Problem Statement:

Explain the below concepts with an example in brief.

● NOSQL Databases

- NOSQL databases are designed to expand transparently to take advantage of new nodes, and they're usually designed with low-cost commodity hardware in mind.

- The challenge of continuously growing data volumes can be tackled by NOSQL databases.

- NOSQL databases are generally designed from the ground up to require less management. Automatic repair, data distribution, and simpler data models lead to lower administration and tuning requirements.

- NOSQL databases typically use clusters of cheap commodity servers to manage the exploding data and transaction volumes. The result is that the cost per gigabyte or transaction/second for NOSQL can be many times less than the cost for RDBMS, allowing you to store and process more data at a much lower price point.

- NOSQL introduces flexibility in terms of changing data models. NOSQL Key Value stores and document databases allow the application to store virtually any structure it wants in a data element.

- NOSQL has below features:
  - Schema-less
  - Open Source
  - Running well on huge clusters
  - Normalization not required
  - Not using the relational model
  - Cheaper as compared to RDBMS

● Types of NOSQL Databases

PFB the type of NOSQL databases:

1. Key value databases

2. Graph databases

3. Document Databases

4. Column Family Stores

KEY VALUE DATABASES:

In these types of databases, the client can either get the value for the key, put a value for a key, or delete a key from the data store. The value is a blob which can store any type of value without enforcing strict type checking.

Since key-value stores always use primary-key access, they generally have great performance and can be easily scaled.

For e.g. COUCHBASE, RIAK, REDIS etc

GRAPH DATABASES:

Graph databases allow you to store entities and relationships between these entities. Entities are also known as nodes, which have properties.

Nodes are like object instances and Relations are known as edges that can have properties.

Edges have directional significance; nodes are organized by relationships which allow you to find interesting patterns between the nodes.

The organization of the graph lets the data to be stored once and then interpreted ins different ways based on relationships.

Relationships are the most prominent part of Graph databases and most of the value of graph databases is derived from the relationships.

For e.g. FLOCKDB, ORIENTDB etc.

DOCUMENT DATABASES:

In these types of databases, client can store and retrieve documents which can be XML, JSON, BSON etc.

These documents are self-describing, hierarchical tree data structures which can consist of maps, collections, and scalar values. The documents stored are similar to each other

but do not have to be exactly the same. Document databases store documents in the value part of the key-value store.

For e.g. MONGODB, COUCHDB, TERRASTORE, ORIENTDB, RAVENDB etc.

COLUMN FAMILY STORES:

Column-family databases store data in column families as rows that have many columns associated with a row key. Column families are groups of related data that is often accessed together.

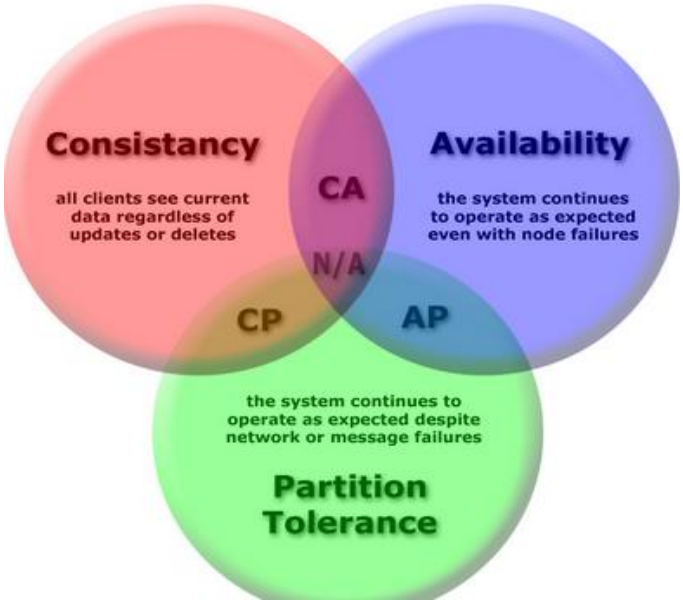
Each column family can be compared to a container of rows in an RDBMS table where the key identifies the row and the row consists of multiple columns. The difference is that various rows do not have to have the same columns, and columns can be added to any row at any time without having to add it to other rows.

When a column consists of a map of columns, then we have a super column. A super column consists of a name and a value which is a map of columns. Think of a super column as a container of columns.

For e.g. CASSANDRA, HBASE, HYPERTABLE, AMAZON DYNAMODB etc.

● CAP Theorem:

CAP theorem can be illustrated with below diagram:



CONSISTENCY:

A read will definitely return the most recently written data to a client.

AVAILABILITY:

A non-failing node will return a reasonable response without any timeout or error.

PARTITION TOLERANCE:

It implies system will continue to function when network partition occurs.

CAP THEOREM:

In a distributed system, managing consistency(C), availability(A) and partition tolerance (P) is important, Eric Brewer put forth the CAP theorem which states that in any distributed system we can choose only two of consistency, availability or partition tolerance.

Availability can be achieved in databases by replicating data on different nodes in a cluster.

Consistency can be achieved by updating several nodes before allowing further reads.

No distributed system is safe from network failures, thus network partitioning generally has to be tolerated. In the presence of a partition, one is then left with two options: consistency or availability.

When choosing consistency over availability, the system will return an error or a time-out if particular information cannot be guaranteed to be up to date due to network partitioning.

When choosing availability over consistency, the system will always process the query and try to return the most recent available version of the information, even if it cannot guarantee it is up to date due to network partitioning.

In the absence of network failure – that is, when the distributed system is running normally – both availability and consistency can be satisfied.

● HBase Architecture

- HBASE Tables are split into REGIONS and are managed by REGION SERVERS.
- HBASE Architecture consists of MASTER PROCESS, REGION SERVERS & CLIENT LIBRARY.

