Cassandra course

Apache Cassandra is a distributed, decentralized, scalable and highly available nosql database.

- **Distributed:** Capable of running on multiple machines.
- **Decentralized:**
    - No master slave
    - Identical nodes
    - No single point of failure
    - High Availability
- **Scalable:**
    - Add or remove nodes
    - Little performance impact
    - No manual intervention
    - No manual rebalancing

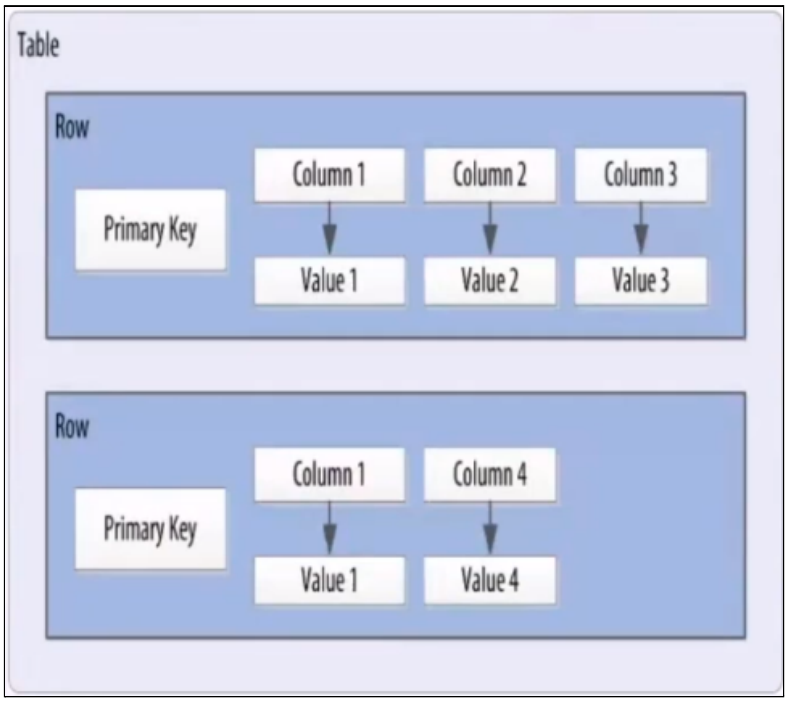
When to use it?

- Large and growing data
- High Availability is important
- Changing data
- Consistency is not a concern
- Faster Writes

Astra DB

- Cassandra as a Service
- Helps you deploy on Google, AWS & Azure in just minutes
- No maintenance/administration
- Pay for what you use model

Data Model



- **Column:** name value pair
- **Row**: container for columns referenced by primary key, important thing to note is that if any column has no value, we don't need to store that column.
- **Table:** container for rows
- **Keyspace:** container for table that span one or more nodes
- **Cluster:** container for keyspace

Constraints

- No joins
- No foreing keys
- No flexible queries

Create keyspace

```
CREATE KEYSPACE <keyspacename>
        WITH REPLICATION {
                class: 'NetworkTopologyStrategy'
                replication_factor: '3'
        }
```

The field replication is used to duplicate your data across your cluster so that you don't lose your data in case of any machine failure.

To see the keyspace in the CQL console: *describe keyspaces;*
And for uses it: *use <keyspacename>*

Create Table

```
CREATE TABLE employee (
        emp_Id Int,
        emp_name text,
        emp_age int,
        PRIMARY KEY((emp_id))
)
```

To see all the tables for a keyspace, we uses the command: *describe tables*
For seeing the table definition we also use the describe command but with the table name.

Insert data

```
INSERT INTO employee (emp_id, emp_name, emp_age)
values (20, 'john', 35)
```

To see the information we can use a select command like select * from employee ;

One important thing for the select clause is we can use only the primary key for the where clause.

Update data

```
UPDATE employee
        set emp_age = 35
        where emp_id = 10;
```

Delete

```
DELETE from employee where emp_id=10;
```

Cassandra Keys

In Cassandra, Primary key is made up of two parts:

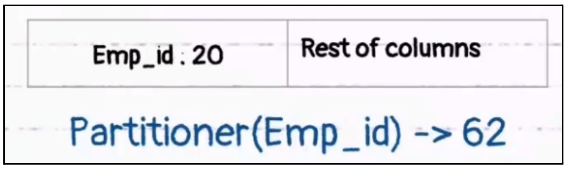
- **Partition key:** decides how data is distributed across nodes.
- **Clustering key:** decide how data is stored on a single machine.

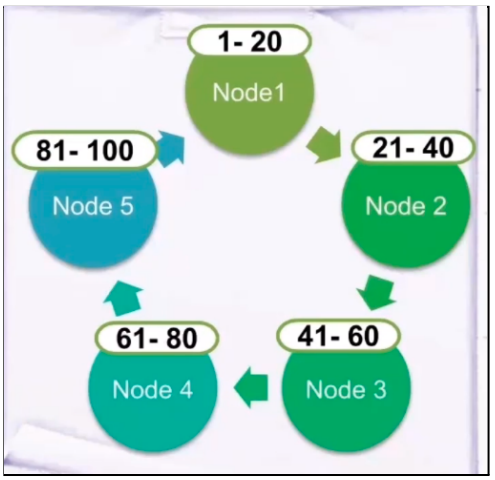
How data is distributed?
Every node is assigned a unique token/range of token that determines which row will go to which node.

How token is generated?
A hashing function called partitioner is used to generate token values.

