Session 11:

ADVANCE HBASE

Assignment 1

PROBLEM STATEMENT –

Task1

Explain the below concepts with an example in brief.

● Nosql Databases

● Types of Nosql Databases

● CAP Theorem

● HBase Architecture

● HBase vs RDBMS

SOLUTION –

1. Nosql Databases

NoSQL refers to a database that is not based on SQL (Structured Query Language), which is the language most commonly associated with relational databases. The term "NoSQL" refers to the fact that traditional relational databases are not adequate for all solutions, particularly ones involving large volumes of data.

NoSQL database, also called **Not Only SQL**, is an approach to data management and database design that's useful for very large sets of distributed data. NoSQL, which encompasses a wide range of technologies and architectures, seeks to solve the scalability and big data performance issues that relational databases weren't designed to address.

NoSQL database examples:

MongoDB, CouchDB, GemFire, Redis, Cassandra, memcached, Hazlecast, HBASE, Mnesia and Neo4j.

2. Types of Nosql Databases

Several different varieties of NoSQL databases have been created to support specific needs and use cases. These databases can broadly be categorized into four types.

1. **Key-value** store NoSQL database
2. **Document** store NoSQL database
3. **Column** store NoSQL database
4. **Graph** base NoSQL database

Example,

Key/Value Databases	Voldemort, Redis, Scalaris
Columnar Databases	HBase, Hypertable, Cassandra
Document Databases	MongoDb, CouchDB
Graph Databases	InfoGrid, Neo4j

3. CAP Theorem

The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes. In other words, the CAP theorem states that in the presence of a network partition, one has to choose between consistency and availability.

Consistent: this means the data in the database remains consistent after the execution of an operation. For example after an update operation, all clients see the same operation.

Availability: this means the system is always on availability, no downtime.

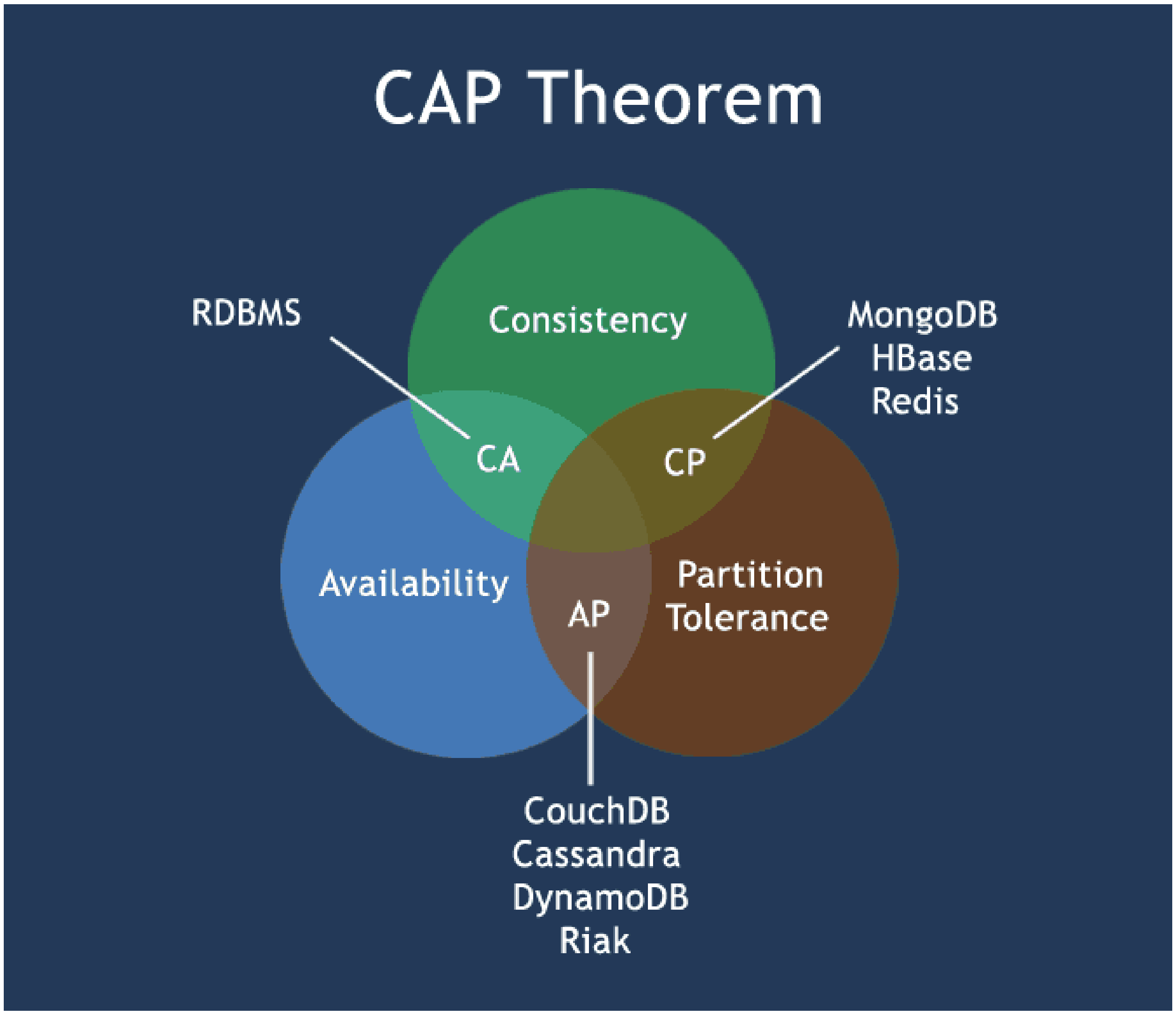
Partition Tolerance: this means the system continues to function even if the communication among the servers are unreliable, i.e., the servers may be partitioned into multiple groups that cannot communicate with one another.

