a cluster
- All nodes perform the same function
- Nodes store data and service client compute requests
- A client may read/write from/to any node, which becomes the coordinator for servicing that particular request
- Nodes can have different capacity/resources available (e.g. memory, CPU, disk)
- C* distributes load based on the available resources

Cassandra

C* "Cluster" = A "ring" of C* nodes

- Each node is identified by a token
- Data is partitioned across the nodes by token
- Each node has responsibility for a subset of data



Cassandra

- Each C* node is assigned a token (a number), whose value determines
  - The logical position of the node in the ring
  - The range of data the node is assigned

- A partition key (also called row key) uniquely identifies data stored in Cassandra
  - The partition key is extracted from the data being stored
  - It is part of the data's primary key
  - The partition key is hashed, and the hash is used to partition the data across the cluster based on the range of data for each node

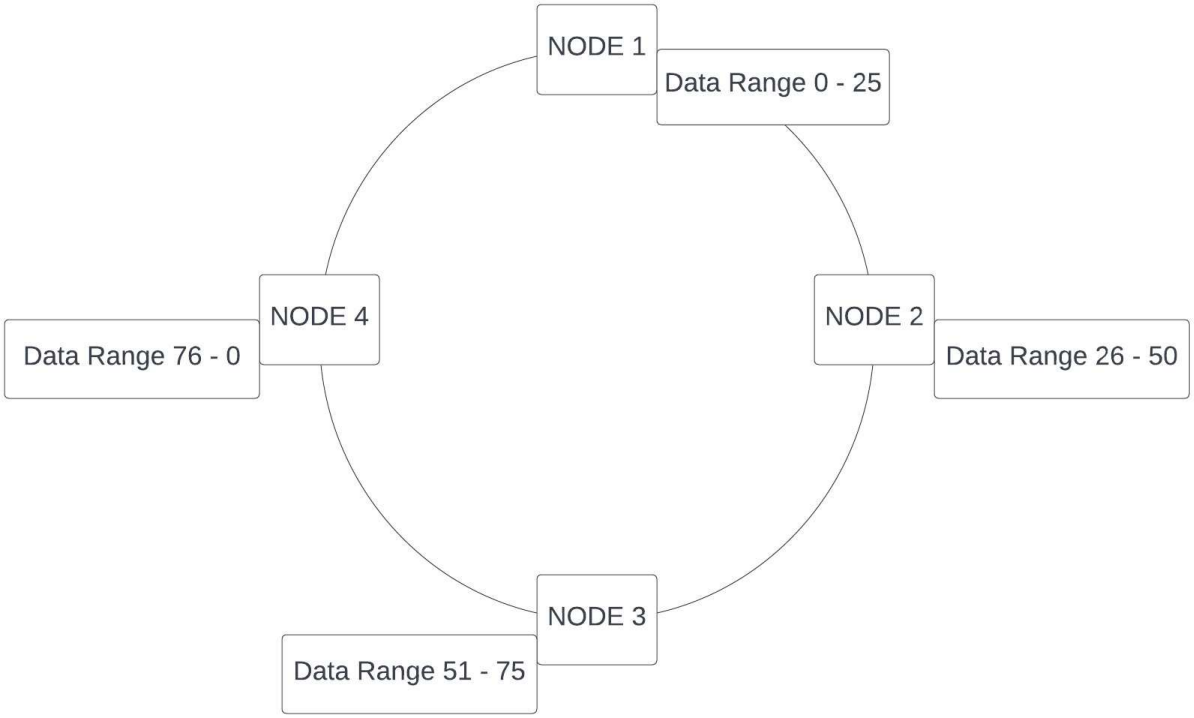
Cassandra

- Partitioning distributes the data randomly around the ring
- So each node shares the load equally – which is critical for scaling

Cassandra

- This [overly simplistic] cluster has 4 nodes with tokens of 0, 26, 51, 76
  - Each node holds data with partition keys that fall between its token and the next token
    - e.g. The node with token=26 is responsible for data with a partition key hash of 12
- The last node is the predecessor of the first (forming the ring)

Cassandra


NODE 1
Data Range 0 - 25
NODE 2
Data Range 26 - 50
NODE 3
Data Range 51 - 75
NODE 4
Data Range 76 - 0

Cassandra

Let's look at an example

Consider a Stock Trade table that tracks stock trades

- Its primary key is ((stock_symbol, trade_date), trade_id)
- The partition key is the first element of the primary key
- In this case a combination of stock_symbol and trade_date

So ALL trades for a stock on a given day reside on the same node

- The trade_id is a UUID - a universally unique identifier
- It is a surrogate key – similar to the relational concept of a sequentially assigned auto-incremented identifier