Example

- Token range = 1-100
- Number of nodes = 5
- Token range per node = 100/5





Suppose we have a row with primary key pk1 and clustering key ck1-a and ck1-b. The partition key will be stored first on the disk, followed by the clustering key and then followed by the rest of the column.

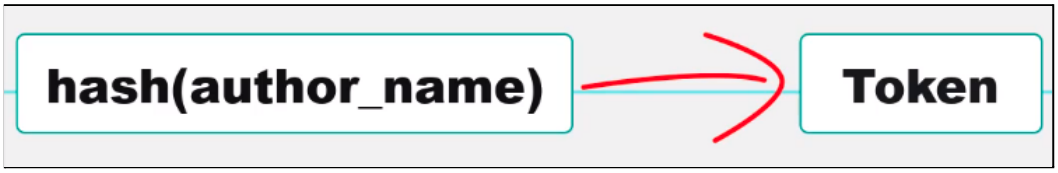
If multiple rows belong to the same partition key, they will be stored one after another. The rows are sorted by the values of clustering keys.

pk1	ck1-a	ck1-b	Rest of columns
	ck2-a	ck2-b	Rest of columns
	ck3-a	ck3-b	Rest of columns
	ck4-a	ck4-b	Rest of columns

Partition/Clustering key hands on

```
CREATE TABLE books_by_author(
        author_name text,
        publish_year int,
        book_id uuid,
        book_name text,
        rating float,
primary key((author_name),publish_year, rating))
WITH CLUSTERING ORDER BY(publish_year DESC, rating ASC);
```

The column in the parentheses defines the partitioning key, which means the hashing algorithm will be applied on the author's name and also means that all the records belonging to the same water will always line up in the same machine.



primary key((author_name),publish_year, rating)

Publish_year and the rating are the clustered keys.

```
token@cqlsh:firstkeyspace> SELECT * FROM books_by_author;

 author_name    | publish_year | rating | book_id                              | book_name
----------------+--------------+--------+--------------------------------------+----------------------
 James peterson |         2021 |      4 | f2ae6e28-e06a-498c-937c-4f381e5146f2 |         The Red Book
 James peterson |         2018 |    4.5 | 8a15bf01-4fc4-4f3b-a518-a151f21de1fb | President is Missing
 James peterson |         2008 |    3.5 | 58380f6d-851e-40c6-b3be-4910798951ec |        Roses are Red
 James peterson |         2008 |      4 | 84a3600a-b556-49be-aac3-ed592bdc7c86 |       Witch & Wizard
 James peterson |         2008 |    4.5 | 21a6f160-2b74-4322-8251-e5ed2a9f23c7 |        Cross Country
```

For reading the data from a table with a composite primary key only partitioning key is mandatory if we are using the where clause, because the partition key indicates which machine has the data. If we don't provide it, Cassandra will have to search in all the machines for the record which will be very slow and inefficient.

Cassandra Data Types

- **Numeric Data Type:** int, bigint, smallint, tinyint, varint, float, double, decimal.
- **Textual data type:** text, varchar.
- **Collection data type:** Set, List, Map.
- **Other data types:** boolean, blob, uuid, timeuuid, user defined.

UUID

- Universally unique identifier
- 128-bit hexadecimal value
- Prevent duplicates.
- Convenience function: uuid()

TimeUUID

- Unique identifier based on the MAC address, system time & a sequence number.
- Prevent duplicates.
- Convenience functions: now(), dateOf(), etc.

Set DataType

- Collection of items
- No duplicates
- Insertion Order is not maintained

List DataType

- Collection of items
- Duplicates allowed
- Insertion Order is maintained

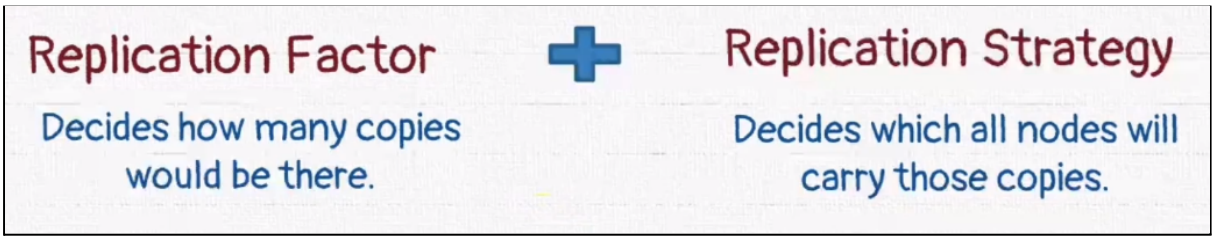
Map DataType

- Collection of key-value pairs

Cassandra Architecture

Replication

Replication is must for high availability



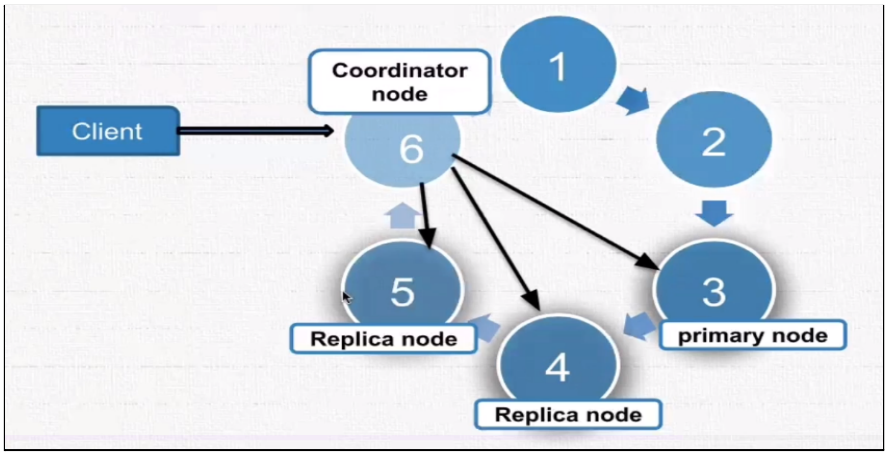
When we created our keyspace we set both of this params