In theoretically it is impossible to fulfill all 3 requirements. CAP provides the basic requirements for a distributed system to follow 2 of the 3 requirements. Therefore all the current NoSQL database follow the different combinations of the C, A, P from the CAP theorem. Here is the brief description of three combinations CA, CP, AP.

CA - Single site cluster, therefore all nodes are always in contact. When a partition occurs, the system blocks.

CP -Some data may not be accessible, but the rest is still consistent/accurate.

AP - System is still available under partitioning, but some of the data returned may be inaccurate.



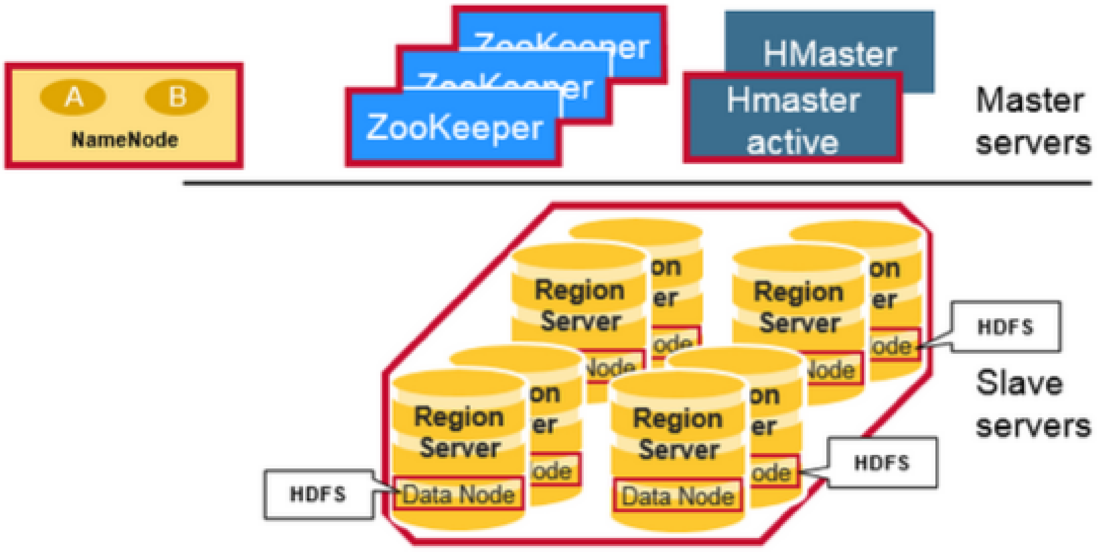
4. HBase Architecture

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.

HBase Master - Region assignment, DDL (create, delete tables) operations are handled by this process.

Zookeeper - which is part of HDFS, maintains a live cluster state.



