Assignment - 11.1

Task1.1: Explain the below concepts with an example in brief.

● Nosql Databases

● Types of Nosql Databases

● CAP Theorem

● HBase Architecture

● HBase vs RDBMS

Answer1.1:

No SQL Databases: NoSQL encompasses a wide variety of different database technologies that were developed in response to the demands presented in building modern applications:

- Developers are working with applications that create massive volumes of new, rapidly changing data types — structured, semi-structured, unstructured and polymorphic data.
- Long gone is the twelve-to-eighteen month waterfall development cycle. Now small teams work in agile sprints, iterating quickly and pushing code every week or two, some even multiple times every day.
- Applications that once served a finite audience are now delivered as services that must be always-on, accessible from many different devices and scaled globally to millions of users.
- Organizations are now turning to scale-out architectures using open source software, commodity servers and cloud computing instead of large monolithic servers and storage infrastructure.

NoSQL Database Types

- **Document databases** pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.
- **Graph stores** are used to store information about networks of data, such as social connections. Graph stores include Neo4J and Giraph.
- **Key-value stores** are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value. Examples of key-value stores are Riak and Berkeley DB. Some key-value stores, such as Redis, allow each value to have a type, such as 'integer', which adds functionality.
- **Wide-column stores** such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows.

CAP Theorem

- **Consistency** means that data is the same across the cluster, so you can read or write from/to any node and get the same data.
- Availability means the ability to access the cluster even if a node in the cluster goes down.
- Partition tolerance means that the cluster continues to function even if there is a "partition" (communication break) between two nodes (both nodes are up, but can't communicate).

In order to get both availability and partition tolerance, you have to give up consistency. Consider if you have two nodes, X and Y, in a master-master setup. Now, there is a break between network communication between X and Y, so they can't sync updates. At this point you can either:

A) Allow the nodes to get out of sync (giving up consistency), or

B) Consider the cluster to be "down" (giving up availability)

All the combinations available are:

- **CA** - data is consistent between all nodes - as long as all nodes are online - and you can read/write from any node and be sure that the data is the same, but if you ever develop a partition between nodes, the data will be out of sync (and won't re-sync once the partition is resolved).
- **CP** - data is consistent between all nodes, and maintains partition tolerance (preventing data desync) by becoming unavailable when a node goes down.
- **AP** - nodes remain online even if they can't communicate with each other and will resync data once the partition is resolved, but you aren't guaranteed that all nodes will have the same data (either during or after the partition)

HBase Architecture

As we know, HBase is a column-oriented NoSQL database. Although it looks similar to a relational database which contains rows and columns, but it is not a relational database. Relational databases are row oriented while HBase is column-oriented.

In a column-oriented databases, all the column values are stored together like first column values will be stored together, then the second column values will be stored together and data in other columns are stored in a similar manner.

- When the amount of data is very huge, like in terms of petabytes or exabytes, we use column-oriented approach, because the data of a single column is stored together and can be accessed faster.
- While row-oriented approach comparatively handles less number of rows and columns efficiently, as row-oriented database stores data is a structured format.
- When we need to process and analyze a large set of semi-structured or unstructured data, we use column oriented approach. Such as applications dealing with Online Analytical Processing like data mining, data warehousing, applications including analytics, etc.
- Whereas, Online Transactional Processing such as banking and finance domains which handle structured data and require transactional properties (ACID properties) use row-oriented approach.

HBase tables has following components:

- **Tables:** Data is stored in a table format in HBase. But here tables are in column-oriented format.
- **Row Key:** Row keys are used to search records which make searches fast. You would be curious to know how? I will explain it in the architecture part moving ahead in this blog.
- **Column Families:** Various columns are combined in a column family. These column families are stored together which makes the searching process faster because data belonging to same column family can be accessed together in a single seek.
- **Column Qualifiers:** Each column's name is known as its column qualifier.
- **Cell:** Data is stored in cells. The data is dumped into cells which are specifically identified by rowkey and column qualifiers.
- **Timestamp:** Timestamp is a combination of date and time. Whenever data is stored, it is stored with its timestamp. This makes easy to search for a particular version of data.

Components of HBase Architecture

HBase has three major components i.e., **HMaster Server**, **HBase Region Server, Regions** and **Zookeeper**.

