**Hadoop\_Assignment 9.3**

**Explain the below concepts with an Example in brief**

**NoSQL Databases**

NoSQL is an approach to databases that represents a shift away from traditional relational database management systems (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:

* **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 master less 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](http://basho.com/resources/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.

**Document stores**: [Document databases](http://basho.com/resources/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.

**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.

**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.

Below diagram illustrates the Examples for Each type.



**CAP Theorem**

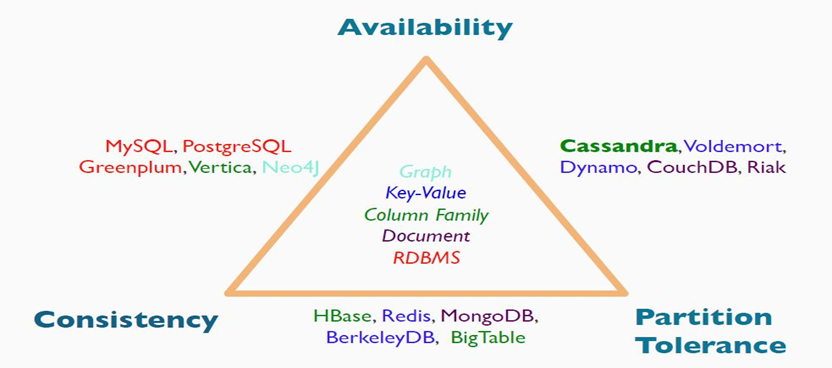
CAP describes that before choosing any Database (Including distributed database), Basing on your requirement we have to choose only two properties out of three.

**Consistency**: Whenever you read a record (or data), consistency guaranties that it will give same data how many times you read. Simply we can say that each server returns the right response to each request, thus the system will be always consistent whenever you read or write data into that.

**Availability**: Availability simply means that each request eventually receives a response (even if it’s not the latest data or consistent across the system or just a message saying the system isn’t working).

**Partition Tolerance**: Partition Tolerance means that the cluster continues to function even if there is a “partition” (communications break) between two nodes (both nodes are up, but can’t communicate).

One property should be scarified among three, so you have to choose combination of CA or CP or AP.



**Consistency – Partition Tolerance**: System will give you a consistent data and it will be distributed across the cluster. But it becomes unavailable when a node goes down.

**Availability – Partition Tolerance:** System will respond even if nodes are not communicating with each other (nodes are up and running but not communicating). But there is no guaranty that all nodes will have same data.

**Consistency – Availability**: System will give you a consistent data and you can write/read data to/from any node, but data partitioning will not be sync when develop a partition between nodes (RDBMS systems such as MySQL are of CA combination systems.)

According to CAP theorem, Cassandra will fall into category of AP combination, that means don’t think that Cassandra will not give a consistent data. We can tune Cassandra as per our requirement to give you a consistent result.

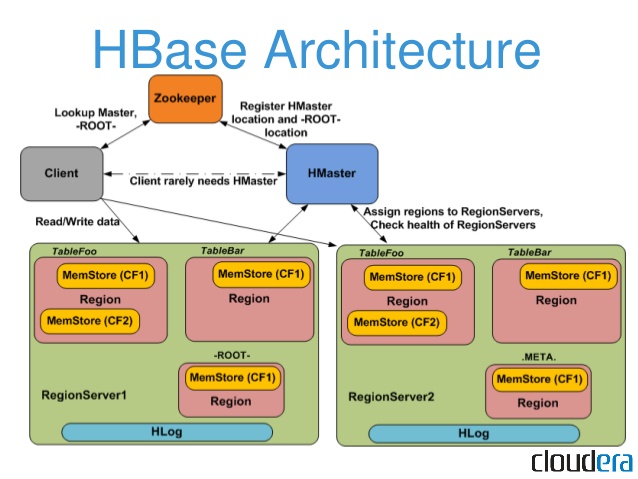
**HBase Architecture**

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 Sharing).

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 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 –

* Manages and Monitors the Hadoop Cluster
* Performs Administration (Interface for creating, updating and deleting tables.)
* Controlling the fail over
* DDL operations are handled by the HMaster

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 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 Data Node.

HMaster contacts ZooKeeper to get the details of region servers. Various services that Zookeeper provides include:

* Tracking server failure and network partitions.
* Maintain Configuration Information

**HBase vs RDBMS**

|  |  |
| --- | --- |
| **HBase** | **RDBMS** |
| Column-oriented | Row-oriented |
| Flexible schema, add columns on the Fly | Fixed schema |
| Good with sparse tables. | Not optimized for sparse tables. |
| No query language | SQL |
| Wide tables | Narrow tables |
| Joins using MR – not optimized | Not really |
| De-normalize your data. | Normalize as you can |
| Horizontal scalability-just add hard war | Hard to share and scale |
| Consistent | Consistent |
| No transactions | Transactional |
| Good for semi-structured data as well as structured data | Good for structured data |
| |  | | --- | | Philosophy of scaling out for more users/data |   High Velocity data | |  | | --- | | Philosophy of scaling up for more users/data |   Moderate velocity data |