Master-Slave Architectural diagram of HBASE

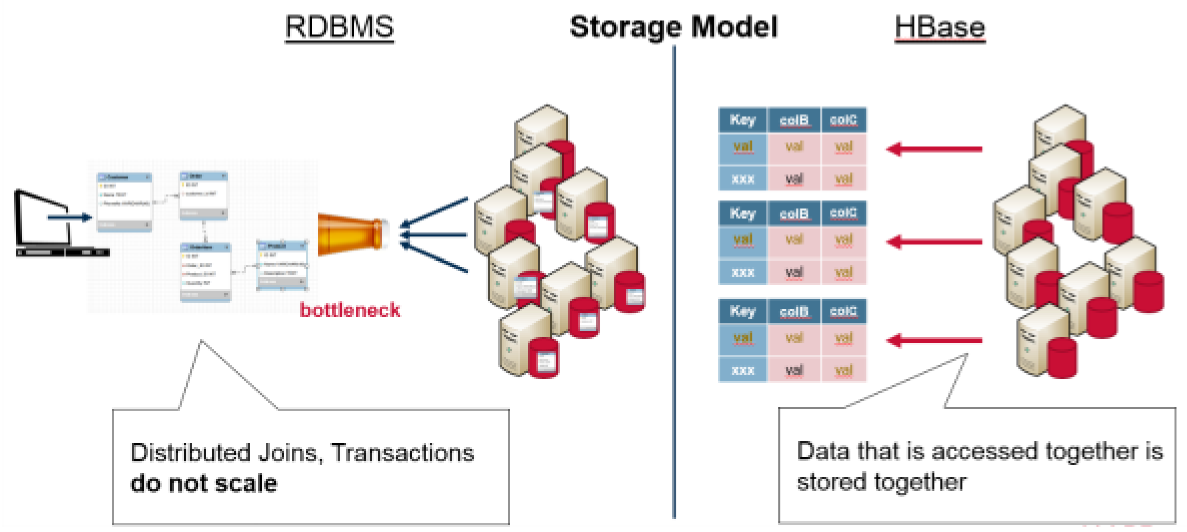
The Hadoop **DataNode** stores the data that the Region Server is managing. All HBase data is stored in HDFS files. Region Servers are collocated with the HDFS DataNodes, which enable data locality (putting the data close to where it is needed) for the data served by the RegionServers. HBase data is 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.

5. HBase vs RDBMS

Difference between HBASE and RDBMS:

HBASE	**RDBMS**
Distributed, Column-oriented	Row-oriented(mostly)
HBase tables guarantee consistency and partition tolerance	RDBMS tables guarantee ACID properties
Flexible schema, add columns on the Fly	Fixed schema
Good with sparse tables.	Not optimized for sparse tables.
uses Java client API and uses Java client API and JRuby	SQL
Wide tables	Narrow Tables
Joins using MR – not optimized	optimized for Joins(small, fast ones)
Tight – Integration with MR	Not really


RDBMS
Storage Model
HBase
bottleneck
Key colB colC
val val val
xxx val val
Distributed Joins, Transactions **do not scale**
Data that is accessed together is stored together

Task2

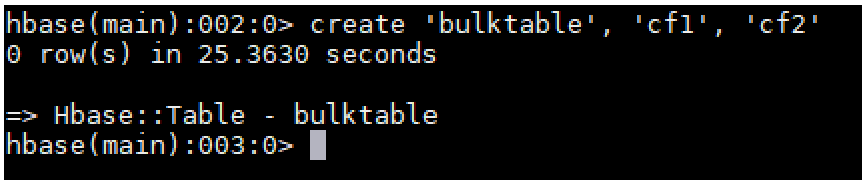
Execute blog present in below link

https://acadgild.com/blog/importtsv-data-from-hdfs-into-hbase/

Step 1:

Create a table with two column family.

create ‘bulktable’, ‘cf1’, ‘cf2’



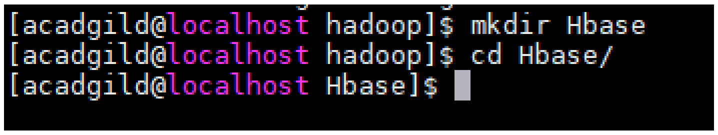
```
hbase(main):002:0> create 'bulktable', 'cf1', 'cf2'
0 row(s) in 25.3630 seconds

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

Step 2:

Create a directory in local drive.

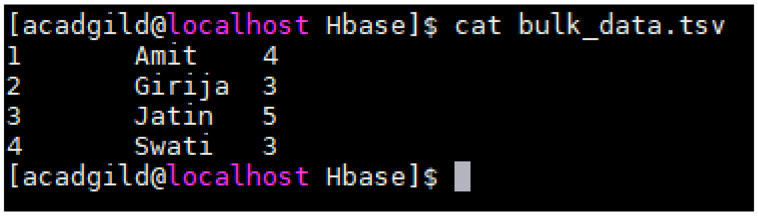
mkdir Hbase



