**Apache Cassandra**

Navya Devineni, Naga Anshitha Velagapudi, Soujanya Janapatla

School of Computer Science & Information Systems

Northwest Missouri State University

**Abstract**

This article describes the Apache Cassandra database. 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: Google's Cloud Big table and Amazon Dynamo. Cassandra combines the distributed nature of Dynamo and the data model of Google's big table. Facebook open sourced Cassandra in 2008 and in 2009 it became an apache incubator project and in 2010 it graduated to a 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 Terabytes of data.

1. **Cassandra Cluster**

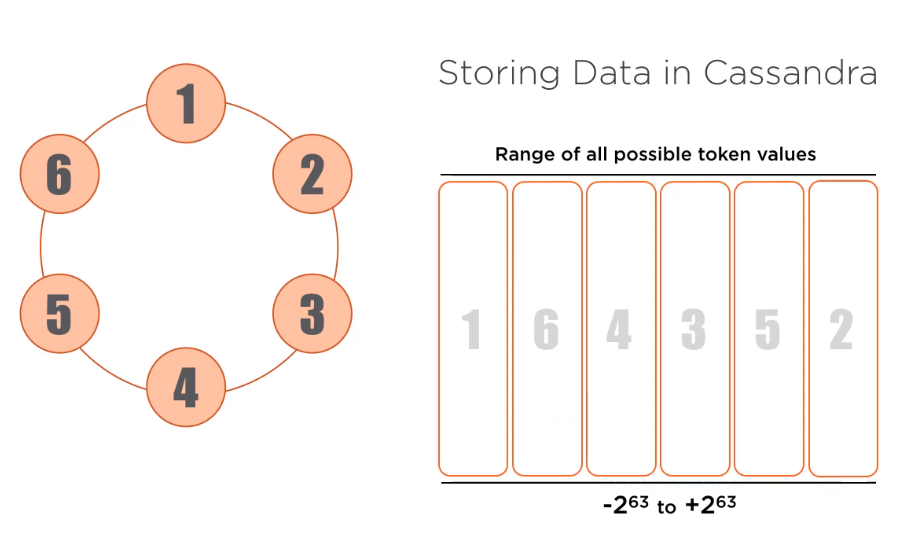


Figure 1: Cassandra Data storage

Figure 1 is probably the most common representation of a Cassandra installation. Each dot on the circle is a Node, which represents 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 functions.

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

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

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

A snitch is what Cassandra uses to gain an understanding of an environment, physical or virtual, in which the cluster is being run. It is used efficiently to round requests and is consulted when storing multiple 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.

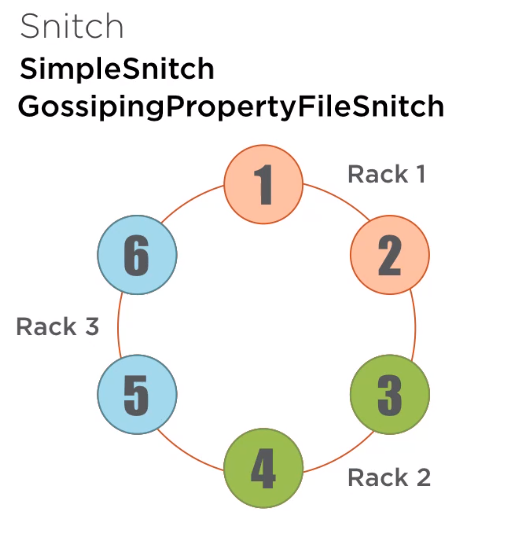


Figure 2: Snitch

**Basics of how Cassandra works**

Consistency levels and replication strategies

Cassandra Query Language

More recent additions

Tunable consistency

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 written to a specific V node owned by a node in the cluster. In Cassandra it is expected 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 circumstances 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**

**Keyspaces**: At the highest level data in Cassandra is organized into keyspaces. The closest analogy to this in the relational world would be oracle or MySQL Table space.

**Tables**: Within a Cassandra keyspaces there are one or more tables. A table here is a pretty close match conceptually to its relational counterpart.

**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.

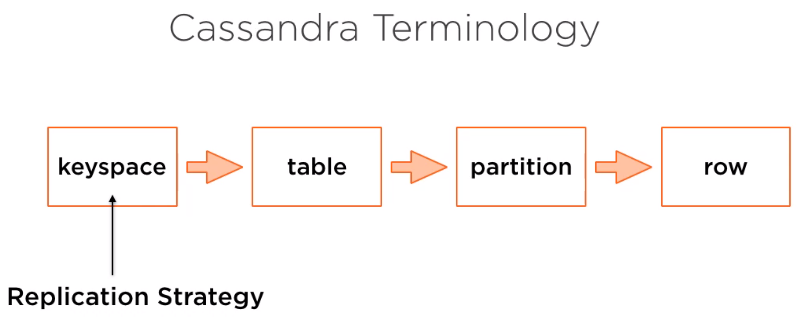


Figure 3: Terminology

The specifics of a replication strategy are to determine at the keyspace levels, if the partition key is used to determine the location of the first copy of the data written to a Cassandra cluster, the keyspace 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 is:

**2.1 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 keyspace 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 keyspace.