SimpleStrategy / Replication Factor: 3

The client initiates the write request with the coordinator node, which is based on the partitioning key where it writes a token value. Suppose that the token value maps to the node number 3, so this node will become the primary node for that data.

If the strategy is a simple strategy the replica nodes are consecutives from the primary.

CREATE KEYSPACE firstkeyspace
WITH replication = {'class' : 'SimpleStrategy', 'replication_factor': '3'}

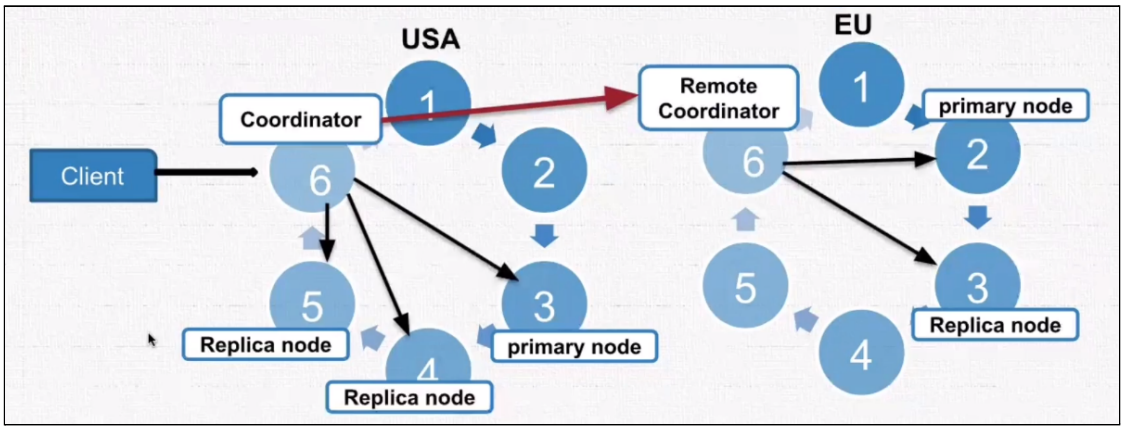


NetworkTopologyStrategy

This kind of strategy is used in case of multiple data centers. It allows us to specify different replication factors for different data centers.

In this case the coordinator will find the primary node and the replica nodes and send them the request but also will find a remote coordinator in another datacenter and send the write request. This remote coordinator will behave equal to the coordinator in this data center.

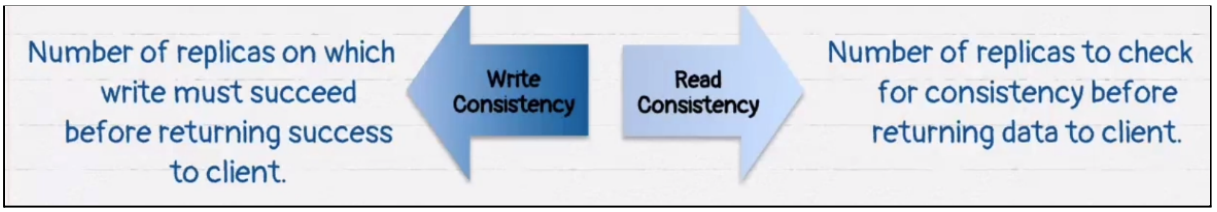
CREATE KEYSPACE firstkeyspace
WITH replication = {'class' : 'NetworkTopologyStrategy', 'USA': '3', 'EU': '2'}



Consistency

Cassandra has tunable consistency:

- Increase or decrease consistency
- Trade off between consistency & performance
- Configure consistency for read & write separately

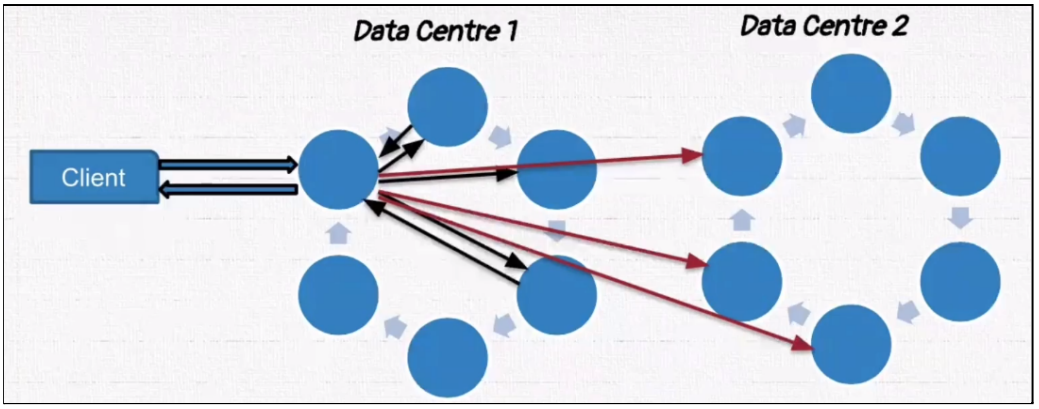


Write Consistency

Write consistency has 4 consistency level:

- **One:** coordinator will wait for only one replica node to complete.
- **All:** coordinator will wait for all replica nodes to complete.
- **Quoram:** if quoram value 2, coordinator will wait for exactly two replica nodes to complete.
- **Local_Quoram:** if local_quoram is 2, 2 replicas from the same datacenter as where the coordinator node resides should complete the write operation before coordinator node sends success notification to client.

