HBase Assignment 3 :

1) NoSQL data bases

Some of the NoSQL databases are given below.

i) MongoDB

MongoDB is a document store, and the current top NoSQL database engine in use today. As is the requirement for NoSQL engines, MongoDB does not use a relational schema, instead using JSON-like "documents" to store data. The document is akin to a record, housing fields and values. MongoDB supports dynamic schemas, and is free and open source software.

Fields and values in JSON

MongoDB also provides these features that you would expect: load balancing, replication, indexing, querying, and can act as a file system (with load balancing and fault tolerance). For those interested in a more detailed treatment of a particular point, check out this discussion of JSON and BSON (Binary JSON) objects, directly from MongoDB.

ii). Cassandra

Originally developed at Facebook, Cassandra is a decentralized, distributed, column-oriented database engine. It is optimized for clusters, especially those across multiple datacenters, and thanks to its asynchronous updating and master-less design, Cassandra provides low latency client access. Like MongoDB, it is also free and open source.

Cassandra is a column-oriented database, meaning that its rows actually contain what we most usually think of as vertical data, or what is traditionally held in relational columns (Rows contains columns? Huh?). The advantage of column-oriented database design is that some types of data lookups can become very fast, given that the desired data could be stored consecutively in a single row (compare this with having to search and read from multiple, nonconsecutive rows to attain the same field value in row-oriented database). This particularity, along with the optimized, decentralized distributed model has solidified Cassandra's popularity over the years.

iii) Redis

Redis is the most popular and widely-used key-value store implementation on our list. What is a key-value store? Key-value stores are simple paradigms at a high-

level: assign values to keys to facilitate the access and storage of these values, which are always found via their keys. Think hash maps and you've got the idea (dictionaries in Python).

Key value

Redis holds its key-value pairings in memory, making their access quick. If data durability can be sacrificed (mainly with non-critical data, or in read-only or -primarily situations), being able to forego data writes means that this memory-only data boasts incredibly fast performance. Over the years, APIs have been developed for an incredibly wide variety of languages as well, making Redis an easy choice for developers.

iv) HBase

Another column-oriented database, HBase is a free and open source implementation of Google's BigTable. While HBase is a legitimate piece of software in its own right, some of its popularity and widespread use undoubtedly comes from its close association with Hadoop, as it is part of the Apache project. It facilitates the efficient lookup of sparse, distributed data, which is one of its strongest selling points.

HBase has a number of high profile implementations in the wild, including those at LinkedIn, Facebook, and Spotify. Numerous related Apache projects also support HBase, notably providing an SQL layer for data access (Phoenix), which would certainly bode well for relational database admins looking to implement a NoSQL solution. And with the high numbers of Hadoop installations in existence, and growing, HBase will be a default NoSQL storage solution for many for years to come.

v) Neo4j

Neo4j is the lone graph database management system on our list, the most popular such system i use today. The graph database is premised on edges acting as relationships, directly relating data instances to one another. Like others on the list, Neo4j has an open source implementation as well.

Built in Java, Neo4j data can be accessed and updated via the Cypher Query Language, a graph-specific SQL-like language.

Graph databases (and Neo4j, of course) have advantages in some use cases, including potentially in certain data mining and pattern recognition scenarios, given that associations between data instances are explicitly stated.

2) Types of NoSQL data bases?

There are four general types of NoSQL databases, each with their own specific attributes:

- **Graph database** – Based on graph theory, these databases are designed for data whose relations are well represented as a graph and has elements which are interconnected, with an undetermined number of relations between them. Examples include: Neo4j and Titan.
- **Key-Value store** – we start with this type of database because these are some of the least complex NoSQL options. These databases are designed for storing data in a schema-less way. In a key-value store, all of the data within consists of an indexed key and a value, hence the name. Examples of this type of database include:Cassandra, DyanmoDB, Azure Table Storage (ATS), Riak, BerkeleyDB.
- **Column store** – (also known as wide-column stores) instead of storing data in rows, these databases are designed for storing data tables as sections of columns of data, rather than as rows of data. While this simple description sounds like the inverse of a standard database, wide-column stores offer very high performance and a highly scalable architecture. Examples include: HBase, BigTable and HyperTable.
- **Document database** – expands on the basic idea of key-value stores where "documents" contain more complex in that they contain data and each document is assigned a unique key, which is used to retrieve the document. These are designed for storing, retrieving, and managing document-oriented information, also known as semi-structured data. Examples include: MongoDB and CouchDB.

The following table lays out some of the key attributes that should be considered when evaluating NoSQL databases.