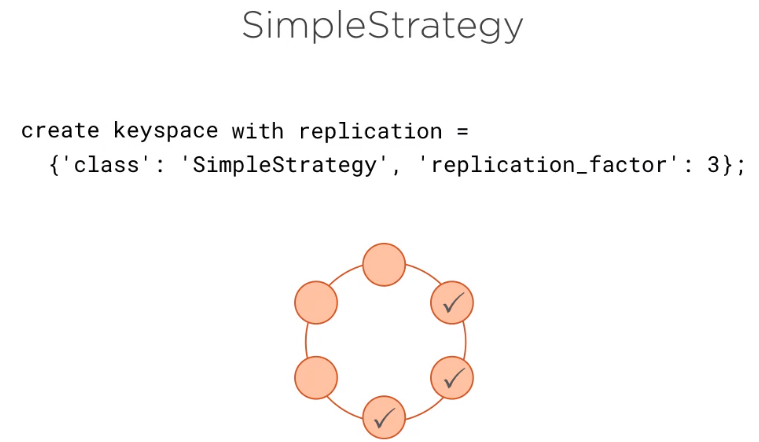


Figure 4: Simple Strategy

* We can see a pattern here with Snitches and replication strategies working hand in hand. And a clustered configuration 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.

**2.2 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 keyspace. But more than that we are telling Cassandra we tell to store 3 copies in Data Center 1 and 1 in Data center 2.

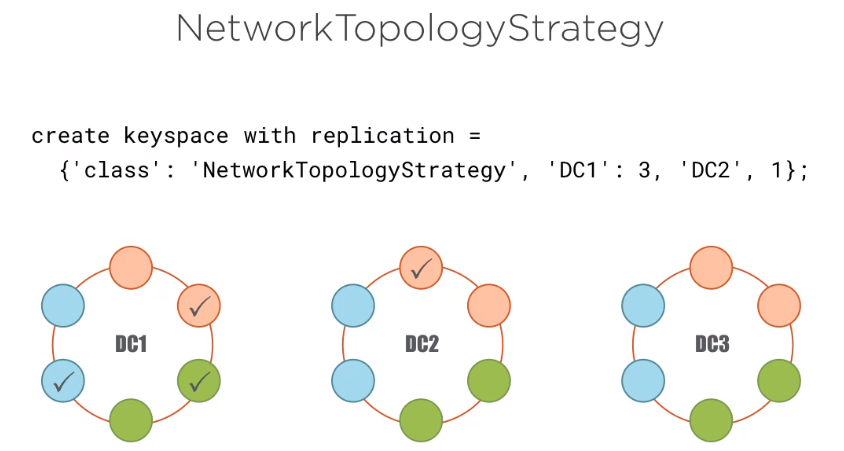


Figure 5: Network Topology Strategy

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

**2.3 Tunable consistency**:

In recent times distributed systems are facing 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 the 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 in 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.

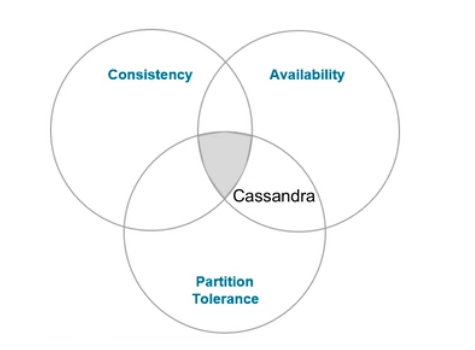


Figure 6: CAP Theorem

In a distributed environment, maintaining consistency is the hard one to achieve because in order to sync with each-self data centers and data centers which are across. In the above figure Cassandra is following AP. It enhances Partition and Availability endures itself but for it being consistent, we should know how much data is needed to tune and give us flexibleness to do it. Below are the terminologies which we use:

RF (Replication Factor) - Number of copies required for data.

CL (Consistency Level) - Number of nodes preferred to recognize the read or write.

**2.3.1 So, the real question is how to tune consistency?**

Consistency is bit of writes. When writing data, we have to write the level of consistency with which we have/want to write and also read data, where we need to question for a particular level of consistency. Developer has control over this.

The example to write the data.

Here is an 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 true that we have to acknowledge the work which is provided exactly by CL is completed.

1. **CL = ONE**

Here, one node is only required to acknowledge write or read.

If at all the acknowledgement from any of the nodes is received, that marked write as done, even though the data is being written simultaneously in the other two nodes. It is only one node’s acceptance that is included in a response.

This is a quick consistency level.

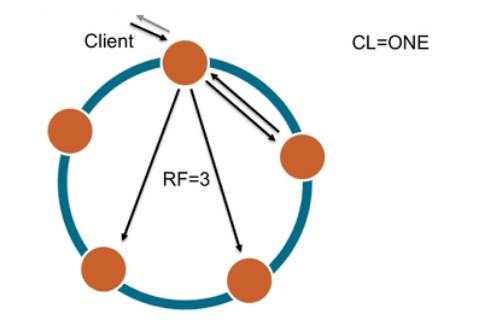


Figure 7: CL=ONE

1. **CL = QUORUM**

We need to acknowledge to write at least 51% of the nodes.