Cassandra

Let'sLet's look at an example

CQL:

```
CREATE TABLE Trade
(
  stock_symbol VARCHAR,
  trade_date TIMESTAMP,
  trade_id TIMEUUID,
  description VARCHAR,  // etc.
  PRIMARY KEY ((stock_symbol, trade_date), trade_id)
);
```

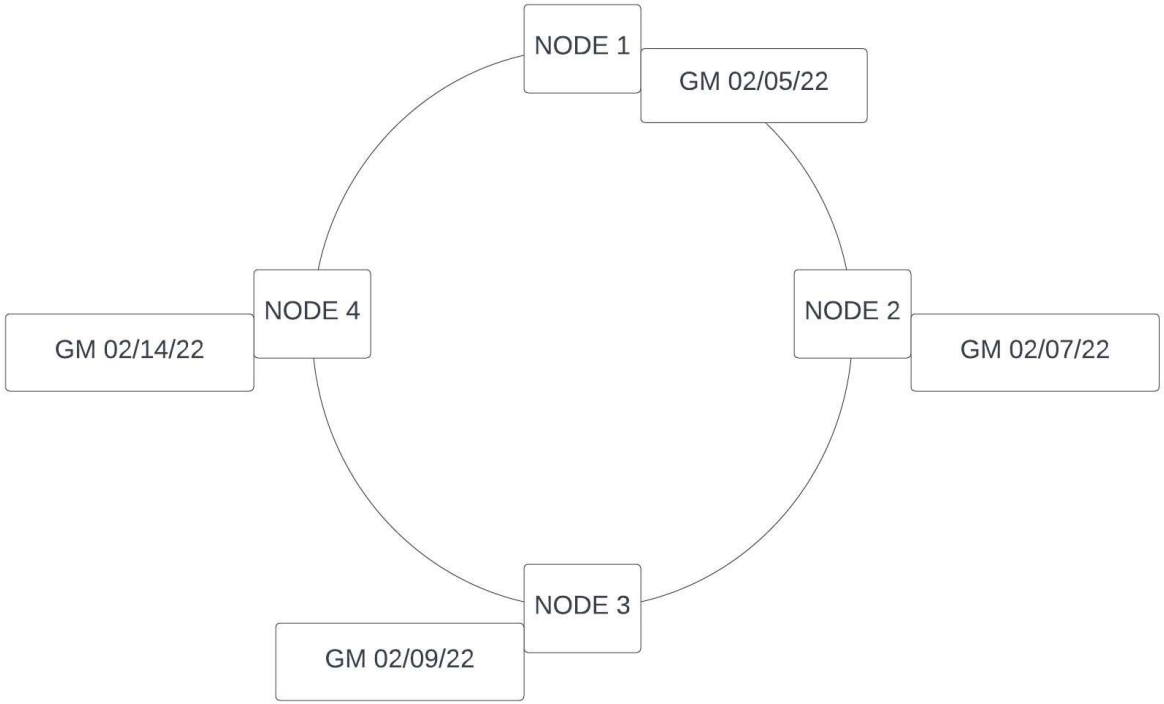
Cassandra

Let's look at an example

- Assume we have trades for GM, on dates in February 2022
- Assume that the partition mechanism results in the data distribution (next slide)
- Note how the trades for a given day are on one node
- There can be many trades for a given day - all on one node
- Different days are distributed across different nodes
- You can easily add more nodes to handle more data
- New trades will be distributed across the new nodes

'

