**Apache Cassandra**

Navya Devineni, Naga Anshitha Velagapudi, Soujanya Janapatla

School of Computer Science & Information Systems

Northwest Missouri State University

**Abstract**

The Apache Cassandra database is the right choice when you need scalability and high availability without compromising performance.

1. **Introduction**

Apache Cassandra is a distributed open source database that can be referred to as a "NoSQL database" or a "wide column store". Cassandra has been used by some of the world’s largest companies to leverage the service that expands the globe.

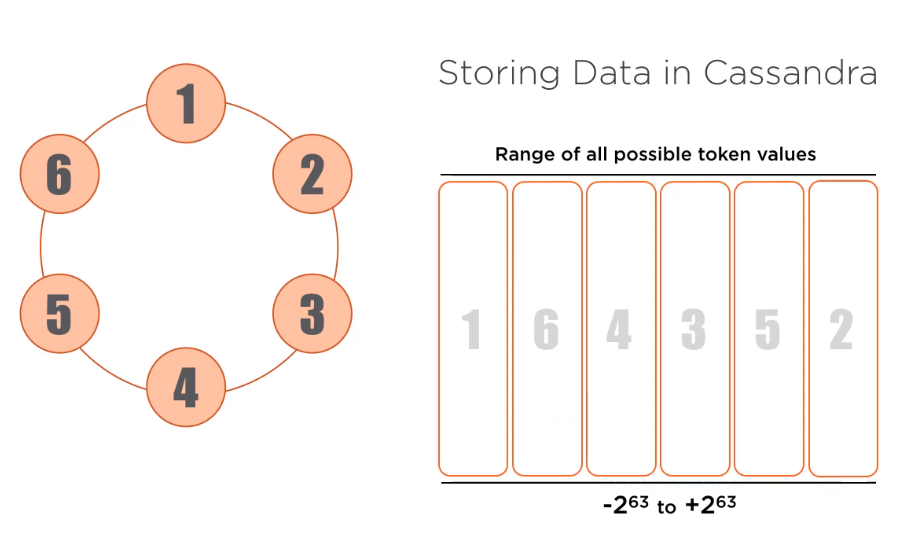
1. **What is Cassandra..?**

We shall start with the brief history of Cassandra. Cassandra was originally developed at Facebook, back in 2008. It was designed as the store for a Facebook feature Inbox Search. Such a feature requires a storage system that can handle a tremendous number of rights as well as geographical replication to reduce search latencies for end users.

Like most modern distributed NoSQL systems Cassandra was founded on the principal outline in 2 similar papers on the topic: Googles Cloud Big table and amazon Dynamo. Cassandra combines the distributed nature of Dynamo and data model of Googles big table. Facebook open sourced Cassandra in 2008 and in 2009 it became an apache incubator project in 2010 it graduated to top level apache project.

Netflix famously migrated from Oracle to Cassandra running in Amazon public cloud, Apple was also revealed to be a significant user of Cassandra running more than 75000 nodes and storing tons of Terra bytes of data.

1. **Cassandra Cluster**



This diagram is probably the most common representation of a Cassandra installation. Each dot on the circle is a Node, which represent a running instance of Cassandra. This diagram helps strive on the point that a Cassandra cluster is a true master-less peer to peer system with no single point of failure. All nodes can perform all Cassandra function.

1. **How Cassandra operates under the hood**

All data stored in Cassandra is associated with a token, there are astronomical number of possible token values, falls in this range -2 63 to + 2 63. As new node are added each node taking over a contiguous range of token values and storing the data associated with it. 2nd node, 3rd node, and so on till 6th node.

1. **Structure Data in Cassandra : Snitches**

A snitches is what Cassandra uses to gain an understanding of an environment physical or virtual, in which the cluster is been run. It is used efficiently to round requests and is consulted when storing multiples copies of the data. The default name is “Simple Snitch” is suitable for development in single data environments. A much more interesting snitch is the "Gossiping Property File Snitch" Gossip is the protocol Cassandra nodes use to talk to one another and keep everyone up to date on the state of the cluster.

