**Cassandra**

# High-level Architecture

Diagram

Description automatically generated

### Data partitioning

Cassandra uses **consistent hashing** for data partitioning.

### Cassandra keys

A picture containing text

Description automatically generated

### Clustering keys

Table

Description automatically generated

### Partitioner

Partitioner is the component responsible for determining how data is distributed on the Consistent Hash ring.

Diagram

Description automatically generated

Cassandra uses the Murmur3 hashing function. Murmur3 will always produce the same hash for a given partition key. This means that we can always find the node where a specific row is stored. Cassandra does allow custom hashing functions, however, once a cluster is initialized with a particular partitioner, it cannot be changed later.

### Coordinator node

Diagram, schematic

Description automatically generated

# Replication

Text

Description automatically generated with medium confidence

### Simple replication strategy

Diagram, schematic

Description automatically generated

### Network topology strategy

Diagram, schematic

Description automatically generated

# Cassandra Consistency Levels

Cassandra’s consistency level is defined as the minimum number of Cassandra nodes that must fulfil a read or write operation before the operation can be considered successful.

Cassandra allows us to specify different consistency levels for read and write operations. Also, Cassandra has tunable consistency, i.e., we can increase or decrease the consistency levels for each request.

## Write consistency levels

**How does Cassandra perform a write operation?** For a write, the coordinator node contacts all replicas, as determined by the replication factor, and considers the write successful when a number of replicas equal to the consistency level acknowledge the write.

### Hinted handoff

Diagram, schematic

Description automatically generated

When a node is down or does not respond to a write request, the coordinator node writes a hint in a text file on the local disk. This hint contains the data itself along with information about which node the data belongs to. When the coordinator node discovers from the Gossiper (will be discussed later) that a node for which it holds hints has recovered, it forwards the write requests for each hint to the target. Furthermore, each node every ten minutes checks to see if the failing node, for which it is holding any hints, has recovered.

One thing to remember: When the cluster cannot meet the consistency level specified by the client, Cassandra fails the write request and does not store a hint.

## Read consistency levels

The consistency level for read queries specifies how many replica nodes must respond to a read request before returning the data.

**Snitch:** The Snitch is an application that determines the proximity of nodes within the ring and tells which nodes are faster. Cassandra nodes use this information to route read/write requests efficiently.

Diagram, schematic

Description automatically generated

**How does Cassandra perform a read operation?** The coordinator always sends the read request to the fastest node. For example, for Quorum=2, the coordinator sends the request to the fastest node and the digest of the data from the second-fastest node. The digest is a checksum of the data and is used to save network bandwidth.

If the digest does not match, it means some replicas do not have the latest version of the data. In this case, the coordinator reads the data from all the replicas to determine the latest data. The coordinator then returns the latest data to the client and initiates a **read repair** request. The read repair operation pushes the newer version of data to nodes with the older version.

Diagram, schematic

Description automatically generated

## Snitch

Snitch keeps track of the network topology of Cassandra nodes. It determines which data-centers and racks nodes belong to. Cassandra uses this information to route requests efficiently.

Here are the two main functions of a snitch in Cassandra:

* Snitch determines the proximity of nodes within the ring and monitors the read latencies to avoid reading from nodes that have slowed down.
* Cassandra’s replication strategy uses the information provided by the Snitch to spread the replicas across the cluster intelligently.

# Gossiper

Let's explore how Cassandra uses gossip protocol to keep track of the state of the system.

Gossip protocol is a peer-to-peer communication mechanism in which nodes periodically exchange state information about themselves and other nodes they know about.

## How does Cassandra use gossip protocol?

Diagram

Description automatically generated

## Node failure detection

Accurately detecting failures is a hard problem to solve as we cannot say with 100% surety that if a system is genuinely down or is just very slow in responding due to heavy load, network congestion, etc. Mechanisms like Heartbeating outputs a boolean value telling us if the system is alive or not; there is no middle ground.

Heartbeating uses a fixed timeout, and if there is no heartbeat from a server, the system, after the timeout, assumes that the server has crashed.

Here the value of the timeout is critical. If we keep the timeout short, the system will be able to detect failures quickly but with many false positives due to slow machines or faulty networks. On the other hand, if we keep the timeout long, the false positives will be reduced, but the system will not perform efficiently for being slow in detecting failures.

# Cassandra's Write Operation

Cassandra stores data both in memory and on disk to provide both high performance and durability. Every write includes a timestamp.

summary of Cassandra’s write path:

1. Each write is appended to a **commit log**, which is stored on disk.
2. Then it is written to **MemTable** in memory.
3. Periodically, MemTables are flushed to **SSTables** on the disk.
4. Periodically, compaction runs to merge SSTables.