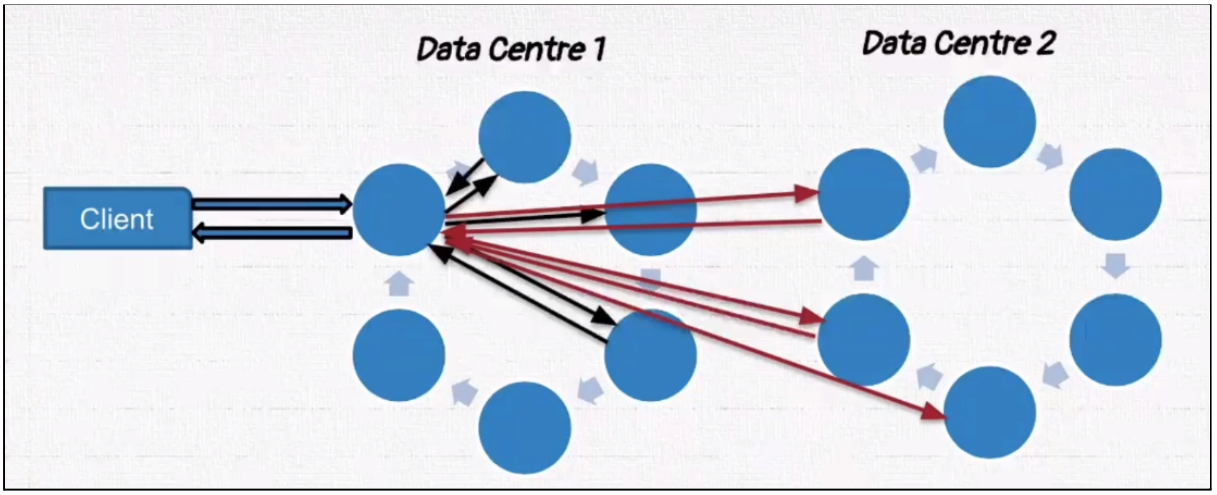
  The client sends the write request to the coordinator node, which will send the request simultaneously to all the replicas in data center 1 and data center 2. But as soon as it gets 2 response back from 2 replicas from data center one, it sends the success notification back to the client not waiting for the data center 2 response, so it will wait only for the local replicators to response.



One more write consistency level is **each_quoram:**

If each_quoram is 2, 2 replicas per datacenter needs to complete the write operation before coordinator node sends success notification to client.

If the client sends the write request coordinator, it sends the request to all the replica nodes in the data center 1 and data center 2 and it will wait for a response from both data centers two responde for each before sending the success notification back to the client.

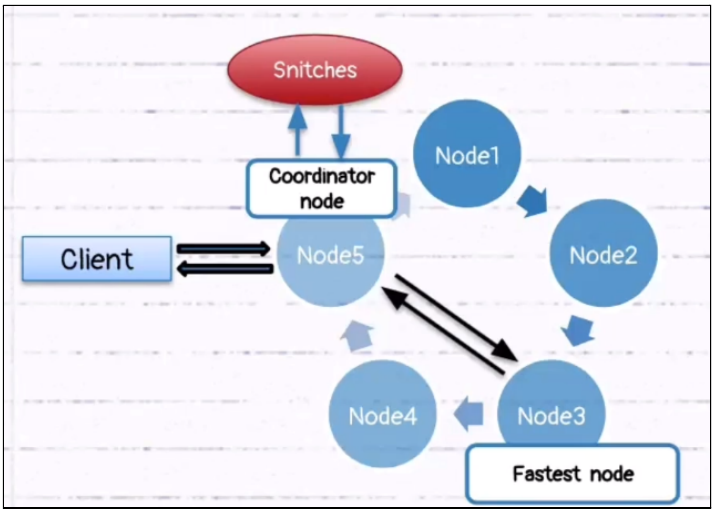


Read Consistency

Consistencies levels:

- **One:** coordinator will send requests only to the fastest node to get requested data.