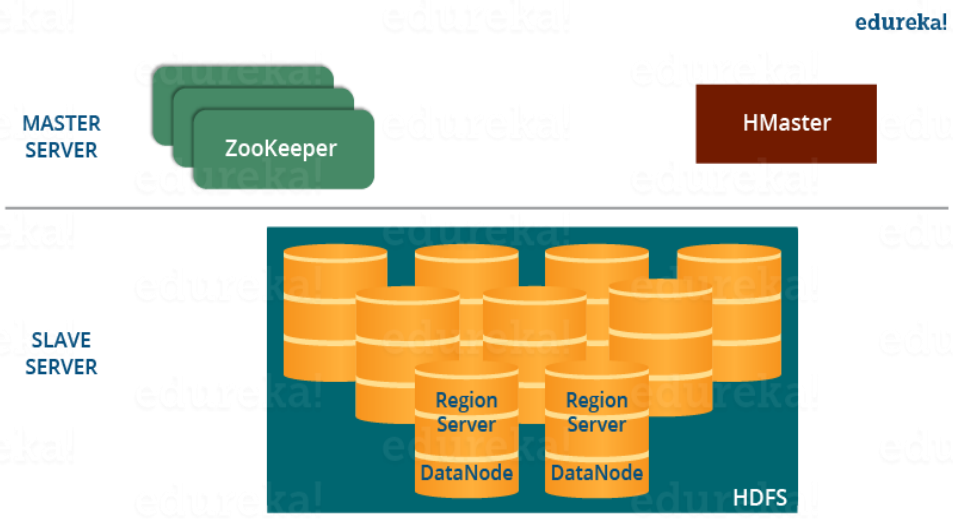

Figure: HMaster & Region Servers

HBase Architecture: Region

A region contains all the rows between the start key and the end key assigned to that region. HBase tables can be divided into a number of regions in such a way that all the columns of a column family is stored in one region. Each region contains the rows in a sorted order.

Many regions are assigned to a **Region Server**, which is responsible for handling, managing, executing reads and writes operations on that set of regions.

HBase Architecture: HMaster

As in the below image, you can see the HMaster handles a collection of Region Server which resides on DataNode. Let us understand how HMaster does that.

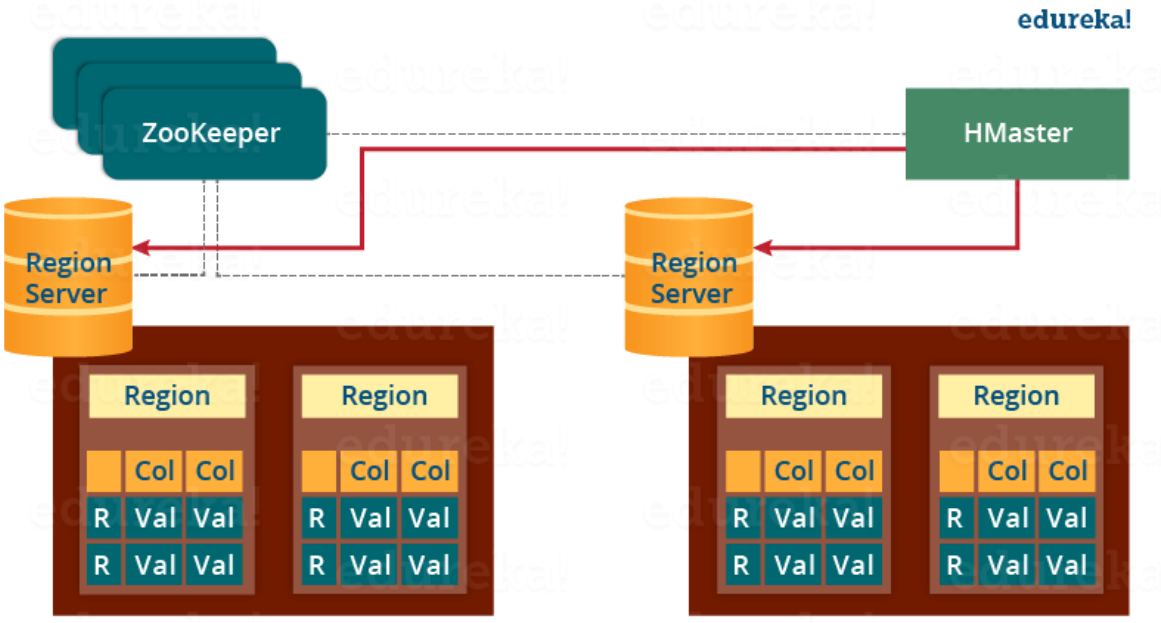

Figure: Components of HBase

HBase has a distributed and huge environment where HMaster alone is not sufficient to manage everything. So, you would be wondering what helps HMaster to manage this huge environment? That's where ZooKeeper comes into the picture. After we understood how HMaster manages HBase environment, we will understand how Zookeeper helps HMaster in managing the environment.