Datamodel	**Performance**	**Scalability**	**Flexibility**	**Complexity**	**Functionality**
Key-value store	High	High	High	None	Variable (None)
Column Store	High	High	Moderate	Low	Minimal

Document Store	High	Variable (High)	High	Low	Variable (Low)
Graph Database	Variable	Variable	High	High	Graph Theory

3) CAP Theorem

In a distributed system, managing consistency(C), availability(A) and partition toleration(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. Many NoSQL databases try to provide options where the developer has choices where they can tune the database as per their needs. For example if you consider Riak a distributed key-value database.

There are essentially three variables r, w, n where

r=number of nodes that should respond to a read request before its considered successful.

w=number of nodes that should respond to a write request before its considered successful.

n=number of nodes where the data is replicated aka replication factor.

In a Riak cluster with 5 nodes, we can tweak the r,w,n values to make the system very consistent by setting r=5 and w=5 but now we have made the cluster susceptible to network partitions since any write will not be considered successful when any node is not responding. We can make the same cluster highly available for writes or reads by setting r=1 and w=1 but now consistency can be compromised since some nodes may not have the latest copy of the data.

The CAP theorem states that if you get a network partition, you have to trade off availability of data versus consistency of data. Durability can also be traded off against latency, particularly if you want to survive failures with replicated data.

NoSQL databases provide developers lot of options to choose from and fine tune the system to their specific requirements. Understanding the requirements of how the data is going to be consumed by the system, questions such as is it read heavy vs write heavy, is there a need to query data with random query parameters, will the system be able handle inconsistent data.

Understanding these requirements becomes much more important, for long we have been used to the default of RDBMS which comes with a standard set of features no matter which product is chosen and there is no possibility of choosing some features over other.

The availability of choice in NoSQL databases, is both good and bad at the same time. Good because now we have choice to design the system according to the requirements. Bad because now you have a choice and we have to make a good choice based on requirements and there is a chance where the same database product may be used properly or not used properly.

An example of feature provided by default in RDBMS is transactions, our development methods are so used to this feature that we have stopped thinking about what would happen when the database does not provide transactions.

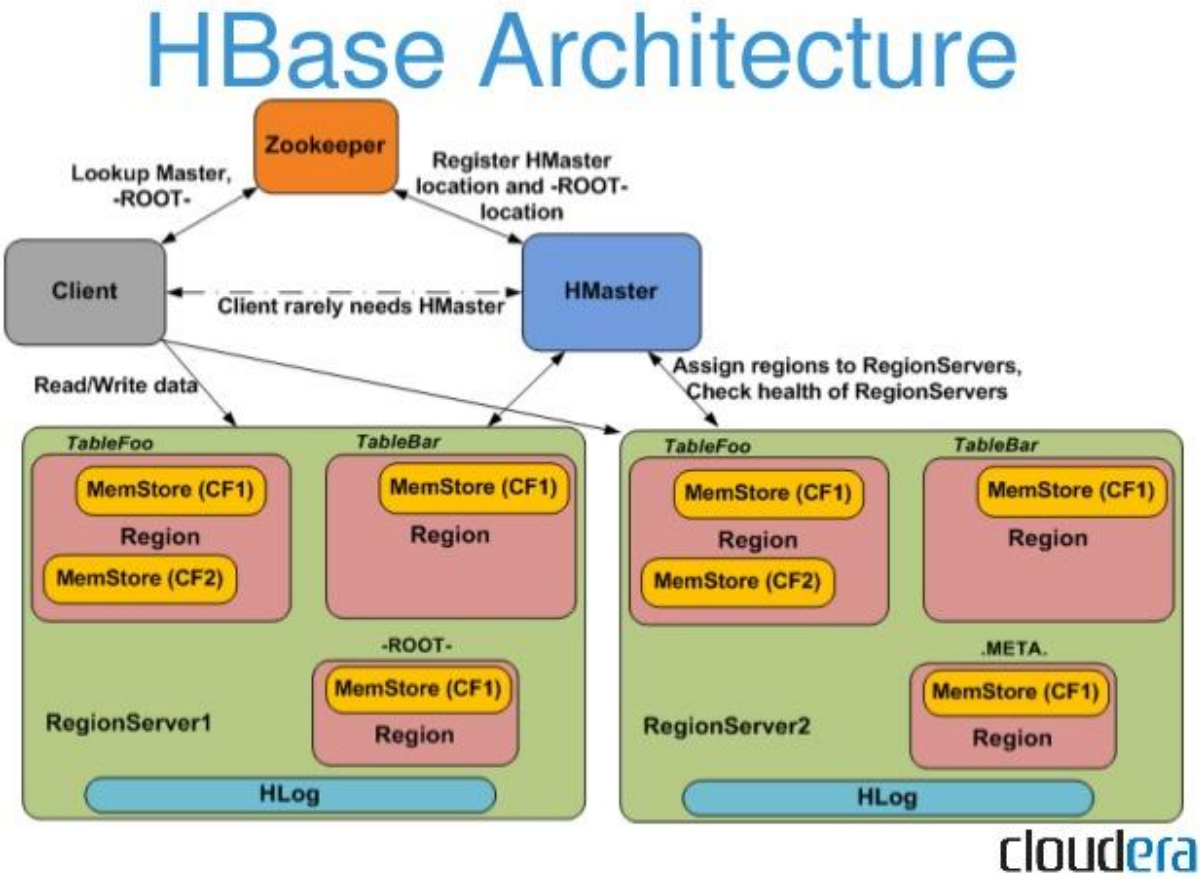
Most NoSQL databases do not provide transaction support by default, which means the developers have to think how to implement transactions, does every write have to have the safety of transactions or can the write be segregated into "critical that they succeed" and "its okay if I lose this write" categories. Sometimes deploying external transaction managers like ZooKeeper can also be a possibility.