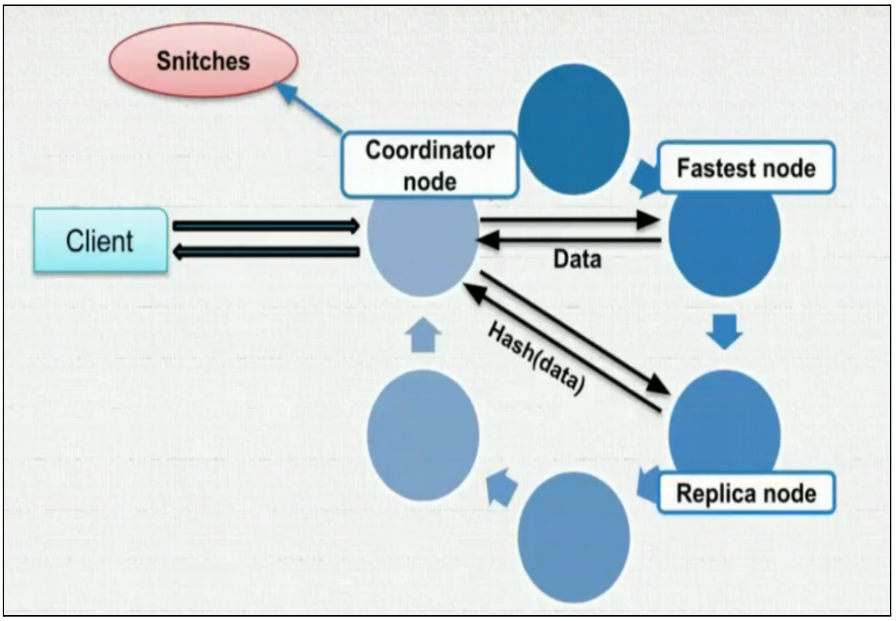
  To know which is the fastest node, the coordinator node consult that from **snitches** which is a program that has all the information about the network topology and knows the fastest route or the fastest or the nearest node.
  In the background, it will check for the consistency in the other replicas node. If it finds that the data is inconsistent, the coordinate node will initiate a read repair request in the background.



- **Quoram:** if the quoram is 2, the coordinator will send a read request to the fastest node for actual data and will send a digest request to one of the replica nodes for hash of data.

  The client initiates a request with the coordinator node, which consults the snitches program to get information about the fastest and the second fastest node, and then sends the read request to the fastest and a request to the second fastest. Once it gets the data and hash of data back, the coordinator node compares these to get us to check for the consistency.

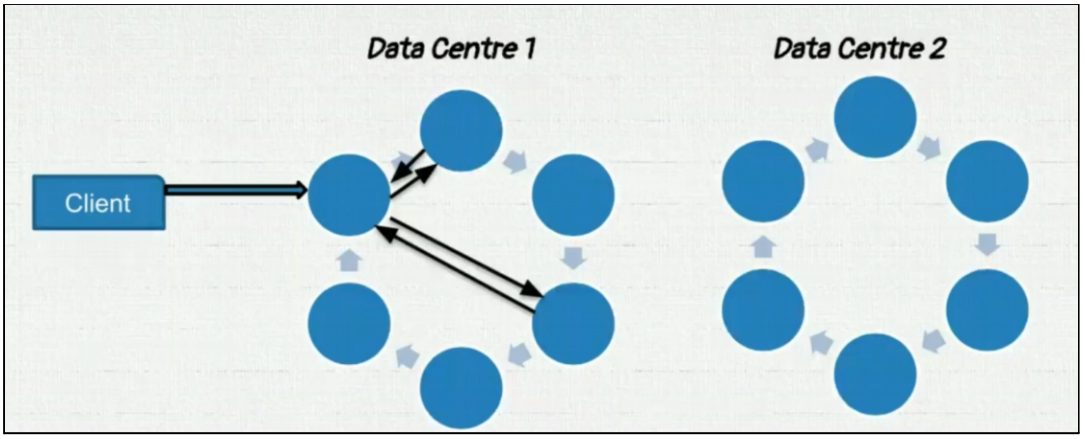
  - If they match, then the coordinator node is good to send back the requested data back to the client.
  - If they don't match, then the coordinator node sends a read request to the second fastest node to get the actual data. Once it gets the data, merges these two data based on the latest timestamp of the column and then sends back the most data back to the client. Also the coordinator node initiates a read repair request in the background to come up with these inconsistencies.



- **Local_Quoram:** if local_quoram is 2, the coordinator will send a read request to 2 replicas in the local datacenter, not communicating with another data center.

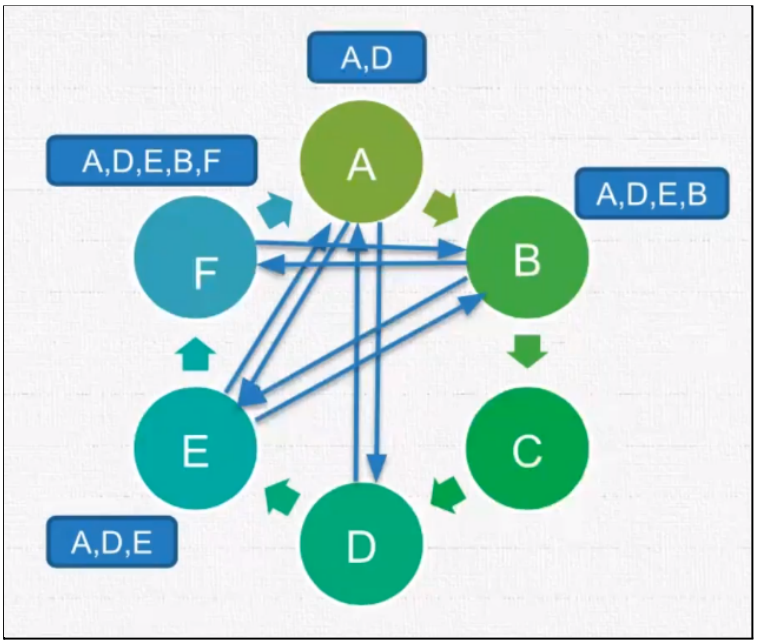
  In the background the coordinator node checks for the inconsistency across their data center as well and if there are inconsistencies, it will initialise a read repair request.

  Each_quoram is not supported for read consistency.



