APACHE CASSANDRA

Author: Manikandan Ganesh

Designation: Graduate Student

School: Illinois Institute of Technology

1. USE CASES:

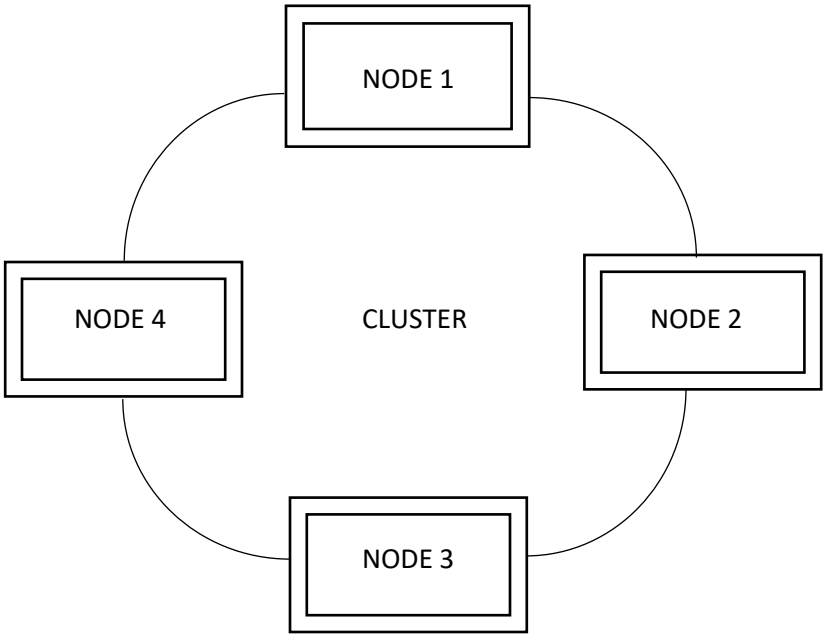
Nearly 1500 organizations use Cassandra for different purposes such as Collecting, Processing and Analyzing data from Sensors (CERN, Internet of Things, Smart Meter), Recommendation to the users from data collected from online websites (Netflix, IMDB, GrubHub), Fraud Detection from multiple resources (Barracuda Networks), Location Based Services (HERE Maps, Google Maps, Uber, Lyft, Hailo) and Segregating data for providing financial advice through Financial Products (Intuit).

2. Advantages:

- **High Throughput –** Amount of data getting traversed over the network is very high
- **Massive Scalability –** As the engine is distributed system, scalability is massive which means the need for additional storage systems can be scaled easily based on the amount of incoming data
- **High Availability –** As the data is distributed across different nodes in the cluster, when one node goes down, the other node can be picked up to respond to the write or read requests. Also, the cluster can be distributed across different data centers. Once a data center goes down, the cluster can be accessed from other data centers.

3. Architecture:

All the nodes present in a single network is called a Cluster. Every node present in a cluster, has the same functionality as the others.



As Cassandra is a distributed database system, data is distributed across all nodes and multiple servers which gives doorway to horizontal scalability.

Each node exchanges information with other nodes across the cluster.

After every write, commit log ensures data durability by capturing the write activity.

Data to be written to the node are first indexed and written to in-memory table called **MemTable** (can be called as "Write Back Cache").

During, the process of writing the data into the MemTable, when the MemTable is full, data is then written to the disk called **SSTable**.

All writes are automatically partitioned and distributed across all nodes throughout the cluster.

Cassandra periodically consolidates SS Table by flushing out unwanted data using a process called **"Compaction"**. **Tombstone** is marker in a row that indicates a column was deleted. Hence, Cassandra is a row oriented database.

An **Authorized** person can connect to any node and access data using **CQL**.

In Cassandra, typically a cluster has **One Keyspace** per application. When a client gets connected to a Node, that Node serves as a coordinator for that client operation. A coordinator typically decides what nodes to be requested in the ring for fetching the data based on the partitioning of data and placement of replicas.

4. Learning about Snitch:

Snitch is the knowledge of the nodes knowing about the location of other nodes in a cluster. The snitch method is specified in the main Cassandra configuration file. Different snitch methods are:

- **Dynamic Snitching**
- **SimpleSnitch**
- **RackInferringSnitch**
- **PropertyFileSnitch**
- **GossipingPropertyFileSnitch**
- **EC2Snitch**
- **EC2MultiRegionSnitch**

The most common type of snitch used is GossipingPropertyFileSnitch. A unique characteristic that it possesses is that it uses Cassandra-rackdc.properties file to specify the rack information and the data center information for a node. To deploy nodes or group of nodes in only one data center, SimpleSnitch is used. Basically, snitch can be used to place the replicas across different nodes in different data centers.

