Task1

Explain the below concepts with an example in brief.

● NoSQL Databases

NoSQL databases resolve this issue by incorporating a wide range of technologies that make the systems **scalable and suitable for big data operations**. This is done by **distributing the load among several Intel-based cheap servers** which empowers the platforms for real-time processing. Need to handle large volumes of structured, semi-structured, and unstructured data.

Example HBASE

The main advantages of storing data in **columns** over relational DBMS are fast search/access and data aggregation. Relational databases store a single row as a continuous disk entry. **Different rows are stored in different places** on the disk while columnar databases **store all the cells corresponding to a column as a continuous disk entry**, thus making the search/access faster.

● Types of NoSQL Databases

Types of the NoSQL databases,

HBASE

Cassandra

MongoDB

HBASE

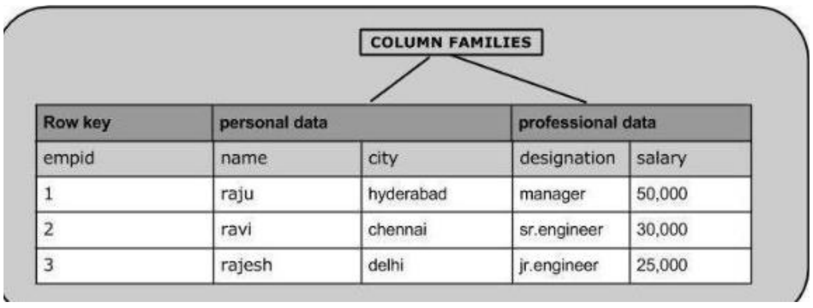
HBase is a column-oriented database and the tables in it are sorted by row. The table schema defines only column families, which are the key value pairs. A table have multiple column families and each column family can have any number of columns. Subsequent column values are stored contiguously on the disk. Each cell value of the table has a timestamp. In short, in an HBase:

Table is a collection of rows.

Row is a collection of column families.

Column family is a collection of columns.

Column is a collection of key value pairs.


COLUMN FAMILIES

Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

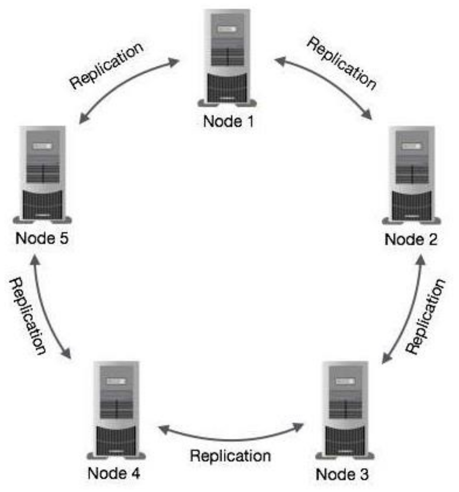
Cassandra:

Cassandra is a distributed database from Apache that is highly scalable and designed to manage very large amounts of structured data. It provides high availability with no single point of failure. Cassandra is for OLTP.

The syntax of creating a Keyspace is as follows –

CREATE KEYSPACE Keyspace name

WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};



It is not suitable for transactional databases but can be used in situations where you need to ingest large amounts of data and then query the data in real time.

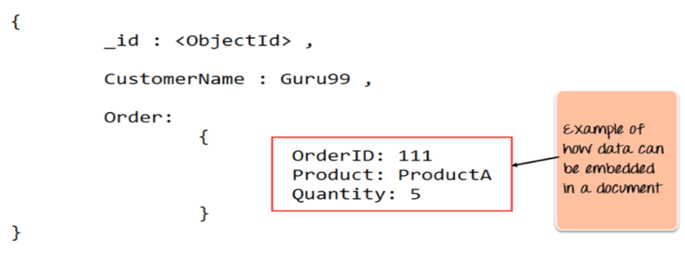
MONGODB:

MongoDB is a document database. Each database contains collections which in turn contains documents. Each document can be different with varying number of fields. The size and content of each document can be different from each other.

EXAMPLE

The _id field is added by MongoDB to uniquely identify the document in the collection.