## Commit log

Diagram

Description automatically generated

## MemTable

* Each node has a MemTable in memory for each Cassandra table.
* Each MemTable contains data for a specific Cassandra table, and it resembles that table in memory.
* Each MemTable accrues writes and provides reads for data not yet flushed to disk.
* Commit log stores all the writes in sequential order, with each new write appended to the end, whereas MemTable stores data in the sorted order of partition key and clustering columns.
* After writing data to the Commit Log and MemTable, the node sends an acknowledgment to the coordinator that the data has been successfully written.

Chart, diagram

Description automatically generated

## SStable

When the number of objects stored in the MemTable reaches a threshold, the contents of the MemTable are flushed to disk in a file called SSTable.

At this point, a new MemTable is created to store subsequent data.

When the MemTable is flushed to SStables, corresponding entries in the Commit Log are removed.

Here is the summary of Cassandra’s write path:

Diagram

Description automatically generated with medium confidence

# Cassandra's Read Operation

Let’s dig deeper into the components involved in Cassandra’s read path.

## Caching

To boost read performance, Cassandra provides three optional forms of caching:

**Row cache:** The row cache, caches frequently read (or hot) rows. It stores a complete data row, which can be returned directly to the client if requested by a read operation.

**Key cache:** Key cache stores a map of recently read partition keys to their SSTable offsets. This facilitates faster read access into SSTables stored on disk and improves the read performance.

**Chunk cache:** Chunk cache is used to store uncompressed chunks of data read from SSTable files that are accessed frequently.

## Reading from MemTable

Let’s take an example. Here we have two partitions of a table with partition keys ‘2’ and ‘5’. The clustering columns are the state and city names. When a read request comes in, the node performs a binary search on the partition key to find the required partition and then return the row.

Table

Description automatically generated

Here is the summary of Cassandra’s read path:

Diagram

Description automatically generated with medium confidence

## Reading from SSTable

Each SStable has a Bloom filter associated with it, which tells if a particular key is present in it or not. Bloom filters are used to boost the performance of read operations. Bloom filters are very fast, non-deterministic algorithms for testing whether an element is a member of a set. They are non-deterministic because it is possible to get a false-positive read from a Bloom filter, but false-negative is not possible.

So, in a way, a Bloom filter is a special kind of key cache.

Cassandra maintains a Bloom filter for each SSTable. When a query is performed, the Bloom filter is checked first before accessing the disk. Because false negatives are not possible, if the filter indicates that the element does not exist in the set, it certainly does not; but if the filter thinks that the element is in the set, the disk is accessed to make sure.

### How are SSTables stored on the disk?

Each SSTable consists of two files

* Data File
* Partition Index File

Table

Description automatically generated

### Partition index summary file

Table

Description automatically generated

### Reading SSTable through key cache

Diagram

Description automatically generated

Here is the summary of Cassandra’s read operation:

1. First, Cassandra checks if the row is present in the Row Cache. If present, the data is returned, and the request ends.
2. Second, check the bloom filter. If bloom filter indicates that data is present in bloom filter than SSTable is checked for Data offset
3. Third, Key Cache is checked for the partition Index. If found could be used for fetching the data from data offset
4. Cassandra continues to seek the partition in the partition summary and partition index. These structures also provide the partition offset in an SSTable which is then used to retrieve the partition and return.

Diagram

Description automatically generated

# Compaction

## How does compaction work in Cassandra?

 SSTables are immutable, which helps Cassandra achieve such high write speeds.

Flushing of MemTable to SStable is a continuous process. This means we can have many SStables lying on the disk.

 Compaction in Cassandra refers to the operation of merging multiple related SSTables into a single new one. During compaction, the data in SSTables is merged: the keys are merged, columns are combined, obsolete values are discarded, and a new index is created.

Chart, diagram

Description automatically generated

## Summary

1. **distributed**, **decentralized**, **scalable**, and **highly available** NoSQL database.
2. designed with the understanding that software/hardware **failures can and do occur**.
3. Cassandra is a **peer-to-peer** distributed system. No Single Point of Failure
4. Data is replicated across the nodes for fault tolerance and redundancy.
5. Cassandra uses the **Consistent Hashing** algorithm to distribute the data among nodes
6. Cassandra utilizes the data model of Google’s Bigtable, i.e., **SSTables** and **MemTables**.
7. Cassandra utilizes distributed features of Amazon’s Dynamo, i.e., consistent hashing, partitioning, and replication.
8. Cassandra offers **Tunable consistency** for both read and write operations to adjust the trade-off between availability and consistency of data.
9. Cassandra uses the **gossip protocol** for inter-node communication.