**Session 9 : HBase**

**Assignment 3**

**● Explain the below concepts with an example in brief.**

**● Nosql Databases**

**● Types of Nosql Databases**

**● CAP Theorem**

**● HBase Architecture**

**● HBase vs RDBMS**

**NoSQL Databases**

NoSQL is an approach to databases that represents a shift away from traditional relational database management systems (RDBMS). To define NoSQL, it is helpful to start by describing SQL, which is a query language used by RDBMS. Relational databases rely on tables, columns, rows, or schemas to organize and retrieve data. In contrast, NoSQL databases do not rely on these structures and use more flexible data models. NoSQL can mean “not SQL” or “not only SQL.” As RDBMS have increasingly failed to meet the performance, scalability, and flexibility needs that next-generation, data-intensive applications require, NoSQL databases have been adopted by mainstream enterprises. NoSQL is particularly useful for storing unstructured data, which is growing far more rapidly than structured data and does not fit the relational schemas of RDBMS. Common types of unstructured data include: user and session data; chat, messaging, and log data; time series data such as IoT and device data; and large objects such as video and images.

BENEFITS OF NOSQL

NoSQL databases offer enterprises important advantages over traditional RDBMS, including:

• Scalability: NoSQL databases use a horizontal scale-out methodology that makes it easy to add or reduce capacity quickly and non-disruptively with commodity hardware. This eliminates the tremendous cost and complexity of manual sharding that is necessary when attempting to scale RDBMS.

• Performance: By simply adding commodity resources, enterprises can increase performance with NoSQL databases. This enables organizations to continue to deliver reliably fast user experiences with a predictable return on investment for adding resources—again, without the overhead associated with manual sharding.

• High Availability: NoSQL databases are generally designed to ensure high availability and avoid the complexity that comes with a typical RDBMS architecture that relies on primary and secondary nodes. Some “distributed” NoSQL databases use a masterless architecture that automatically distributes data equally among multiple resources so that the application remains available for both read and write operations even when one node fails.

• Global Availability: By automatically replicating data across multiple servers, data centers, or cloud resources, distributed NoSQL databases can minimize latency and ensure a consistent application experience wherever users are located. An added benefit is a significantly reduced database management burden from manual RDBMS configuration, freeing operations teams to focus on other business priorities.

• Flexible Data Modeling: NoSQL offers the ability to implement flexible and fluid data models. Application developers can leverage the data types and query options that are the most natural fit to the specific application use case rather than those that fit the database schema. The result is a simpler interaction between the application and the database and faster, more agile development.

**Types of NoSQL databases**

Several different varieties of NoSQL databases have been created to support specific needs and use cases. These fall into four main categories:

• Key-value data stores: Key-value NoSQL databases emphasize simplicity and are very useful in accelerating an application to support high-speed read and write processing of non-transactional data. Stored values can be any type of binary object (text, video, JSON document, etc.) and are accessed via a key. The application has complete control over what is stored in the value, making this the most flexible NoSQL model. Data is partitioned and replicated across a cluster to get scalability and availability. For this reason, key value stores often do not support transactions. However, they are highly effective at scaling applications that deal with high-velocity, non-transactional data. E.g. Cassandra

• Document stores: Document databases typically store self-describing JSON, XML, and BSON documents. They are similar to key-value stores, but in this case, a value is a single document that stores all data related to a specific key. Popular fields in the document can be indexed to provide fast retrieval without knowing the key. Each document can have the same or a different structure. E.g. MongoDB

• Wide-column stores: Wide-column NoSQL databases store data in tables with rows and columns similar to RDBMS, but names and formats of columns can vary from row to row across the table. Wide-column databases group columns of related data together. A query can retrieve related data in a single operation because only the columns associated with the query are retrieved. In an RDBMS, the data would be in different rows stored in different places on disk, requiring multiple disk operations for retrieval. E.g.

• Graph stores: A graph database uses graph structures to store, map, and query relationships. They provide index-free adjacency, so that adjacent elements are linked together without using an index.

Multi-modal databases leverage some combination of the four types described above and therefore can support a wider range of applications.

**CAP Theorem**

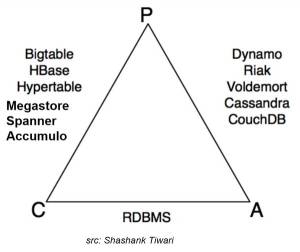
This theorem from Eric Brewer in 2000, followed up later by Lynch in 2002 state that a database can't get all these three notions at the same time:

* [consistency](https://gerardnico.com/wiki/design/consistency),
* [availability](https://gerardnico.com/wiki/counter/resource/metric/availibilty)
* [partitioning](https://gerardnico.com/wiki/database/partition_tolerance) (data is partitioned over multiple computer)

You have to choose two or sacrifice performance.

If two node of the system cannot talk to each other, can they make forward progress on their own?

* If not, you sacrifice Availability
* If so, you might have to sacrifice Consistency.

[](https://gerardnico.com/wiki/_detail/database/cap_theorem_database_type.jpg?id=database%3Acap_theorem)

Conventional databases assume no partitioning. Clusters were assumed to be small and local.

If you update a value on node A there are 2 options to propagate the change to B after a network failure:

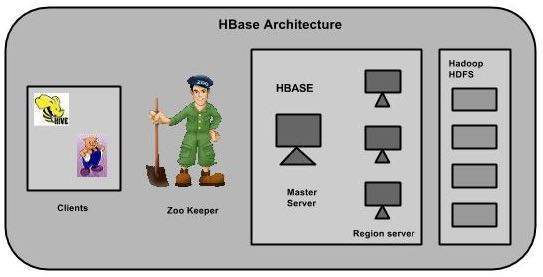
* asynchronously: The update will have to wait and B will be in a inconsistent state
* synchronously: The entire system will not be available until B receives the update and that's can take a lot of time.

The only way to obtain all notions is to run the system on a single node and this is no more a distributed system.

**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.

Note: The term ‘store’ is used for regions to explain the storage structure.



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

MasterServer

The master server -

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

Regions

Regions are nothing but tables that are split up and spread across the region servers.

Region server

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.

When we take a deeper look into the region server, it contain regions and stores as shown below:



The store contains memory store and HFiles. Memstore is just like a cache memory. Anything that is entered into the HBase is stored here initially. Later, the data is transferred and saved in Hfiles as blocks and the memstore is flushed.

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**

There differences between RDBMS and HBase are given below.

* Schema/Database in RDBMS can be compared to namespace in Hbase.
* A table in RDBMS can be compared to column family in Hbase.
* A record (after table joins) in RDBMS can be compared to a record in Hbase.
* A collection of tables in RDBMS can be compared to a table in Hbase..