1. **Consistency levels and Replication Strategies**

We looked at how a single copy of data is distributed across the cluster. Here we will discuss Replication Strategies to store multiple copies of data in a cluster. Cassandra support for Tunable Consistency, while reading and writing data to the cluster.

1. **Replication strategies**

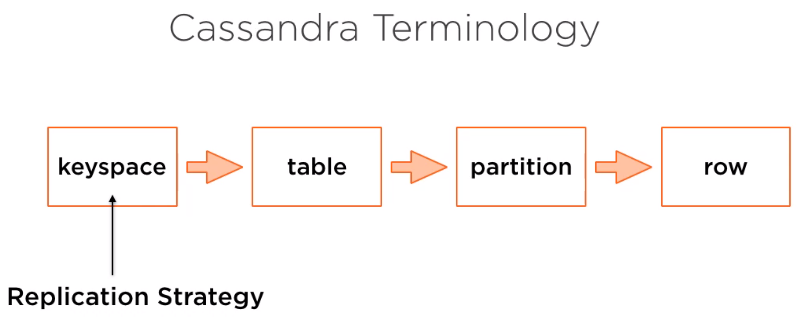
A virtual node shows data been written to a specific V node owned by a node in the cluster. In a Cassandra it is expect to store multiple copies of the data on different nodes throughout the cluster.

This gives increased reliability as well as performance. Not only it can easily tolerate a node becoming unavailable but in certain circumference we may choose to read a specific copy of the data from a node for example: In a data center geographically closer to the system making the query.

1. **Cassandra Terminology**

* Key spaces: At the highest level data in Cassandra in organized into key spaces. The closest analogy to this in the relational world would be oracle or MySQL Table space.
* Tables: With in a Cassandra key spaces there are one or more table. A table here is a pretty close match conceptually to it is relational counter-part.
* Partitions: All data written to Cassandra is associated with a partition key. This partition key determines where the data is located in the cluster, and all data in a partition is stored together.

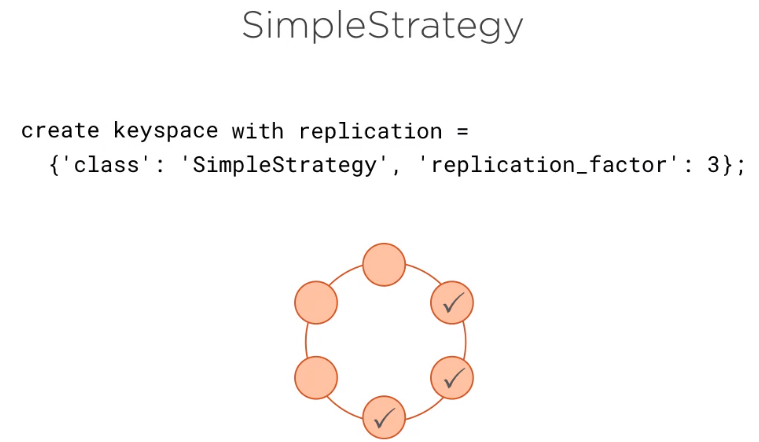
The partition is the primary interaction point when reading or writing data to Cassandra. Finally data within a partition may be represented as one or more rows.



The specifics of a replication strategy are to determine at the key space levels, if the partition key is used to determine the location of the first copy of the data written to a Cassandra cluster, the key space settings are used to determine the number of copies of the data and where they are stored throughout the cluster.