What you can note is that the Order Data (OrderID , Product and Quantity) which in RDBMS will normally be stored in a separate table, while in MongoDB it is actually stored as an embedded document in the collection itself. This is one of the key differences of how data is modelled in MongoDB.



```
{
        _id : <ObjectId> ,

        CustomerName : Guru99 ,

        Order:
                {
                      OrderID: 111
                      Product: ProductA
                      Quantity: 5

                }
}
```

Example of how data can be embedded in a document

CAP THEOREM:

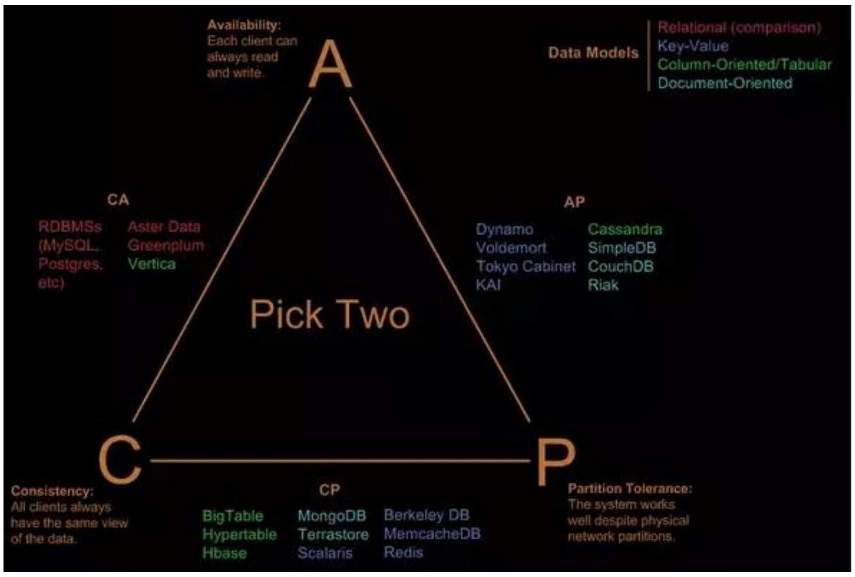
The CAP theorem by Eric Brewer _originally_ states that any distributed system with shared data is likely to see tension between the following systemic properties:

Consistency - All the replicas are in sync and maintain the same state of any given object at any given point of time. Also known as sequential consistency.

Availability - A request will eventually complete successfully. A read/write request on any node of the system will never be rejected as long as the particular node is up and running.

Network Partition Tolerance - When the network connecting the nodes goes down, the system will still continue to operate even though some/all nodes can longer communicate with each other.

Alternately, the theorem states that it is possible for the system to satisfy any two but not all of the properties mentioned above.

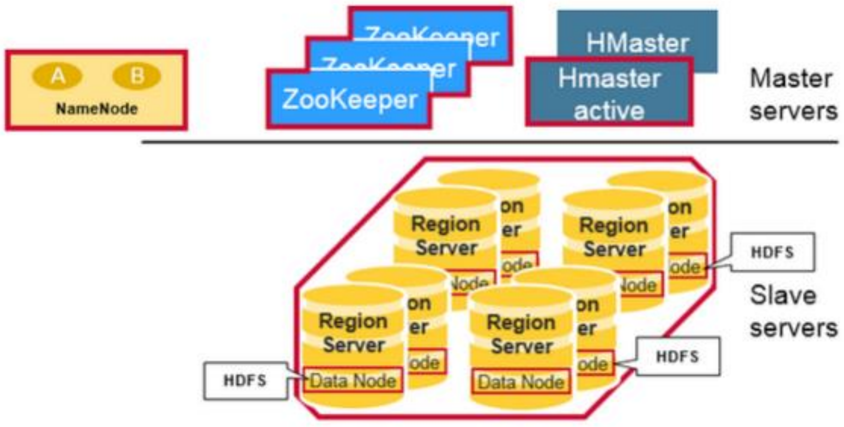


● **HBase Architecture**

Physically, HBase is composed of three types of servers in a master slave type of architecture. Region servers serve data for reads and writes. When accessing data, clients communicate with HBase RegionServers directly. Region assignment, DDL (create, delete tables) operations are handled by the HBase Master process. Zookeeper, which is part of HDFS, maintains a live cluster state.

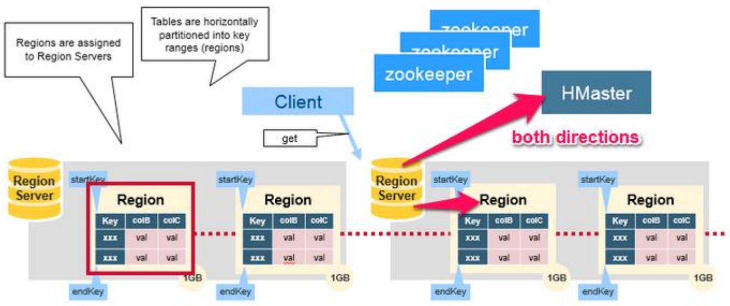
The Hadoop DataNode stores the data that the Region Server is managing. All HBase data is stored in HDFS files. Region Servers are collocated with the HDFS DataNodes, which enable data locality (putting the data close to where it is needed) for the data served by the RegionServers. HBase data is local when it is written, but when a region is moved, it is not local until compaction.