5. Learning about Gossip:

Gossiping is an act of talking with each other in a way to know about some information which is internally circulating amongst a group of people. Likewise, Gossip in Cassandra is a way how nodes in a cluster communicate with each other. It happens every one second having one node

communicating with up to three nodes in a cluster. During a Gossip, a node shares information about its state (such as up, down) and states of the other three nodes, exchanges information that it had over previous gossip exchanges and so on. It is a way of internal communication with nodes in a cluster.

6. Learning about Data Distribution:

Typically, nodes carry hash value generated by the partitioner (Murmur 3 partitioner is a default partitioner). The value ranges from -2^63 to 2^63. In addition to this, the row of data also carries a hash value ranging from -2^63 to 2^63.

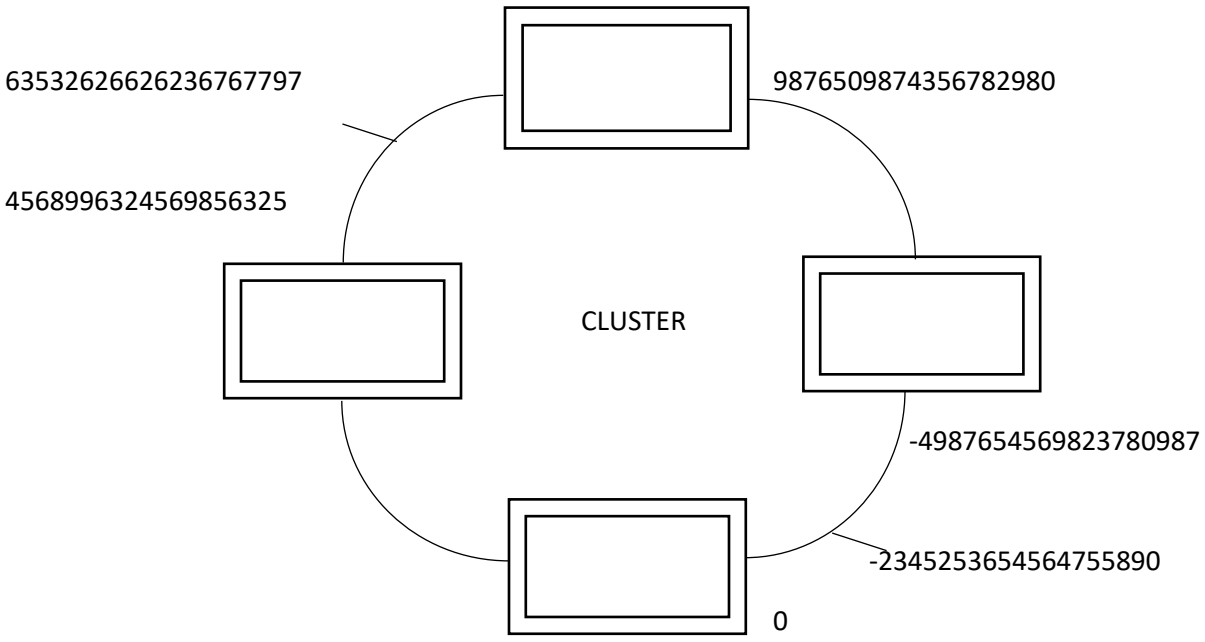
Let us say, we need to insert the following data into the nodes of the Cassandra cluster:

Event_ID	**Event Name**
E2035686	Executives Meet
E2035687	Client Meet

Here each row of values carries hash value generated by the Murmur partitioner. For example, the value from the Event_ID column carries the following hash values:

E2035686 → -2345253654564755890

E2035687 → 63532626626236767797



The token ranges can be calculated based on the number of nodes in a cluster. Let us say, we have 3 nodes in the cluster, the formula to calculate token range is {[(2^64/4) * i] – 2^63} where i ranges from 1 to 3.

7. Learning about Replication:

Replication ensures data availability. While creating a keyspace in Cassandra, there is an item called **"replication factor"** to be included in the keyspace definition. The two initial key spaces that will be present in the Cassandra database are, **system** and **system_traces.**

In cqlsh, type the following command:

DESCRIBE KEYSPACE system_traces;

The above command fetches he following output:

CREATE KEYSPACE system_traces WTH replication = {

'class': 'SimpleStrategy',

'replication_factor': '2'

};

The value of replication factor given above determines the number of nodes where the data is to be written. In the above description having replication factor to be 2, the data for system_traces is to be written across two nodes.

It is always better to have replication factor more than 1. That will ensure that the data will be written across 2 or more nodes depending upon the value given.

8. Learning about Virtual Nodes

Virtual Nodes(vnodes) ensures a node is responsible not just for one token range but for many small slices of token ranges. With more number of incoming data into the nodes, the cluster uniformity or distribution of data must be maintained to accommodate more data into many different clusters. For this purpose, vnodes are used. Usually, Nodes accommodate only the desired token range keyspace data. With vnodes, the token ranges are sliced into small ranges and the nodes are assigned to small slices of token ranges. This makes it more comfortable for the clusters to add new nodes eliminating the need to double the nodes with the increase in incoming data sets.