

4 types of NoSQL databases

- ① Key-value
- ② Column database
- ③ Document database
- ④ Graph database

M	T	W	T	F	S	S
Page No.:						YOUVA
Date:						

Cassandra

- NoSQL database
- massively scalable.
- Peer-to-peer architecture
- to handle big data workloads across multiple data centres with no single point of failure.

Features ^{no. of nodes ↑}

- ① Scalability
- ② Flexibility ^{strong, weak, semi-strong}
- ③ Transaction Support ^(ACID properties)
- ④ Highly Available & Fault Tolerant ^(Data replication)
- ⑤ Open source

Node

Cluster - data evenly distributed around nodes in a cluster
Datacenter

Partition - Partition key defined for each table

Gossip
Snitch

Replication Factor

Consistency

Partitioning

① Random → default

- Partition data evenly as possible across all nodes using a hash of every column family row key

② Ordered → stores column family row keys in sorted order across the nodes in DB cluster.

Data Distribution & Partitioning

- Each node \rightarrow owns a set of tokens
token range manually configured or
randomly assigned.
- Partition key (PK) for each table.
hash fn to convert PK to token.
 \rightarrow This determines which nodes store
that piece of data.

By default, MySQL 3 Partitioner

Hash Algo

- fn used to map data of arbitrary size to
data of fixed size
- slight difference in i/p \rightarrow large differences in
o/p

MySQL 3 Partitioner use

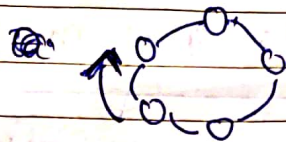
MySQL Hash fn



creates 64-bit hash value

of row key

range: -2^{63} to $+2^{63}$

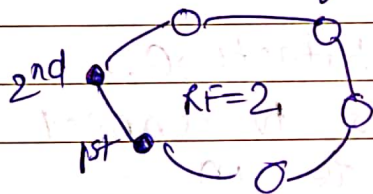
-  Each node \rightarrow configured with an initial token

Replication

- Replication factor \Rightarrow 1 : One copy of a row
 \Rightarrow 2 : two copies } Stored in clusters.

Controlled at Keyspace level in Cassandra

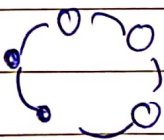
① Simple Topology



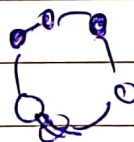
{ 'class': 'SimpleStrategy',
 'replication_factor' : 2
 } ;

② Network Topology - Multiple DCs

DC1
RF=2



DC2
RF=3



CREATE Keyspace Test with

replication = { 'class': 'NTS', 'DC1': 2, 'DC2': 3 } ;

How is location of each node (rack, dc) known?

Switch
→ determines which DC & racks are ^{(1) written to} ^{(2) read from}

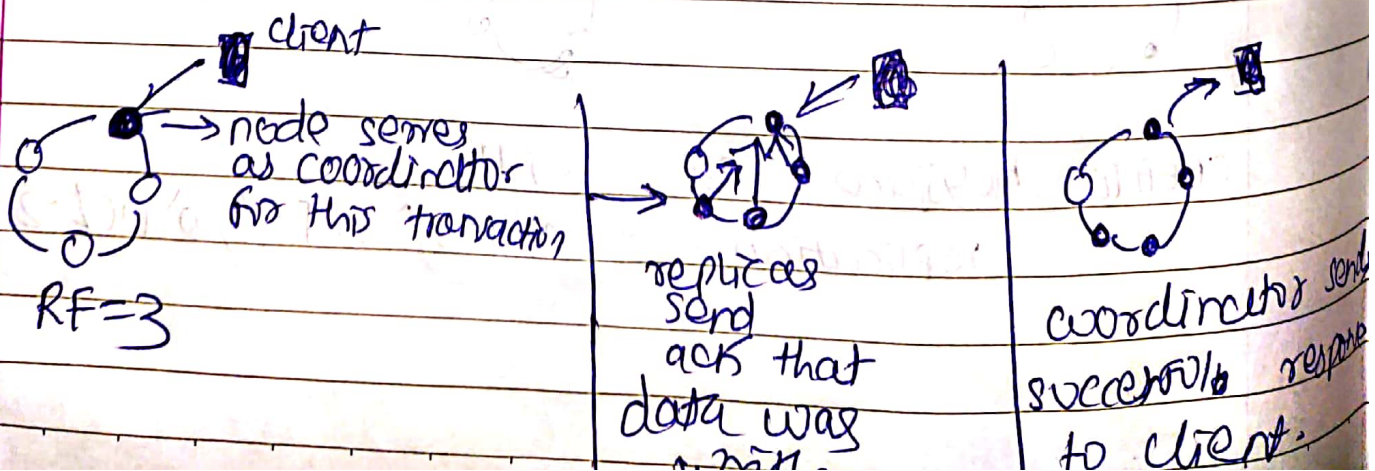
Switches inform Class about Network topology so that requests are routed efficiently