Gossip Protocol

Nodes gossip with each other every second to share state info about each other.



The information flows very fast and very efficiently around the cluster and within minutes the cluster is up to date with the state information.

If a node doesn't respond back the starting node assigns a suspicious level to that node. Suspicious level indicates the probability of being dead or alive. This mechanism is called: **accrual failure detection model**.

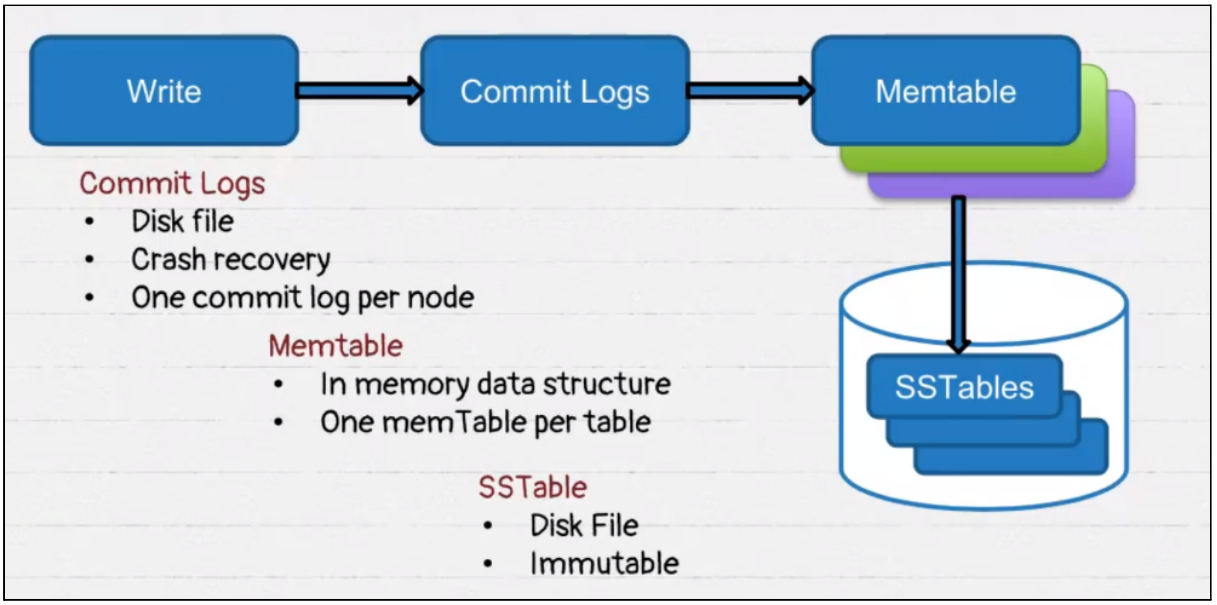
Gossiper class: Gossiper initiates a gossip session with any random node in the cluster to maintain a list of state information of nodes locally.

How write happens?

When a write operation comes it is written straightaway to commit logs, which is a disk file that is used for a crash recovery purpose, meaning if the system crashes, the committee can ensure that data is not lost and there is only one commit log per node.

After the commit log the data is written to memtable that is an in-memory data structure, each table has one. As we know the memory is limited so we cannot keep all the data in memory, so when the memTable reach some threshold value, then the content of these tables are flush to the SSTables which is again a disk file and then a new memTable is created for that table that was flushed.

These SSTables are immutable like files. One important point to note here is that every operation either delete or update is a write operation.



Cassandra Storage

Bloom Filter:

- In-memory data structure.
- Each SSTable has a bloom filter associated.
- It tells whether the partition key is present or not.
- It can give false positives.

Row Cache:

- In-memory data structure.
- Subsets of rows in memory
- Configurable - All, None, N

Key Cache:

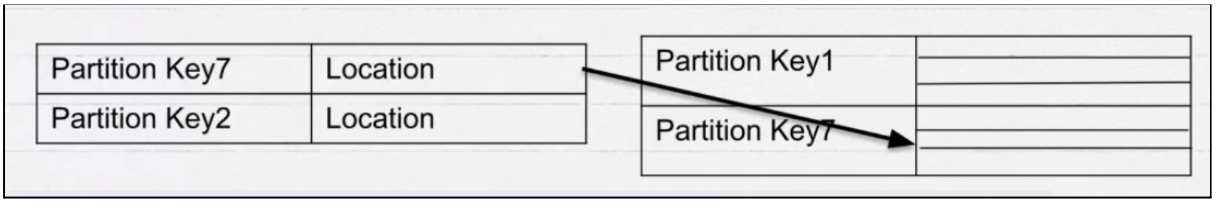
- In-memory data structure.
- Key-Value pair
- Key: Partition key
- Value: location of data on disk

Data File:

- Disk data structure.
- Actual Data is stored.
- Partitions and rows.

