**HBASE ASSIGNMENT 3.9**

Explain the below concept with an example in brief.

1. NOSQL databases.

NoSQL is an approach to database design that can accommodate a wide variety of data models, including key-value, document, columnar and graph formats. NoSQL, which stand for "not only SQL is an alternative to traditional relational databases in which data is placed in tables and data Schema is carefully designed before the database is built. NoSQL databases are especially useful for working with large sets of distributed data. They are of following type.

**Key value pair:**

Key-value stores, or key-value databases, implement a simple data model that pairs a unique key with an associated value. Because this model is simple, it can lead to the development of key-value databases, which are extremely performant and highly scalable for session management and caching in web applications. Implementations differ in the way they are oriented to work with RAM, solid-state drives or disk drives. Examples include Aerospike, Berkeley DB, MemchacheDB, Redis

**Document Databases:**

Document databases, also called document stores, store semi-structured data and descriptions of that data in document format. They allow developers to create and update programs without needing to reference master schema. Use of document databases has increased along with use of JavaScript and the JSON a data interchange format that has gained wide currency among web application developers, although XML and other data formats can be used as well.  Document databases are used for content management and mobile application data handling. Couchbase Server,CouchDB DocumentDB, MarkLogic and MongoDB are examples of document databases.

**Wide-Column store:**

Wide-column stores organize data tables as columns instead of as rows. Wide-column stores can be found both in SQL and NoSQL databases. Wide-column stores can query large data volumes faster than conventional relational databases. A wide-column data store can be used for recommendation engines, catalogs, fraud detection and other types of data processing.Google Big Table Cassandra and HBase are examples of wide-column stores.

**Graph Store:**

Graph data stores organize data as nodes, which are like records in a relational database, and edges, which represent connections between nodes. Because the graph system stores the relationship between nodes, it can support richer representations of data relationships. Also, unlike relational models reliant on strict schemas, the graph data model can evolve over time and use.Graph Databases are applied in systems that must map relationships, such as reservation systems or customer relationship management. Examples of graph databases include AllegroGraph, IBM Graph, Neo4j and Titan.

B.CAP Theorem:

The CAP theorem is a tool used to makes system designers aware of the trade-offs while designing networked shared-data systems. CAP has influenced the design of many distributed data systems. It made designers aware of a wide range of tradeoffs to consider while designing distributed data systems. Over the years, the CAP theorem has been a widely misunderstood tool used to categorize databases. CAP stands for consistency, Availability and Partition tolerant. We will discuss each of them .

1. Consistency:

Its guarantee that every node in a distributed cluster returns the same, most recent, successful write. Consistency refers to every client having the same view of the data. There are various types of consistency models. Consistency in CAP (used to prove the theorem) refers to linearizability or sequential consistency, a very strong form of consistency.

2. Availability:

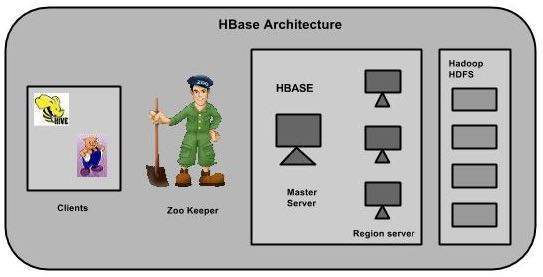
Every non-failing node returns a response for all read and writes requests in a reasonable amount of time. The key word here is every. To be available, every node on (either side of a network partition) must be able to respond in a reasonable amount of time.

3. Partition tolerant:

The system continues to function and upholds its consistency guarantees in spite of network partitions. Network partitions are a fact of life. Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.

Hbase Architecture:

In HBase, tables are split into regions and are served by the region servers. Regions are vertically divided by column families into “Stores”. Stores are saved as files in HDFS. Shown below is the architecture of HBase.



HBase has three major components: the client library, a master server, and region servers. Region servers can be added or removed as per requirement.

Master Server:

The Master Server Assigns regions to the region servers and takes the help of Apache ZooKeeper for this task. It handles load balancing of the regions across region servers. It unloads the busy servers and shifts the regions to less occupied servers .It Maintains the state of the cluster by negotiating the load balancing and Is responsible for schema changes and other metadata operations such as creation of tables and column families.

Region Server:

Regions are nothing but tables that are split up and spread across the region servers. The region servers have regions that Communicate with the client and handle data-related operations. Handle read and write requests for all the regions under it. Decide the size of the region by following the region size thresholds.

ZOOKEEPER:

Zookeeper is an open-source project that provides services like maintaining configuration information, naming, providing distributed synchronization, etc. Zookeeper has ephemeral nodes representing different region servers. Master servers use these nodes to discover available servers. In addition to availability, the nodes are also used to track server failures or network partitions. Clients communicate with region servers via zookeeper. In pseudo and standalone modes, HBase itself will take care of zookeeper.

**HBASE Vs RDBMS:**

|  |  |
| --- | --- |
| **HBase** | **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. |