- **REGIONS**:
  - They are vertically divided by column families into STORES and STORES are saved as files in HDFS.
  - Tables are horizontally divided into row key range into REGIONS.
  - A region contains all rows in the table between the region's start key and end key.
  - Regions are assigned to the nodes in the clusters which are called as REGION SERVERS.

- **MASTER PROCESS:**
  - It assigns REGIONS to REGION SERVERS with the help of ZOOKEEPER.
  - It handles load balancing of the regions across the region servers.
  - It unloads busy servers and shifts the regions to less occupied servers.
  - It maintains the state of the cluster with load balancing act.
  - It is liable for schema alterations and other metadata operations.

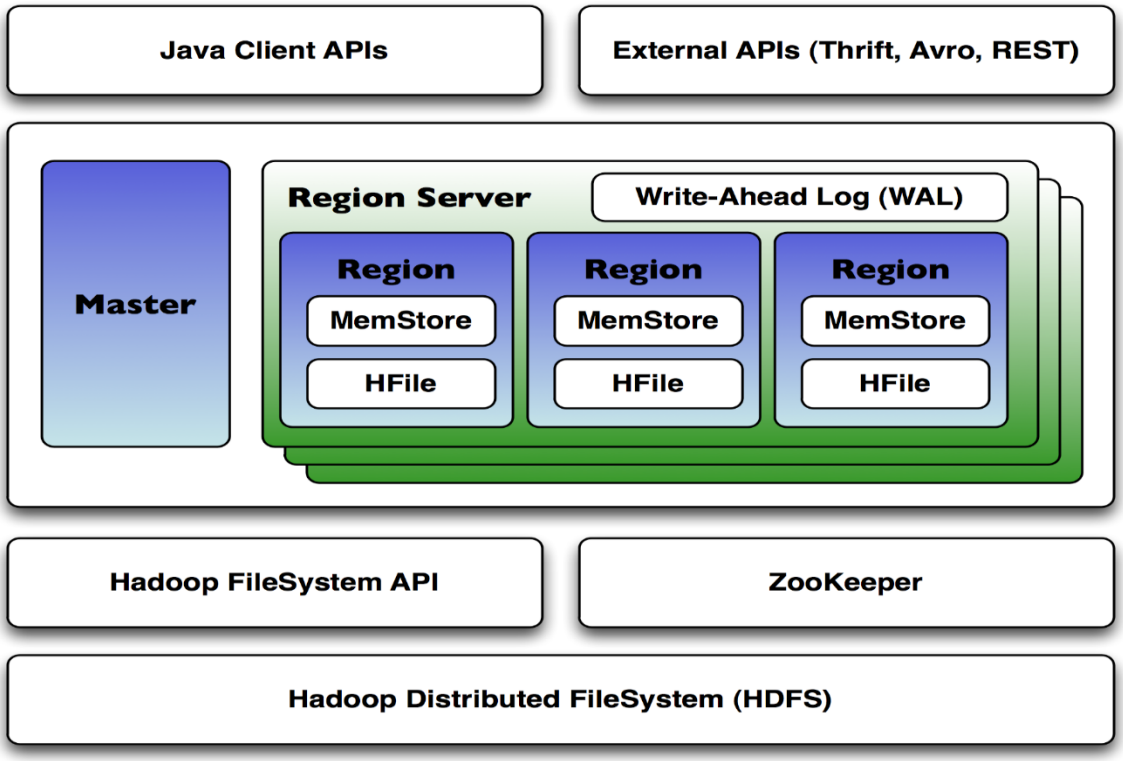
- **REGION SERVER:**
  - It communicates with the client and serves requests for read and write operations.
  - It handles all read and write operations for the all regions coming under its supervision.
  - It decides the region size by following the region size thresholds.

- A region server can serve up to 1000 regions.
- It gets hosted on DATA NODES where data is processed.
- REGION SERVER contains REGIONS & STORES.
- STORES contain MEMSTORE & HFILES where MEMSTORE is similar to CACHE MEMORY.
- Initially data is stored into MEMSTORE and later gets transferred to HFILES in form of blocks for write operation.

❖ **ZOOKEEPER: THE DISTRIBUTED SERVICE COORDINATOR**

- It is used as a coordinator to maintain server state of the cluster.
- It monitors the state of servers whether they are available and alive or not.
- It provides server failure notifications.
- It is an open source project that provides services like configuration information, naming, providing distributed synchronization.
- It has ephemeral nodes representing different region servers. Master servers use these nodes to discover available serves.
- Ephemeral nodes are used to track server failures and network partitions.
- Clients communicate with REGION SERVES via ZOOKEEPER.
- In pseudo and standalone modes, HBase itself will take care of ZOOKEEPER.
- ZOOPKEEPER stores the location of METATABLE, which is a HBASE CATALOG table to store the locations of regions in cluster.

PFB the HBASE ARCHITECTURE diagram in brief.



● HBase vs RDBMS

- HBASE is schema-less and does not have any fixed columns. HBASE defines only column family.
  On the contrary, RDBMS has fixed schema.

- HBASE is horizontally scalable.
  On the contrary, RDBMS is hardly scalable.

- HBASE is built for wide tables.
  On the contrary, RDBMS is built for thin and small tables.

- HBASE defines column families in tables.
  On the contrary, RDBMS defines concrete structure of tables.

- HBASE does not implement normalization.
  On the contrary, RDBMS enforces normalization.

- HBASE does not perform type checking on table values.
  On the contrary, RBMS enforces strict type checking on table values during insert operation.

- HBASE is a non-relational database modelled after Google's BIGTABLE.
  On the other hand, RDBMS is a relational database and tables may have relation with each other.