Cassandra


NODE 1
GM 02/05/22
NODE 2
GM 02/07/22
NODE 3
GM 02/09/22
NODE 4
GM 02/14/22

Cassandra

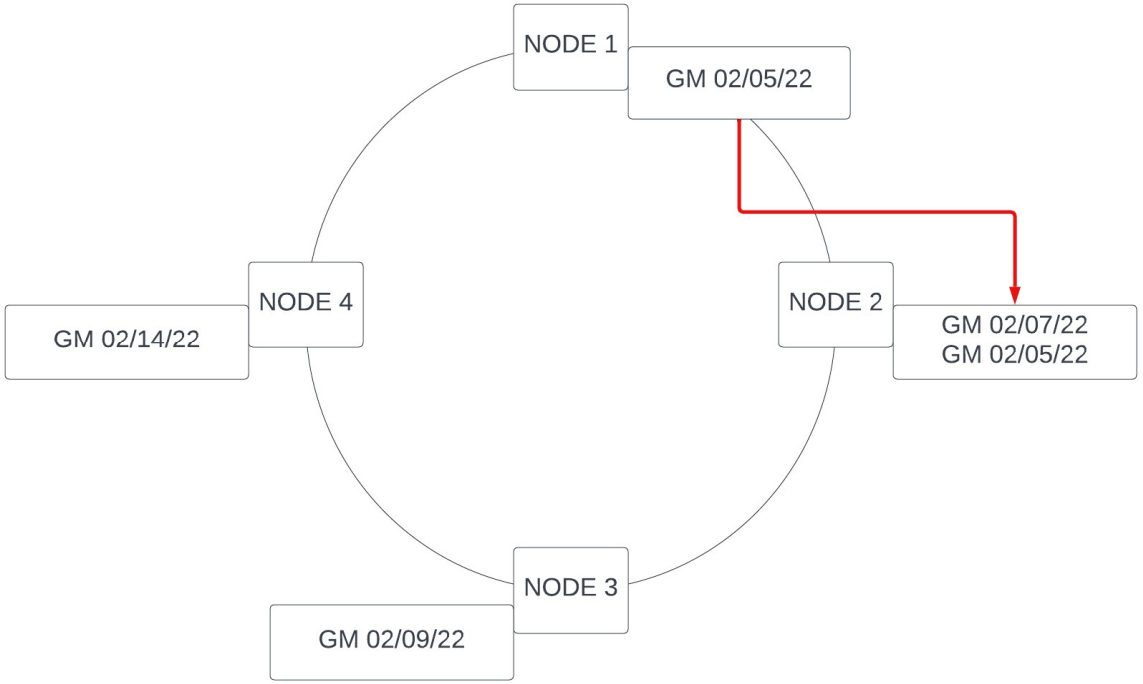
Configurable Replication

C* replicates data on multiple nodes for high availability and scalability

- All replicas are equal - no primary or master replica
- Replication defined at the data level via the replication factor "RF" (total number of replicas)
  - RF 1: One copy on a single node
  - RF 2: Two copies on two different nodes, etc.
- Provides very flexible replication

Example Below: This is a 4-node cluster with RF=2 showing data on two nodes

Cassandra



– In this simple example, the data is spread to nodes 1 and 2
– So node 2 holds its own data, plus the data replicated from 1

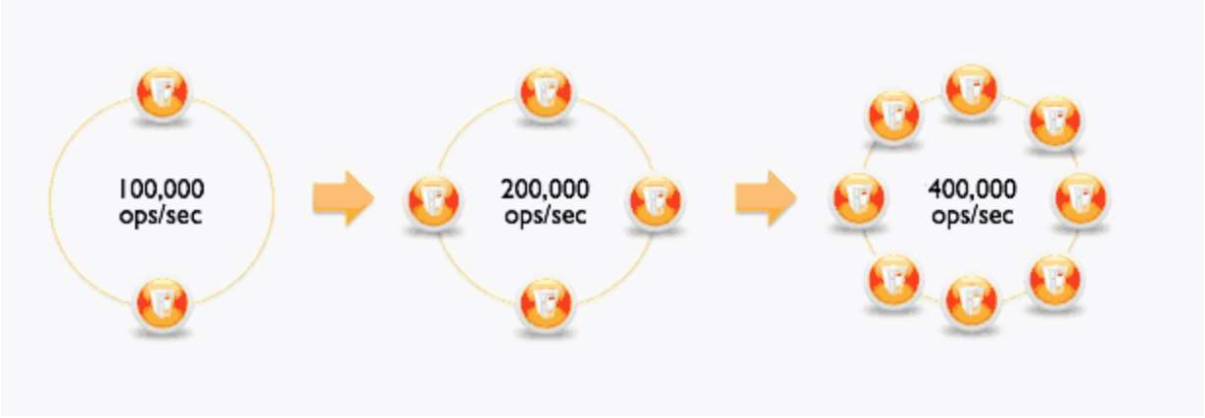
Cassandra

Linear Scalability as in Cassandra

Capacity may be easily scaled by simply by adding new nodes

For example:

- Suppose 2 nodes can handle 100,000 transactions per second,
- 4 nodes will support 200,000 transactions/sec, and
- 8 nodes will tackle 400,000 transactions/sec:1



Cassandra

Suppose C* has been implemented in a cluster spread across multiple data centers. If one data center becomes unreachable:

Cassandra, implements the concept of **Eventual Consistency**

- C* will allow writes to nodes in any datacenter it can reach
- When the network failure is fixed, and the partition is available, C* will bring all the nodes up to date
- However, until then, the data will be inconsistent

Cassandra

C* provides what is called **Tunable Consistency**

Data is replicated to multiple nodes

Consider what happens when an update occurs?
– The update may not propagate to all replicas immediately

What happens if a read of that row occurs before the update is propagated to all replicas?

– The read may retrieve older (pre-updated) data from a replica

Consistency control may be configured by the C* Admin
Tighter control eliminates the risk of a "stale read", but slows things down.

That is, **Tunable Consistency**

Cassandra

For more information:

Great Tutorial: TutorialsPoint

https://www.tutorialspoint.com/cassandra/index.htm

Great Overview Article: Dzone

https://dzone.com/articles/an-introduction-to-apache-cassandra

Cassandra

Next Topic: Cloud Databases