Problem Statement

1. Nosql Databases
2. Types of Nosql Databases
3. CAP Theorem
4. HBase Architecture
5. HBase vs RDBMSObjective

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 is especially useful when an enterprise needs to access and analyze massive amounts of unstructured data or data that's stored remotely on multiple virtual servers in the cloud.
NoSQL data stores respond to key data needs that are not met by relational databases.
NoSQL database examples:
Dozens of NoSQL data stores are available; the following are among the most popular:
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 (Brewer's 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.

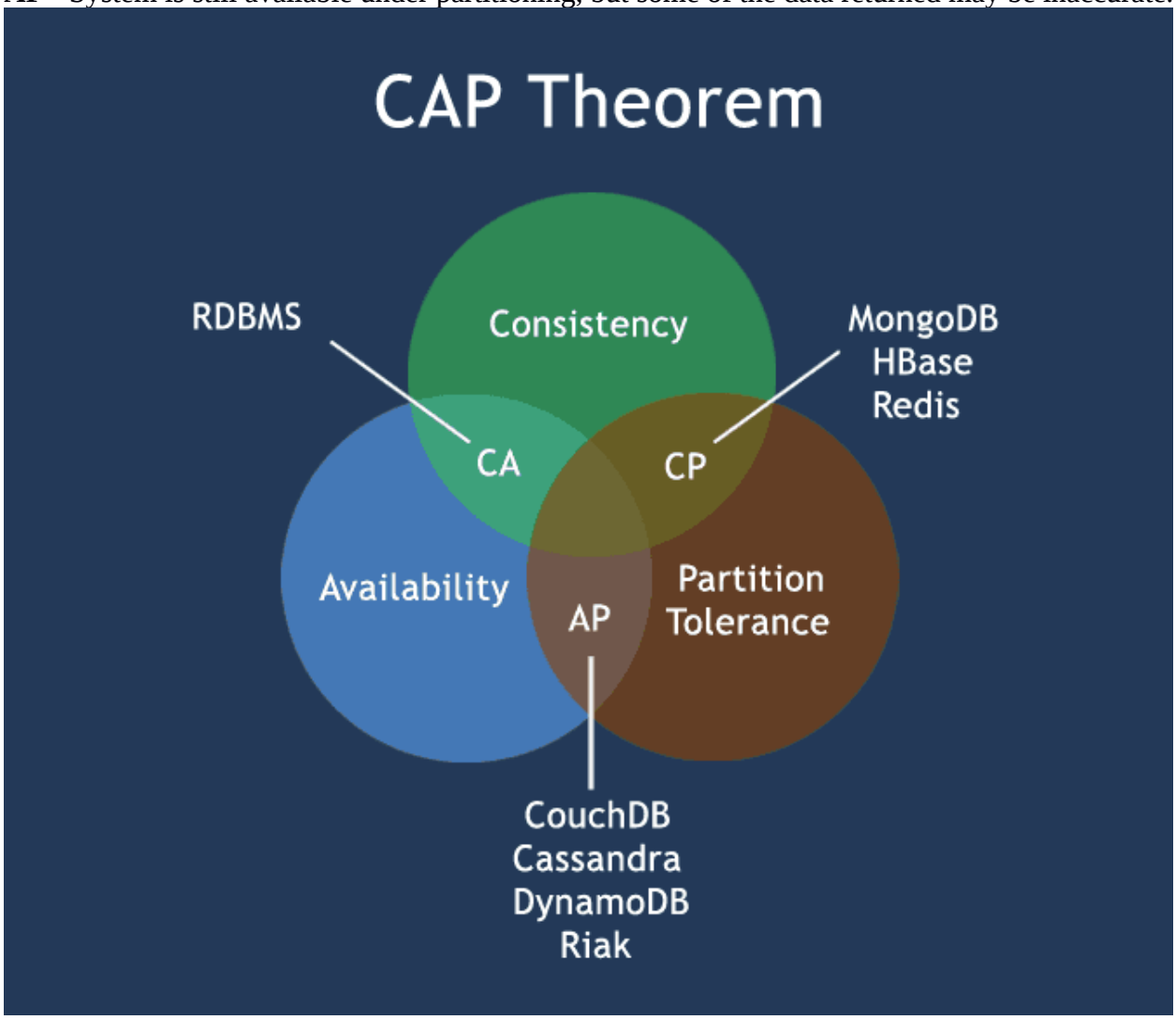
Partition Tolerance: this means the system continues to function even if the communication among the servers are unreliable, i.e., the servers may be partitioned into multiple groups that cannot communicate with one another.

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.