Considering RF = 3 we need two instead of three nodes to write for acknowledgement, the other third node we will be able to get the data simultaneously.

QUORUM = (Replication Factor / 2) +1

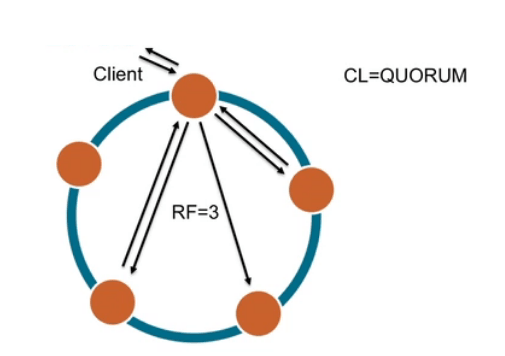


Figure 8: CL=QUORUM

1. **CL = ALL**

It means all the nodes have to acknowledge writing.

This is not yet recommended until and unless you need to create a case for it, as it is equal to the tolerance of partition and available in a way to be consistent all the time.

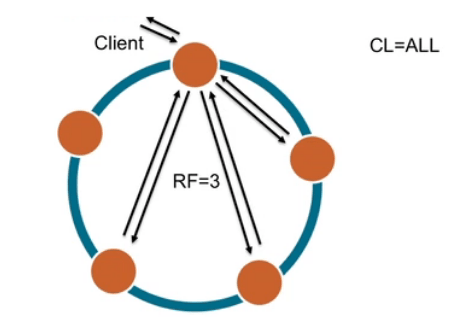


Figure 9: CL=ALL

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

Different options we have are:

**2.4 Strong Consistency:**

It should state that there is no stale data when the read operation is done when we add the data right then and should be visible.

**2.4.1 But, how is the data achieved?**

WRITE CL = ALL, READ CL = ONE: Each and everything is not recommended for production of the environment as it is to make writing slowly.

WRITE CL = QUORUM, READ CL = QUORUM: it will be given high to read and write the accuracy without the sacrifice of availability.

**2.5 Eventual Consistency:**

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

How exactly can we locate and maintain consistency from multiple data centers?

**3. 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 weak to strongest are:

* ANY
* ONE, TWO, THREE
* QUORUM
* LOCAL\_ONE
* LOCAL\_QUORUM
* EACH\_QUORUM
* ALL (not available but for consistency)

The best CLs that are being picked out are ONE, QUORUM, LOCAL\_ONE where consistency plays a vital role. Replication and consistency are fixed together because it is all about the working update of all the replicas on time and consistency level regulates the count of replicas that are required to admit the success of write and read operation.

.

**4. Advantages of Apache Cassandra:**

**Open source**: It is an open source project and free of cost. People who think alike share their views, ideas, and discussions regarding the topic Big Data.



Figure 10: Open Source

**Peer to Peer Architecture:** There is no failure point as Cassandra supports peer to peer architecture rather than master-slave. Requests can be entertained by any server from any client.

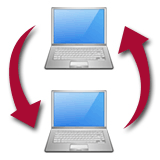


Figure 11: Peer to Peer Architecture

**Elastic Scalability:**  Major advantage of Cassandra is elastic scalability, as it can be scaled up or down easily. Nodes can be added or removed without disturbing the architecture.

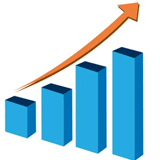
****

Figure 12: Elastic Scalability

**High availability and fault tolerance:** Replication (when data is stored at two or more locations) of data is another feature making Cassandra availability high and fault-tolerance. This tends to retrieve data from other locations, in the failure of nodes.



Figure 13: High availability and Fault Tolerance

**High Performance:**Basic idea in implementing Cassandra was to tackle the multicore machines capabilities that are hidden. Cassandra deals with a huge amount of data sets.



Figure 14: High Performance

**Column Oriented**:  In Cassandra, column names are also filled with original data and columns are stored depending on the names of the column. Rows also contain columns.

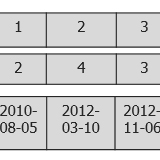


Figure 15: Column Oriented

**Schema-Free:** Cassandra data model is also popular as schema optional data model. You can customize columns as you wish within rows. It is popular for schema free or schema less databases.

**5. Conclusion:**

To sum up, several companies are using NoSQL databases compared to MySQL databases. Big Data problems are also solved by Cassandra, it is used to stream videos or to develop production applications used for business purposes. Apache Cassandra is best to use when you want to run your application globally.

**References:**

[1] [Cassandra website](http://cassandra.apache.org/doc/latest/tools/index.html)

[2] [Wikipedia](https://en.wikipedia.org/wiki/Apache_Cassandra)

[3] [Edureka](https://www.edureka.co/blog/apache-cassandra-advantages/)

[4] [Oodles Technologies](https://www.oodlestechnologies.com/blogs/Tuning-consistency-with-Apache-Cassandra/)