The two strategies for configuring this replication:

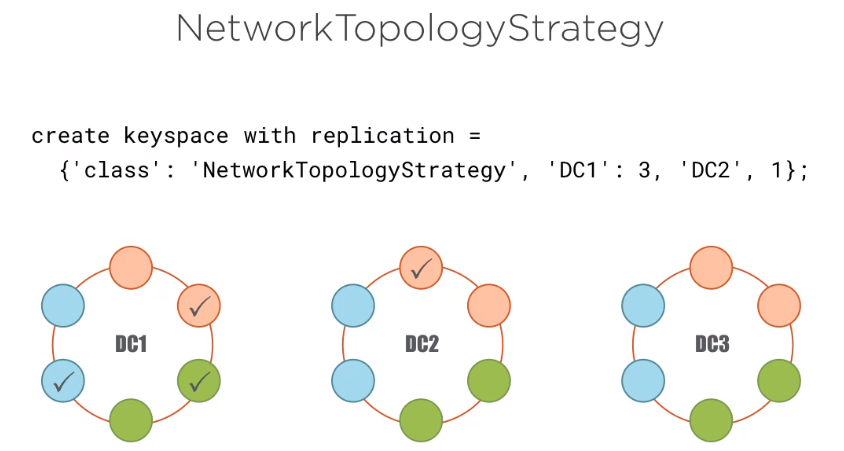
* Simple Strategy: As the name says it's the best used in development environments or single data center clusters. Example a SQL to create a key space with a simple strategy replication, we are asking Cassandra to store 3 copies of all the partitions in all the tables written to the cluster in this key space.



We can see a pattern here with Snitches and replication strategies working hand in hand. And a clustered configured with a simple Snitch. This replication strategy is already available, we can ask Cassandra to store multiple copies of data and it will do its best to store them on different nodes.

More interesting is network topology strategy, which is configured here.

* Network Topology Strategy: We are not simply specifying how many copies of data to store, but instead enumerating each data center and specifying how many copies of data are stored in each. In this case, we are storing 4 copies of data for each partition in each table in key space. But more than that we are telling Cassandra we tell to store 3 copies in Data Center 1 and 1 in Data center 2.



This replication strategy is ideal for production environments especially in multi data center clusters.

**Tunable consistency**:

In recent times distributed systems is facing with one of the challenges, i.e.; keeping replicas consistent together. Balancing availability and partitioning will be required for maintaining consistency. Providentially, Apache Cassandra is balancing by tuning according to our needs. In tuning consistency, the main part is CAP theorem.

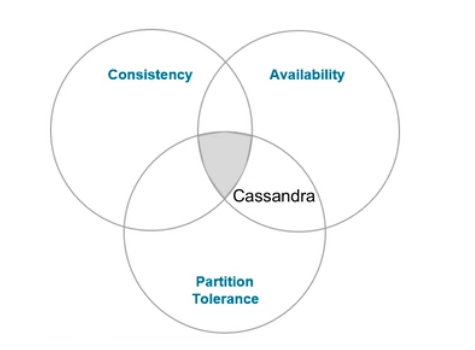
CAP Theorem:

CAP theorem has 3 properties, of which we can have 2 of 3 properties in any network shared-data in distributed systems.

C: stands for Consistency-At the same time, same data should be all nodes.

A: stands for High Availability-Here, the request we made must be addressed.

P: stands for tolerance to network partitions-Even in case of network partitions, the system should carry on to operate.



In a distributed environment, maintaining consistency is the hard one to achieve because in order to sync with each self data centers and across data centers. In the above figure Cassandra is following AP. It enhances Partition and Availability endurances itself but it is consistent, it is flexible by permitting us to tune it depending on how much persistently we need our data.

Below are the terminologies which we use:

* RF (Replication Factor) - Number of copies for data.
* CL (Consistency Level) - Number of nodes required to acknowledge the read or write.

So, the real question is how to tune consistency?

Consistency are bit of writes. In a while of writing data, we have to write the level of consistency with in which where we want to write and also read data, where we need to question for a particular level of consistency. This control has been given to the developer.

The example to write the data.

For example, RF = 2 the data is copied to 2 nodes.