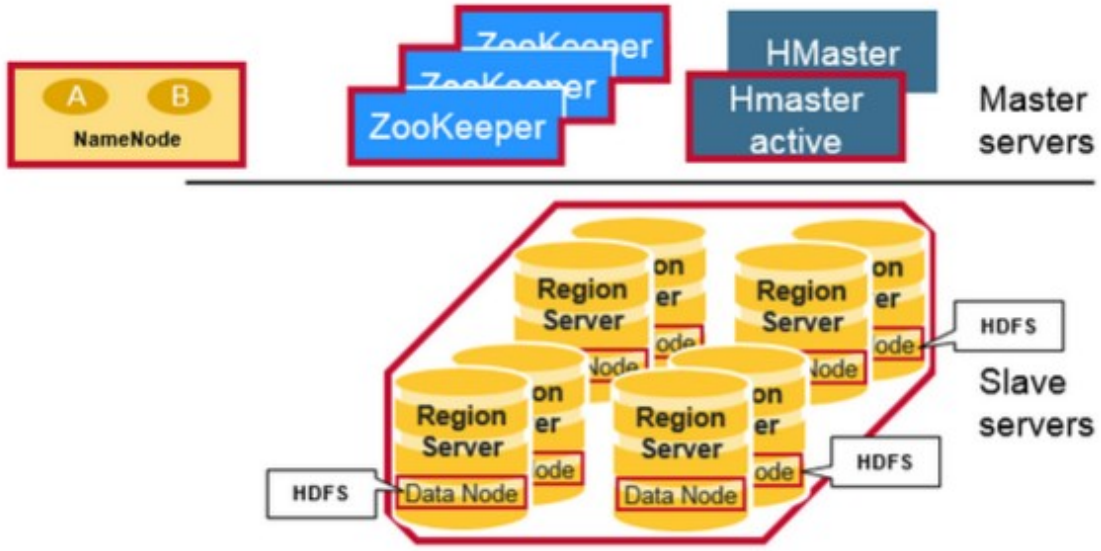
AP - System is still available under partitioning, but some of the data returned may be inaccurate.



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.

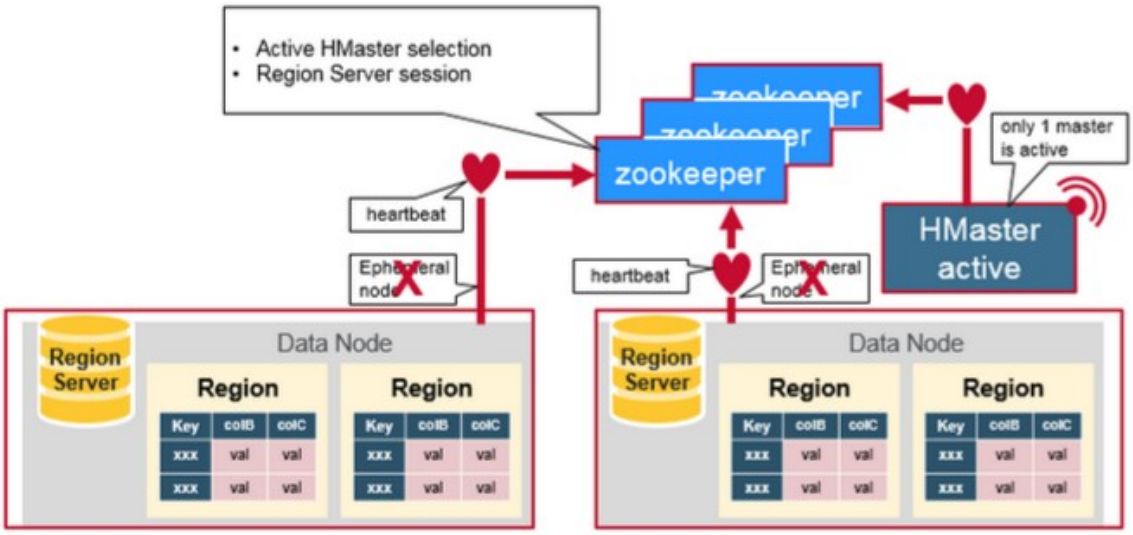


Master-Slave Architectural diagram of HBASE

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.

How the Components Work Together:

Zookeeper is used to coordinate shared state information for members of distributed systems. Region servers and the active **HMaster** connect with a session to **ZooKeeper**. The **ZooKeeper** maintains ephemeral nodes for active sessions via heartbeats.