The NameNode maintains metadata information for all the physical data blocks that comprise the files.

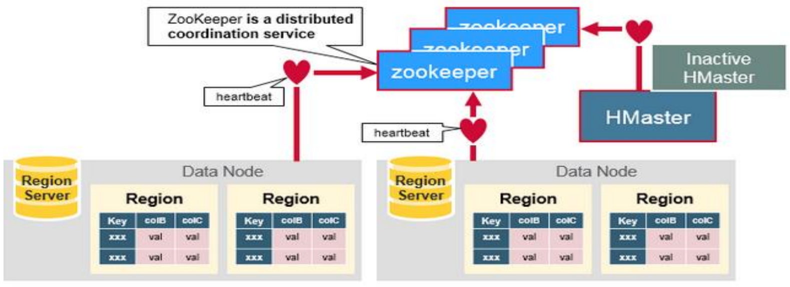


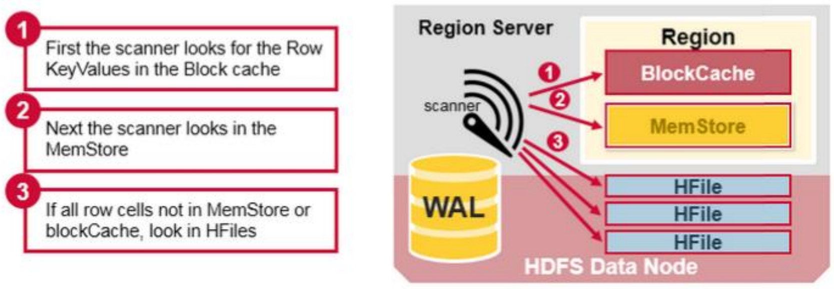
Regions

HBase Tables are divided horizontally by row key range into “Regions.” A region contains all rows in the table between the region’s start key and end key. Regions are assigned to the nodes in the cluster, called “Region Servers,” and these serve data for reads and writes. A region server can serve about 1,000 regions.

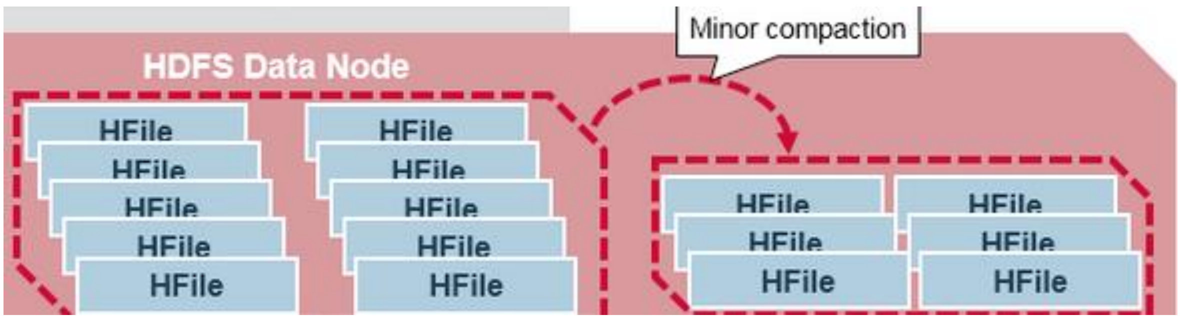


ZoopKeeper:

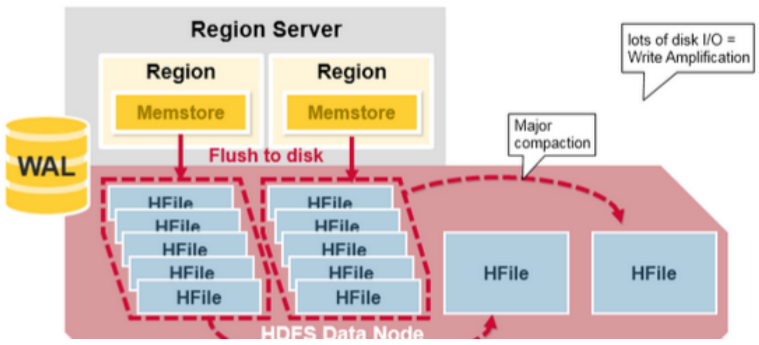



1 First the scanner looks for the Row KeyValues in the Block cache
2 Next the scanner looks in the MemStore
3 If all row cells not in MemStore or blockCache, look in HFiles
Region Server
Region
BlockCache
MemStore
scanner
WAL
HFile
HFile
HFile
HDFS Data Node