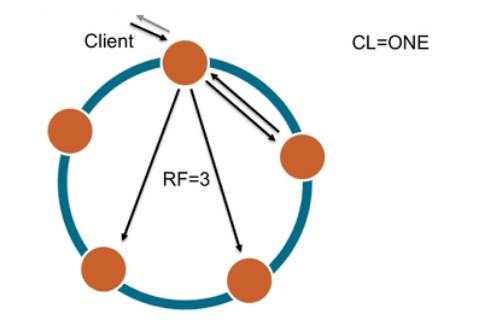
How to make sure that the data has been written totally to every node?

Yes, it is correct, we have to acknowledge that the work is completed that is provided exactly by CL.

**CL = ONE**

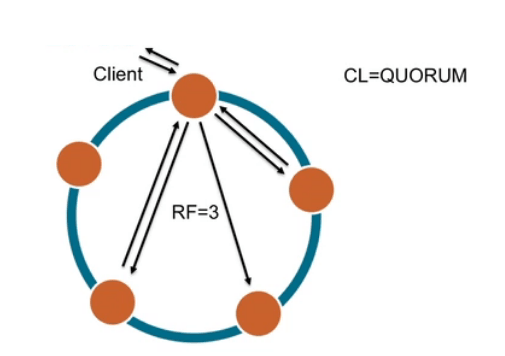
Here, one node is required to acknowledge to read and write.

* If acknowledgement from any of the node is received, that is marked as done, even though the data is being written simultaneously with the other two nodes. It is only one node’s acknowledgement that is included in a response.
* This is quick consistence level.



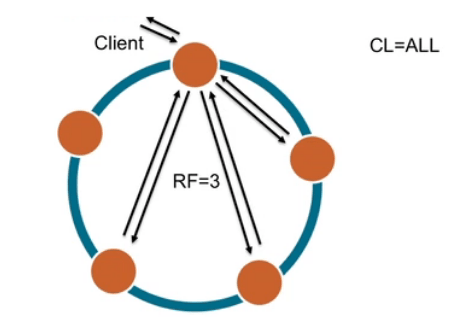
**CL = QUORUM**

* We need to acknowledge to write at least 51% of the nodes.
* Considering RF = 3 we need two nodes instead of three to acknowledge the write, the other third node we will be able to get the data simultaneously.
* QUORUM = (Replication Factor / 2)+1



**CL = ALL**

* It means all the nodes have to be acknowledge write.
* This is not yet recommended until and unless you need to create a case for it, as it is equal to off the tolerance of partition and available in a way to be consistent all the time.



* We can adjust the level of consistency according to our needs.

Different options we have are:

**Strong Consistency:**

The data which we had just written, should have to be available when we read it stating there is no exact data.

But, how the data is achieved?

* WRITE CL = ALL, READ CL = ONE: Each and everything is not recommended for production of environment as it is to make write slowly.
* WRITE CL = QUORUM, READ CL = QUORUM: it will be given high to read and write the accuracy without the sacrifice of availability.

**Eventual Consistency:**

* In background, copying the data to other nodes is done. So, it will be available to all nodes and has low latency approach which is very beneficial for analytical data, time-series data, and log data.

We can maintain consistency across multiple data centers with

**LOCAL QUORUM:**

* Writes can be accepted by only local replicas, but other data centers can also write the data which provides speed along with consistency.
* All accessible consistency levels in Cassandra from weakest to strongest are:
* ANY
* ONE, TWO, THREE
* QUORUM
* LOCAL\_ONE
* LOCAL\_QUORUM
* EACH\_QUORUM
* ALL(not available but for consistency)

The best CL that are being chosen are ONE, QUORUM, LOCAL\_ONE where consistency plays a vital role. Consistency and replication are fixed together because it is all about the working update of all the replicas on time and consistency level regulates the number of replicas that are required to acknowledge the success of read and write operation.