Apache Cassandra

## Introduction

Apache Cassandra is a free and open-source, distributed, wide column store, NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure.

Cassandra accommodates all possible data formats including: structured, semi-structured, and unstructured.

Cassandra was designed to run on cheap commodity hardware. It performs blazingly fast writes and can store hundreds of terabytes of data, without sacrificing the read efficiency.

**Distributed**: A distributed database is a database in which data is stored across different physical locations.

**Wide column store**: A wide-column store is a type of NoSQL database. It uses tables, rows, and columns, but unlike a relational database, the names and format of the columns can vary from row to row in the same table. A wide-column store can be interpreted as a two-dimensional key–value store.

**A single point of failure** is a part of a system that, if it fails, will stop the entire system from working.

* NoSQL databases allow developers to store huge amounts of unstructured data, giving them a lot of flexibility.

There are mainly 3 kinds of NoSQL databases:

* **Document databases** store data in documents similar to JSON (JavaScript Object Notation) objects. Each document contains pairs of fields and values. The values can typically be a variety of types including things like strings, numbers, booleans, arrays, or objects, and their structures typically align with objects developers are working with in code. Because of their variety of field value types and powerful query languages, document databases are great for a wide variety of use cases and can be used as a general purpose database. They can horizontally scale-out to accomodate large data volumes. *MongoDB* is an example.
* **Key-value** databases are a simpler type of database where each item contains keys and values. A value can typically only be retrieved by referencing its value, so learning how to query for a specific key-value pair is typically simple. Key-value databases are great for use cases where you need to store large amounts of data but you don’t need to perform complex queries to retrieve it. Common use cases include storing user preferences or caching. *Redis and DynamoDB* are popular key-value databases.
* **Wide-column** stores store data in tables, rows, and dynamic columns. Wide-column stores provide a lot of flexibility over relational databases because each row is not required to have the same columns. Many consider wide-column stores to be two-dimensional key-value databases. Wide-column stores are great for when you need to store large amounts of data and you can predict what your query patterns will be. Wide-column stores are commonly used for storing Internet of Things data and user profile data. *Cassandra and HBase* are two of the most popular wide-column stores.
* **Graph databases** store data in nodes and edges. Nodes typically store information about people, places, and things while edges store information about the relationships between the nodes. Graph databases excel in use cases where you need to traverse relationships to look for patterns such as social networks, fraud detection, and recommendation engines. *Neo4j and JanusGraph* are examples of graph databases.

## Architecture

The design goal of Cassandra is to handle big data workloads across multiple nodes without any single point of failure. Cassandra has peer-to-peer distributed system across its nodes, and data is distributed among all the nodes in a cluster.

* All the nodes in a cluster play the same role. Each node is independent and at the same time interconnected to other nodes.
* Each node in a cluster can accept read and write requests, regardless of where the data is actually located in the cluster.
* When a node goes down, read/write requests can be served from other nodes in the network.

In Cassandra, one or more of the nodes in a cluster act as replicas for a given piece of data. If it is detected that some of the nodes responded with an out-of-date value, Cassandra will return the most recent value to the client. After returning the most recent value, Cassandra performs a **read repair** in the background to update the stale values.

Cassandra uses the **Gossip Protocol** in the background to allow the nodes to communicate with each other and detect any faulty nodes in the cluster.

### Key Components

* **Node** − It is the place where data is stored.
* **Data center** − It is a collection of related nodes.
* **Cluster** − A cluster is a component that contains one or more data centers.
* **Commit log** − The commit log is a crash-recovery mechanism in Cassandra. Every write operation is written to the commit log.
* **Mem-table** − A mem-table is a memory-resident data structure. After commit log, the data will be written to the mem-table. Sometimes, for a single-column family, there will be multiple mem-tables.
* **SSTable** − It is a disk file to which the data is flushed from the mem-table when its contents reach a threshold value.
* **Bloom filter** − These are nothing but quick, nondeterministic, algorithms for testing whether an element is a member of a set. It is a special kind of cache. Bloom filters are accessed after every query.

Users can access Cassandra through its nodes(Nodes as in the servers, not graph nodes!) using **Cassandra Query Language** **(CQL)**. CQL treats the database (Keyspace) as a container of tables. Programmers use **cqlsh**: a prompt to work with CQL or separate application language drivers.

### Write Operations

Every write activity of nodes is captured by the commit logs written in the nodes. Later the data will be captured and stored in the mem-table. Whenever the mem-table is full, data will be written into the SStable data file. All writes are automatically partitioned and replicated throughout the cluster. Cassandra periodically consolidates the SSTables, discarding unnecessary data.

### Read Operations

During read operations, Cassandra gets values from the mem-table and checks the bloom filter to find the appropriate SSTable that holds the required data.

## Data Model

Cassandra arranges the nodes in a cluster, in a ring format, and assigns data to them.

### Keyspace

Keyspace is the outermost container for data in Cassandra. The basic attributes of a Keyspace in Cassandra are −

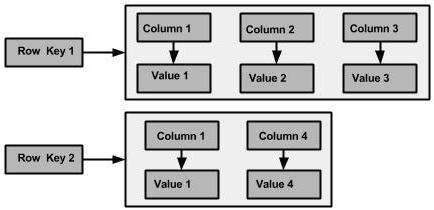
* **Replication factor** − It is the number of machines in the cluster that will receive copies of the same data.
* **Replica placement strategy** − It is nothing but the strategy to place replicas in the ring. We have strategies such as simple strategy (rack-aware strategy), old network topology strategy (rack-aware strategy), and network topology strategy (datacenter-shared strategy).
* **Column families** − Keyspace is a container for a list of one or more column families. A column family, in turn, is a container of a collection of rows. Each row contains ordered columns. Column families represent the structure of your data. Each keyspace has at least one and often many column families.

|  |  |
| --- | --- |
| **Relational Table** | **Cassandra column Family** |
| A schema in a relational model is fixed. Once we define certain columns for a table, while inserting data, in every row all the columns must be filled at least with a null value. | In Cassandra, although the column families are defined, the columns are not. You can freely add any column to any column family at any time. |
| Relational tables define only columns and the user fills in the table with values. | In Cassandra, a table contains columns, or can be defined as a super column family. |

A Cassandra column family has the following attributes −

* **keys\_cached** − It represents the number of locations to keep cached per SSTable.
* **rows\_cached** − It represents the number of rows whose entire contents will be cached in memory.
* **preload\_row\_cache** − It specifies whether you want to pre-populate the row cache.

An Example of a column family:



### Column

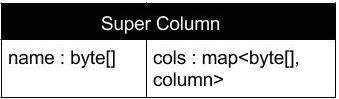
A column is the basic data structure of Cassandra with *three values*, namely **key or column name, value, and a time stamp**. Given below is the structure of a column.



### SuperColumn

A super column is a special column, therefore, it is also a key-value pair. But a super column stores a map of sub-columns.

Generally column families are stored on disk in individual files. Therefore, to optimize performance, it is important to keep columns that you are likely to query together in the same column family, and a super column can be helpful here.



|  |  |
| --- | --- |
| In RDBMS, a table is an array of arrays. (ROW x COLUMN) | In Cassandra, a table is a list of “nested key-value pairs”. (ROW x COLUMN key x COLUMN value) |