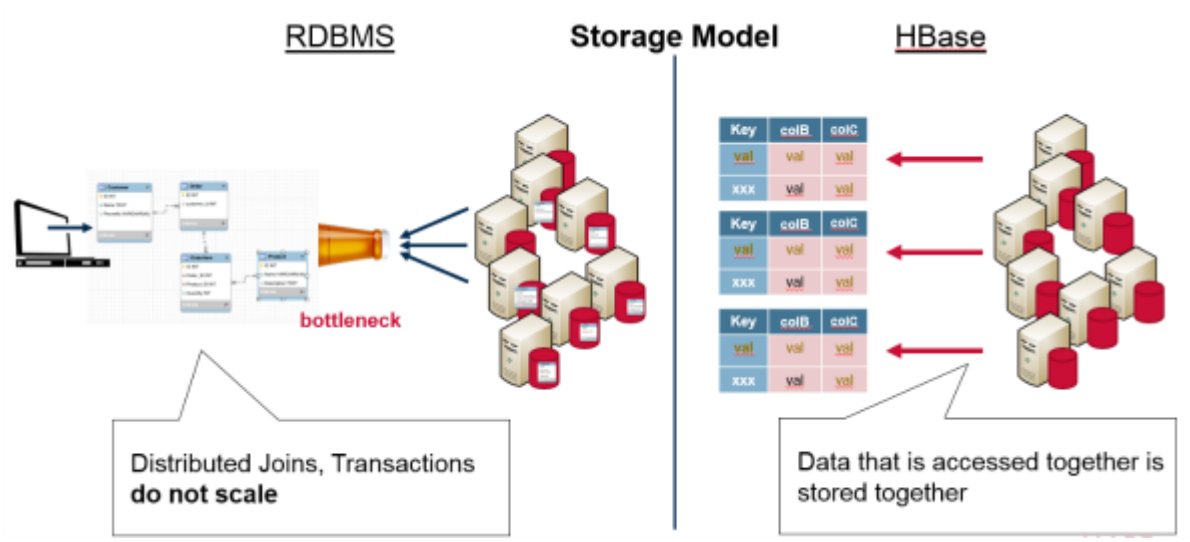


Each Region Server creates an ephemeral node. The **HMaster** monitors these nodes to discover available **region servers**, and it also monitors these nodes for server failures. **HMasters** vie to create an ephemeral node. **Zookeeper** determines the first one and uses it to make sure that only one master is active. The active **HMaster** sends heartbeats to **Zookeeper**, and the inactive **HMaster** listens for notifications of the active **HMaster** failure.
If a **region server** or the active **HMaster** fails to send a heartbeat, the session is expired and the corresponding ephemeral node is deleted. Listeners for updates will be notified of the deleted nodes. The active **HMaster** listens for region servers, and will recover region servers on failure. The Inactive **HMaster** listens for active **HMaster** failure, and if an active **HMaster** fails, the inactive **HMaster** becomes active.

5. HBase vs RDBMS

Hadoop and **RDBMS** are varying concepts of processing, retrieving and storing the data or information. While Hadoop is an open-source Apache project, RDBMS stands for Relational Database Management System. Hadoop framework has been written in Java which makes it scalable and makes it able to support applications that call for high performance standards. Hadoop framework enables the storage of large amounts of data on files systems of multiple computers. Hadoop is configured to allow scalability from a single computer node to several thousands of nodes or independent workstations in a manner that the individual nodes utilize local computer storage CPU processing power and memory.
Database Management Systems focus on the data storage in table form which includes columns and rows. SQL is utilized to retrieve needed data which is stored in such tables. The RDBMS concept stores relationships between such tables in various forms so that one column of entries of a particular table will act as a reference for another table. Such column values are known as primary keys and foreign keys with the keys being used to reference other existing tables so that the appropriate data can be related and also be retrieved by combining such tables using SQL queries as required. The tables and relationships can be altered by integrating relevant tables using SQL queries.
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



Task2

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

Start History Server

./mr-jobhistory-daemon.sh --config $HADOOP_CONF_DIR start historyserver

```
[acadgild@localhost sbin]$ mr-jobhistory-daemon.sh --config $HADOOP_CONF_DIR start historyserver
starting historyserver, logging to /home/acadgild/install/hadoop/hadoop-2.6.5/logs/mapred-acadgild-historyserver-localhost.localdomain.out
[acadgild@localhost sbin]$ jps
9874 JobHistoryServer
3618 NodeManager
9907 Jps
3078 NameNode
5206 HRegionServer
5113 HMaster
3370 SecondaryNameNode
3178 DataNode
3515 ResourceManager
5019 HQuorumPeer
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost sbin]$
[acadgild@localhost sbin]$
```

Steps:
1.Create 'bulktable', 'cf1', 'cf2'

2. Come out of HBase shell to the terminal and also make a directory for Hbase in the local drive; So,
since you have your own path you can use it.

mkdir -p Hbase

3.

Create a file inside the HBase directory named bulk_data.tsv with tab separated data inside using below command in terminal.
vi hbase/bulk_data.tsv

4.
we copy the data inside HDFS

5.
hbase hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=HBASE_ROW_KEY,cf1:name,cf2:exp bulktable /hbase/data/bulk_data.tsv