HBase Architecture: ZooKeeper – The Coordinator

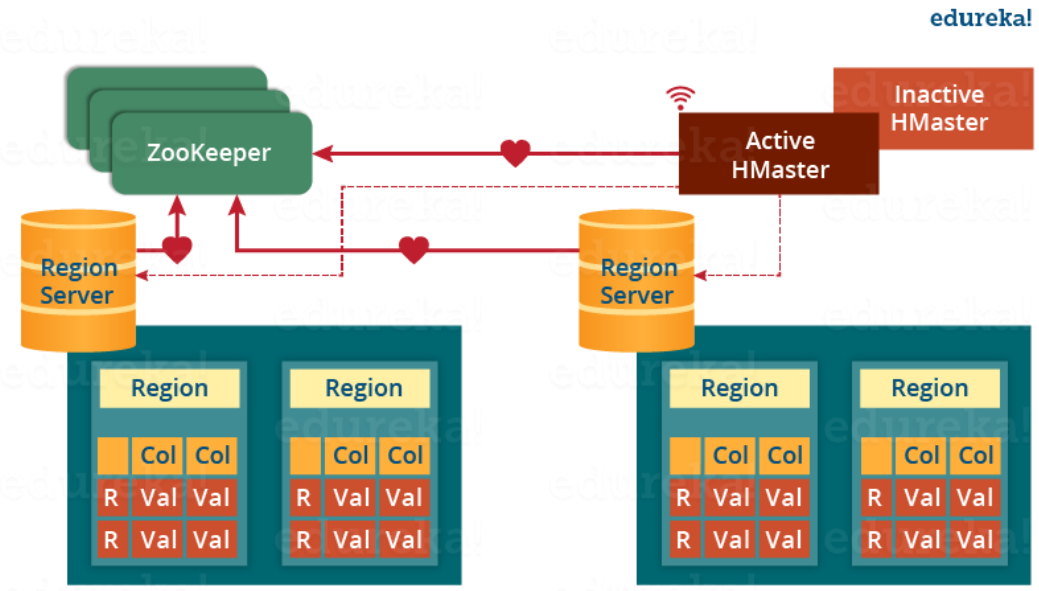

Figure: ZooKeeper as Coordination Service

- Zookeeper acts like a coordinator inside HBase distributed environment. It helps in maintaining server state inside the cluster by communicating through sessions.
- Every Region Server along with HMaster Server sends continuous heartbeat at regular interval to Zookeeper and it checks which server is alive and available as mentioned in above image. It also provides server failure notifications so that, recovery measures can be executed.
- Referring from the above image you can see, there is an inactive server, which acts as a backup for active server. If the active server fails, it comes for the rescue.
- The active HMaster sends heartbeats to the Zookeeper while the inactive HMaster listens for the notification send by active HMaster. If the active HMaster fails to send a heartbeat the session is deleted and the inactive HMaster becomes active.
- While if a Region Server fails to send a heartbeat, the session is expired and all listeners are notified about it. Then HMaster performs suitable recovery actions which we will discuss later in this blog.
- Zookeeper also maintains the .META Server's path, which helps any client in searching for any region. The Client first has to check with .META Server in which Region Server a region belongs, and it gets the path of that Region Server.

HBase vs RDBMS:

HBase 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.

Task2: Execute blog present in below link - https://acadgild.com/blog/importtsv-data-from-hdfs-into-hbase/

Answer2: A utility that loads data in the TSV format into HBase. ImportTsv takes data from HDFS into HBase via Puts.

Open the terminal and start all the daemons including Hbase daemons and MR History server.

```
[acadgild@localhost ~]$ jps
2881 DataNode
3218 ResourceManager
2757 NameNode
3029 SecondaryNameNode
4069 HMaster
3973 HQuorumPeer
3320 NodeManager
11288 Jps
4187 HRegionServer
5243 Main
3679 JobHistoryServer
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost ~]$
```

Now in Hbase shell, run the command to create table 'bulktable.

```
hbase(main):002:0> create 'bulktable', 'cf1', 'cf2'
0 row(s) in 1.4380 seconds

=> Hbase::Table - bulktable
hbase(main):003:0>
```

Create a directory in local and get into it using

mkdir hbase

cd hbase

```
[acadgild@localhost ~]$ cd hbase
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost hbase]$
```

Create **bulk_data.tsv** file in the same location and insert some data.

Once the file is created – move it to HDFS in using put command.

Now run below command to import the data from HDFS to HBase.

hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=' ' -Dimporttsv.columns="HBASE_ROW_KEY,cf1:name,cf2:exp" bulktable /bulk_data.tsv

```
[acadgild@localhost ~]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=' ' -Dimporttsv.columns="HBASE_ROW_KEY
,cf1:name,cf2:exp" bulktable /bulk_data.tsv
2018-04-17 00:45:37,839 WARN  [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-ja
va classes where applicable
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/acadgild/install/hbase/hbase-1.2.6/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBin
der.class]
SLF4J: Found binding in [jar:file:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4
```

Now check in HBase for using scan command as below.

```
hbase(main):001:0> scan 'bulktable'
ROW                                COLUMN+CELL
 1                                 column=cf1:name, timestamp=1523906137787, value=Amit
 1                                 column=cf2:exp, timestamp=1523906137787, value=4
 2                                 column=cf1:name, timestamp=1523906137787, value=Girija
 2                                 column=cf2:exp, timestamp=1523906137787, value=3
 3                                 column=cf1:name, timestamp=1523906137787, value=Jatin
 3                                 column=cf2:exp, timestamp=1523906137787, value=5
 4                                 column=cf1:name, timestamp=1523906137787, value=Swati
 4                                 column=cf2:exp, timestamp=1523906137787, value=3
4 row(s) in 0.5480 seconds

hbase(main):002:0> You have new mail in /var/spool/mail/acadgild
[acadgild@localhost ~]$
```