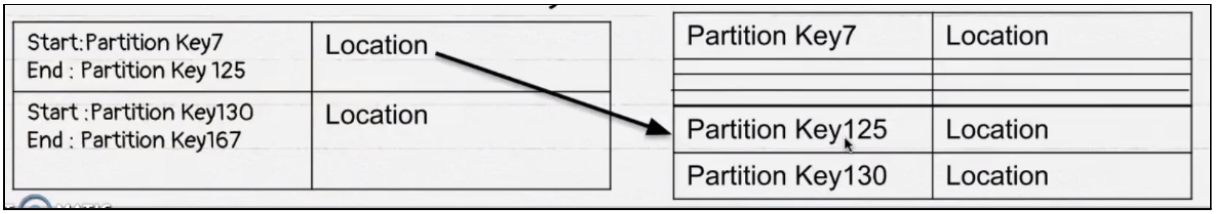
Partition Index File:

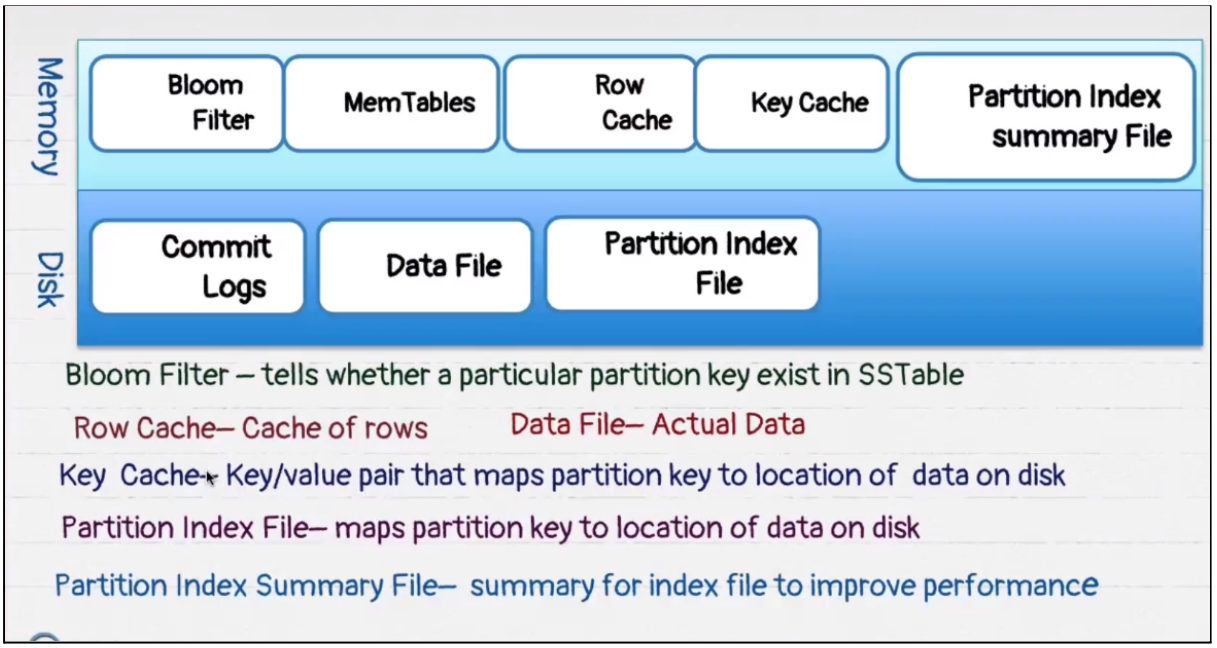
- Location for row in data file for a partition key.



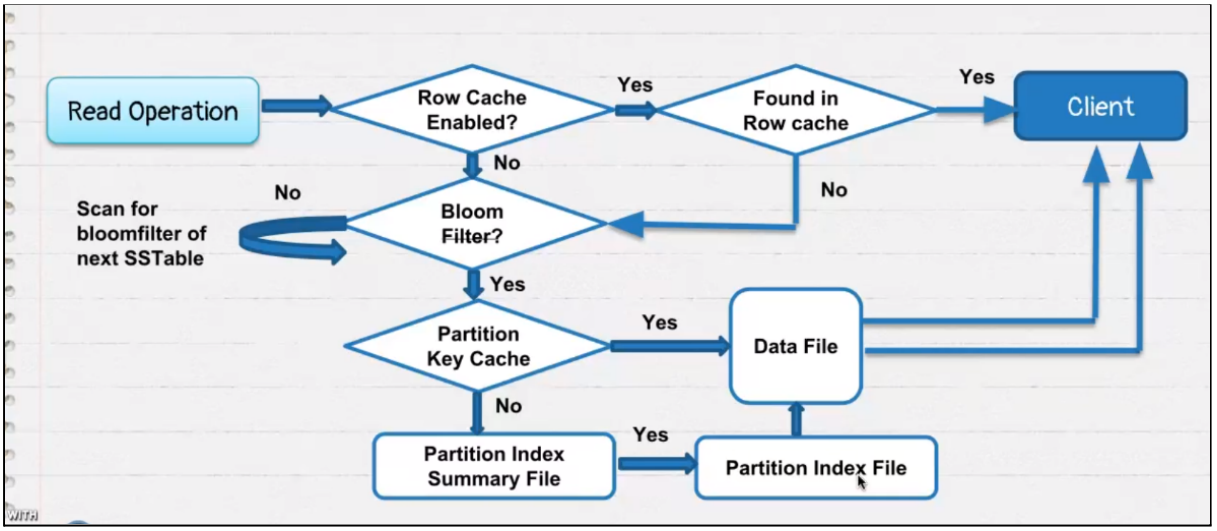
Partition Index summary File:

- In-memory data structure
- Summary for index file

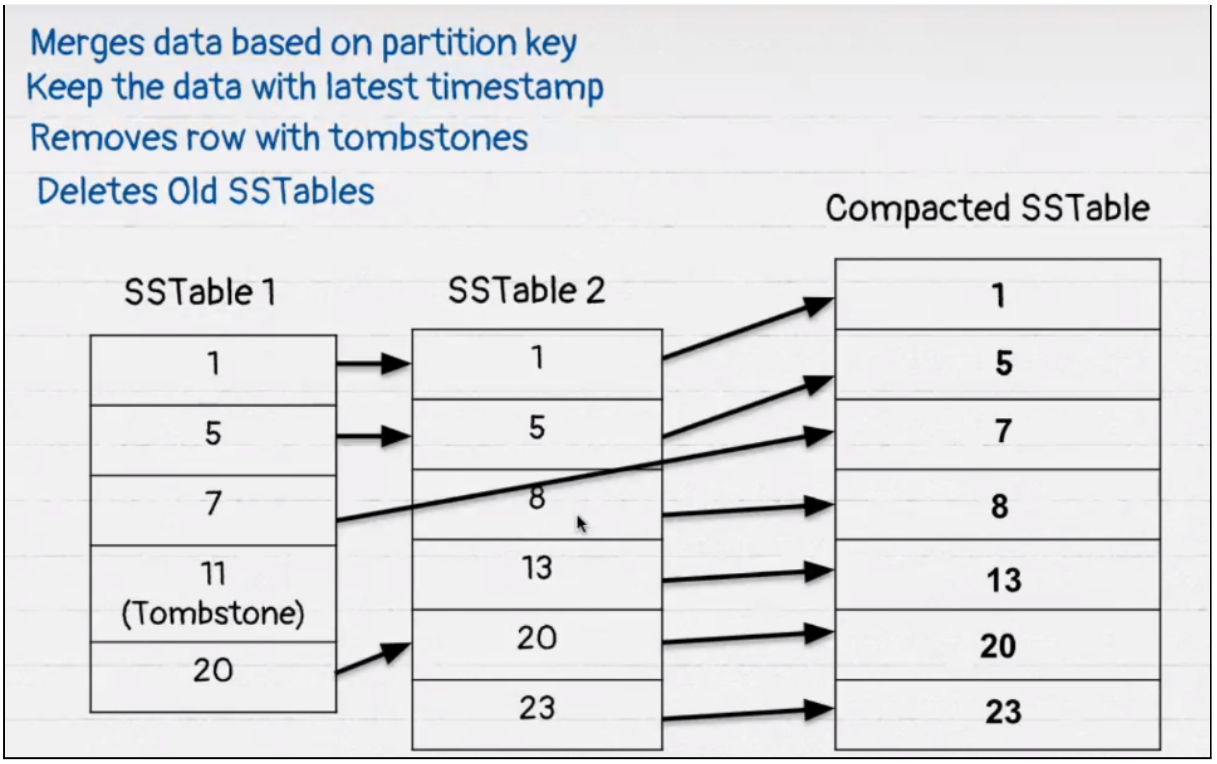




How Read happens?



Compaction



Cassandra Configuration & Managing Cluster

1. Configure JAVA_HOME and PYTHON_HOME as environment variables.
2. Download Cassandra binaries. Download
3. Set CASSANDRA_HOME as environment variable
4. Create data/cassandraLogs dir in the cassandra downloaded.
5. Edit configuration, modifying the logback.xml ${cassandra.logdir} for the cassandraLogs dirpath
6. (Optional-troubleshooting) lib/sigar-amd64-winnt rename
7. Run cassandra.bat

Cassandra Configuration Files

- **cassandra.yml**: main configuration file to maintain the cassandra cluster
- **cassandra-rackdc:** indicates the rack and datacenter. We can use this for gossip protocol.
- **cassandra-topology:** indicates the ip-address, datacenter and rack information.

Node Tool Utility

nodetool help: command list.

nodetool describe cluster: give information about the cluster such as name, snitch, partitioner, schema version.

nodetool status: give information about the status of the nodes running on our cluster.

nodetool -h *hostname* info: information about a particular node

nodetool gossipinfo: information about the gossip protocol on course

nodetool version: give the version of cassandra artifact

nodetool disablegossip: disable the gossip protocol

nodetool invalidatekeycache: invalidate the key cache in-memory

nodetool tablestats *keyspacename.tablename*: give information about a particular table.

Cassandra with Java

Dependencies:

```
<dependency>
  <groupId>com.datastax.oss</groupId>
  <artifactId>java-driver-core</artifactId>
  <version>4.13.0</version>
</dependency>
<dependency>
  <groupId>com.datastax.cassandra</groupId>
  <artifactId>cassandra-driver-core</artifactId>
  <version>3.0.0</version>
</dependency>
<dependency>
```

Best Practices & Anti Patterns

Cassandra Data Modelling

- Analyze & understand the data
- develop well organized and performant Cassandra Cluster

Key Goals:

- **Query first design:** access patterns in Advance.
- **No Joins:** duplication & denormalization.
- **Uniform Distribution:** no wide partitions & uniform distribution

Cassandra Anti-Patterns

- **Queue Like Design:** consume & delete data.
- **Too many deletes/updates:** tombstones/dead records.
- **Dynamic Schema:** changing tables schema frequently.
- **Collections for large objects:** Max map and list is 2GB. Recommended ~ MB's
- **Wide Partitions:** Thumb Rule ~ 100 MB
- **Columns-Blob/Text:** recommended size < 1MB
- **Select Without Partition key:** full table scan
- **Too many tables:** table count < 200
- **SimpleStrategy in production:** NetworkTopologyStrategy preferrable

Cassandra on AWS

The Amazon cloud has given a server service that is Amazon keyspace.