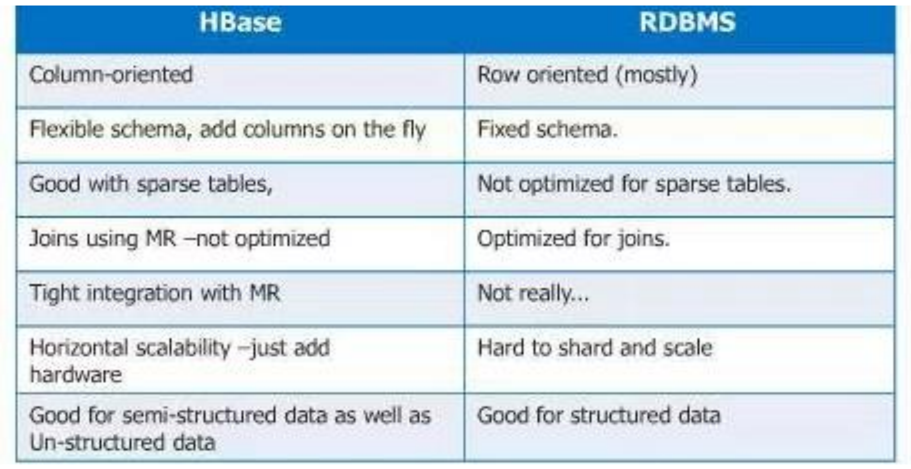
Minor compaction :small hfiles in to other hfiles



Major compaction : Major compaction merges and rewrites all the HFiles in a region to one HFile per column family,



Hbase vs RDBMS



HBase	RDBMS
Column-oriented	Row oriented (mostly)
Flexible schema, add columns on the fly	Fixed schema.
Good with sparse tables,	Not optimized for sparse tables.
Joins using MR –not optimized	Optimized for joins.
Tight integration with MR	Not really...
Horizontal scalability –just add hardware	Hard to shard and scale
Good for semi-structured data as well as Un-structured data	Good for structured data

TASK 2

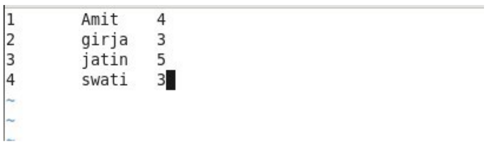
ImportTSV Data from HDFS into HBase

hbase(main):001:0> create 'bulktable','cf1','cf2'

=> Hbase::Table – bulktable

[acadgild@localhost ~]$ mkdir hbase

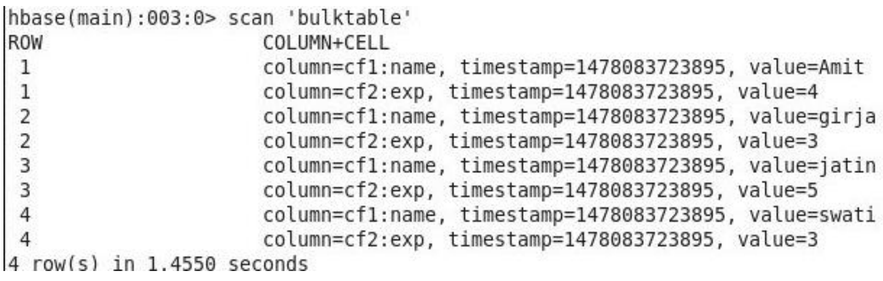
[acadgild@localhost ~]$ cd hbaseCreate bulk_data.csv with below data



```
1       Amit    4
2       girja   3
3       jatin   5
4       swati   3
~
~
~
```

[acadgild@localhost hbase]$ **hbase org.apache.hadoop.hbase.mapreduce.ImportTsv –Dimporttsv.columns=HBASE_ROW_KEY,cf1:name,cf2:exp bulktable /hbase/bulk_data.tsv**

OUTPUT



```
hbase(main):003:0> scan 'bulktable'
ROW                   COLUMN+CELL
 1                    column=cf1:name, timestamp=1478083723895, value=Amit
 1                    column=cf2:exp, timestamp=1478083723895, value=4
 2                    column=cf1:name, timestamp=1478083723895, value=girja
 2                    column=cf2:exp, timestamp=1478083723895, value=3
 3                    column=cf1:name, timestamp=1478083723895, value=jatin
 3                    column=cf2:exp, timestamp=1478083723895, value=5
 4                    column=cf1:name, timestamp=1478083723895, value=swati
 4                    column=cf2:exp, timestamp=1478083723895, value=3
4 row(s) in 1.4550 seconds
```