```
[acadgild@localhost hadoop]$ mkdir Hbase
[acadgild@localhost hadoop]$ cd Hbase/
[acadgild@localhost Hbase]$ 
```

Step 3:

Create a file inside the HBase directory named bulk_data.tsv with tab separated data.

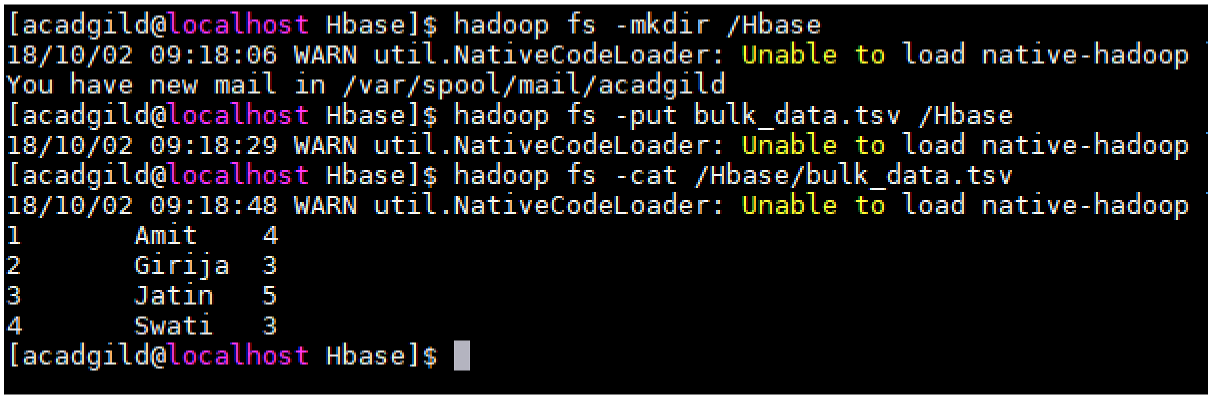


```
[acadgild@localhost Hbase]$ cat bulk_data.tsv
1       Amit    4
2       Girija  3
3       Jatin   5
4       Swati   3
[acadgild@localhost Hbase]$ 
```

Step 4:

Data should be present in HDFS while performing the import task to Hbase.

Copy the file bulk_data.tsv to HDFS.

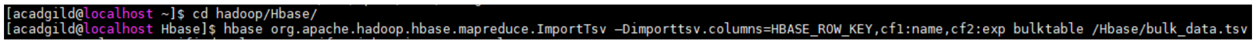


```
[acadgild@localhost Hbase]$ hadoop fs -mkdir /Hbase
18/10/02 09:18:06 WARN util.NativeCodeLoader: Unable to load native-hadoop
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost Hbase]$ hadoop fs -put bulk_data.tsv /Hbase
18/10/02 09:18:29 WARN util.NativeCodeLoader: Unable to load native-hadoop
[acadgild@localhost Hbase]$ hadoop fs -cat /Hbase/bulk_data.tsv
18/10/02 09:18:48 WARN util.NativeCodeLoader: Unable to load native-hadoop
1       Amit    4
2       Girija  3
3       Jatin   5
4       Swati   3
[acadgild@localhost Hbase]$ 
```

Step 5:

Run ImportTsv command to import data into the Hbase table.

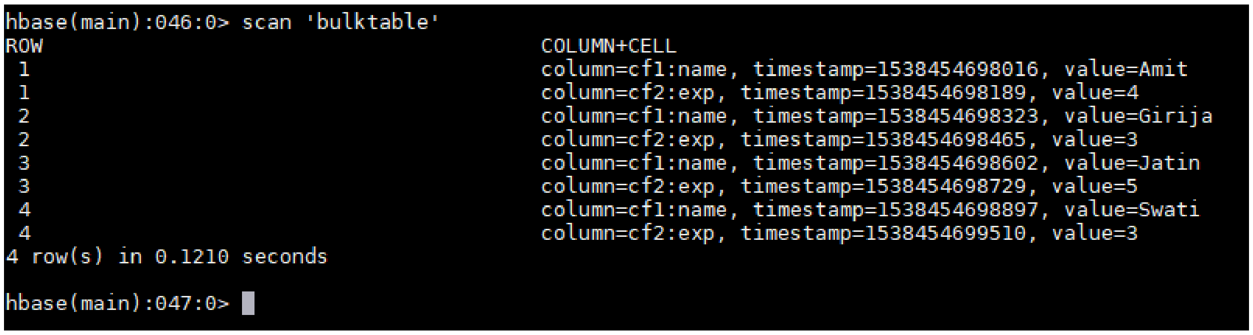
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv –Dimporttsv.columns=HBASE_ROW_KEY,cf1:name,cf2:exp bulktable /hbase/bulk_data.tsv



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

Now check whether the data is present in the Hbase table by using the below command.

scan ‘bulktable’



```
hbase(main):046:0> scan 'bulktable'
ROW                                        COLUMN+CELL
 1                                         column=cf1:name, timestamp=1538454698016, value=Amit
 1                                         column=cf2:exp, timestamp=1538454698189, value=4
 2                                         column=cf1:name, timestamp=1538454698323, value=Girija
 2                                         column=cf2:exp, timestamp=1538454698465, value=3
 3                                         column=cf1:name, timestamp=1538454698602, value=Jatin
 3                                         column=cf2:exp, timestamp=1538454698729, value=5
 4                                         column=cf1:name, timestamp=1538454698897, value=Swati
 4                                         column=cf2:exp, timestamp=1538454699510, value=3
4 row(s) in 0.1210 seconds

hbase(main):047:0>
```

We can see that data is successfully imported to the Hbase table.