4) HBase Architecture

HBase provides low-latency random reads and writes on top of HDFS. In HBase, tables are dynamically distributed by the system whenever they become too large to handle (Auto Sharding). The simplest and foundational unit of horizontal scalability in HBase is a Region. A continuous, sorted set of rows that are stored together is referred to as a region (subset of table data).

HBase architecture has a single HBase master node (HMaster) and several slaves i.e. region servers. Each region server (slave) serves a set of regions, and a region can be served only by a single region server. Whenever a client sends a write request, HMaster receives the request and forwards it to the corresponding region server.

HBase Architecture, Data Flow, and Use cases



HBase can be run in a multiple master setup, wherein there is only single active master at a time. HBase tables are partitioned into multiple regions with every region storing multiple table's rows.

Components of Apache HBase Architecture

HBase architecture has 3 important components- HMaster, Region Server and ZooKeeper.

HMaster

HBase HMaster is a lightweight process that assigns regions to region servers in the Hadoop cluster for load balancing. Responsibilities of HMaster –

i) Manages and Monitors the Hadoop Cluster

ii) Performs Administration (Interface for creating, updating and deleting tables.)

iii) Controlling the failover

iv) DDL operations are handled by the HMaster

v) Whenever a client wants to change the schema and change any of the metadata operations, HMaster is responsible for all these operations.

Region Server

These are the worker nodes which handle read, write, update, and delete requests from clients. Region Server process, runs on every node in the hadoop cluster. Region Server runs on HDFS DataNode and consists of the following components –

Block Cache – This is the read cache. Most frequently read data is stored in the read cache and whenever the block cache is full, recently used data is evicted.

MemStore- This is the write cache and stores new data that is not yet written to the disk. Every column family in a region has a MemStore.

Write Ahead Log (WAL) is a file that stores new data that is not persisted to permanent storage.

HFile is the actual storage file that stores the rows as sorted key values on a disk.

Zookeeper

HBase uses ZooKeeper as a distributed coordination service for region assignments and to recover any region server crashes by loading them onto other region servers that are functioning.

ZooKeeper is a centralized monitoring server that maintains configuration information and provides distributed synchronization. Whenever a client wants to communicate with regions, they have to approach Zookeeper first.

HMaster and Region servers are registered with ZooKeeper service, client needs to access ZooKeeper quorum in order to connect with region servers and HMaster. In case of node failure within an HBase cluster, ZKquoram will trigger error messages and start repairing failed nodes.

ZooKeeper service keeps track of all the region servers that are there in an HBase cluster- tracking information about how many region servers are there and which region servers are holding which DataNode. HMaster contacts ZooKeeper to get the details of region servers. Various services that Zookeeper provides include –

i) Establishing client communication with region servers.

ii) Tracking server failure and network partitions.

iii) Maintain Configuration Information

iv) Provides ephemeral nodes, which represent different region servers.

Understanding the fundamental of HBase architecture is easy but running HBase on top of HDFS in production is challenging when it comes to monitoring compactions, row key designs manual splitting, etc

5) H Base Vs RDBMS:-

H BASE and other column-oriented DATABASE are often compared to more traditional and popular relational database or RDBMS.

H Base	RDBMS
1. Column-oriented	1. Row-oriented(mostly)
2. Flexible schema, add columns on the Fly	2. Fixed schema
3. Good with sparse tables.	3. Not optimized for sparse tables.
4. No query language	4. SQL
5. Wide tables	5. Narrow tables
6. Joins using MR – not optimized	6. optimized for Joins(small, fast ones)
7. Tight – Integration with MR	7. Not really
8. De-normalize your data.	8. Normalize as you can
9. Horizontal scalability-just add hard war.	9. Hard to share and scale.
10. Consistent	10. Consistent
11. No transactions.	11. transactional
12. Good for semi-structured data as well as structured data.	12. Good for structured data.