- does its best not to have more than one replica on same rack

Reading & Writing to Nodes

- 'Location Independent' architecture.
which allows any user to connect to any node in any data center & read/write the data they need.
- All writes being partitioned & replicated for ^{uses} them automatically throughout the cluster.

Writing Data



What if node is down?



only two nodes respond write
 client choose \rightarrow success or not
 WC (Write Consistency)

Two nodes down?

- Write Consistency = QUORUM
 - (NO \rightarrow success) write
 - Failed to write majority replicas.
- returns must record data from majority of replicas.

Tunable Data Consistency

- One to all responding
- Handles multi-data center operations

ANY \rightarrow from any replica

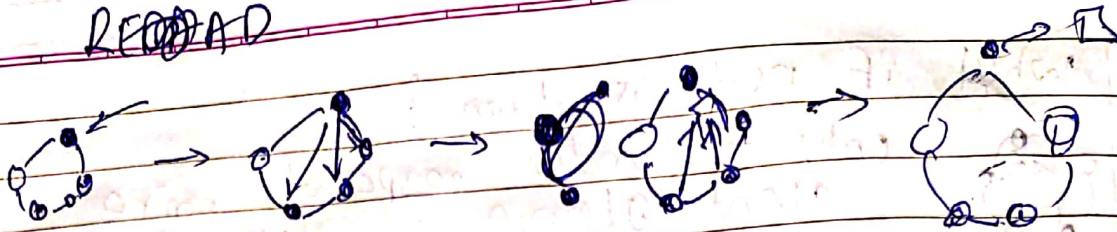
QUORUM

LOCAL QUORUM

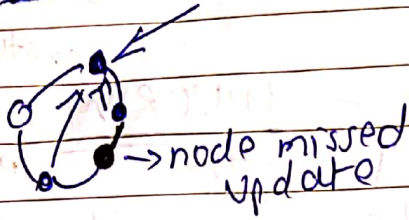
ALL

\rightarrow from all replicas
 most recent data

~~READ~~ AD



Nodes disagree to write



- write was successful

~~READ~~ AD

node back online has older data than other replicas

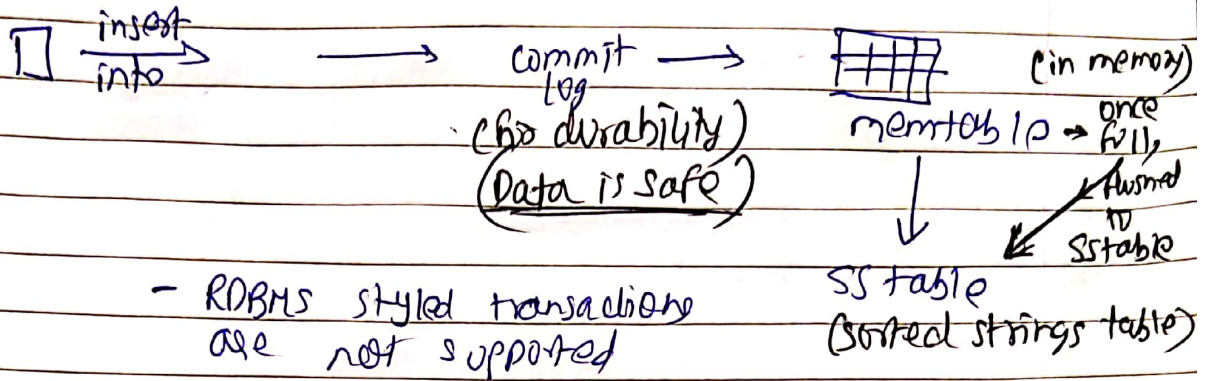
READ repair

- coordinator notifies "out-of-date" node that it has old data.
- out of date node receives updated data from another replica.

WRITES

- atomic at row level

→ all columns are written (or) updated or none are



Cassandra Fastest DB for write

READS

- Data is read from memtable in memory
- Multiple SSTables may also be read.

Bloom Filters prevent excessive reading of SSTables

