Session 11 - ADVANCED HBASE

Assignment 1

Task 1:

Explain the below concepts with an example in brief.

- Nosql Databases
- Types of Nosql Databases
- CAP Theorem
- HBase Architecture
- HBase vs RDBMS

NoSQL Databases

NoSQL is an approach to database design that can accomodate a wide variety of data models, including key-value, document, columnar and graph formats. NoSQL, which stand for "not only SQL," is an alternative to traditional relational databases in which data is placed in tables and data schema is carefully designed before the database is built. NoSQL databases are especially useful for working with large sets of distributed data.

NoSQL databases have many advantages compared to traditional, relational databases.

One major, underlying difference is that NoSQL databases have a simple and flexible structure. They are schema-free.

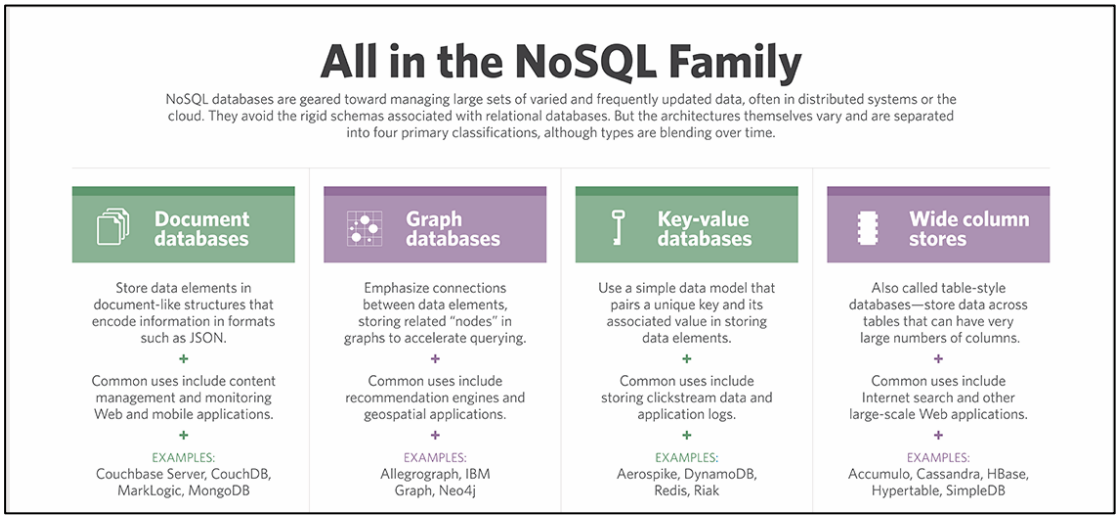
Unlike relational databases, NoSQL databases are based on key-value pairs. Some store types of NoSQL databases include column store, document store, key value store, graph store, object store, XML store, and other data store modes.

Usually, each value in the database has a key. Some NoSQL database stores also allow developers to store serialized objects into the database, not just simple string values.

Open-source NoSQL databases don't require expensive licensing fees and can run on inexpensive hardware, rendering their deployment cost-effective.

Also, when working with NoSQL databases, whether they are open-source or proprietary, expansion is easier and cheaper than when working with relational databases. This is because it's done by horizontally scaling and distributing the load on all nodes, rather than the type of vertical scaling that is usually done with relational database systems, which is replacing the main host with a more powerful one.

Types of NoSQL Databases



Document Store

Document stores are similar to key value stores in that they are schema-less and based on a key-value model. Both, therefore, share many of the same advantages and disadvantages. Both lack consistency on the database level, which makes way for applications to provide more reliability and consistency features.

There are however, key differences between the two.

In Document Stores, the values (documents) provide encoding for the data stored. Those encodings can be XML, JSON, or BSON (Binary encoded JSON).

Also, querying based on data can be done.

The most popular database application that relies on a Document Store is MongoDB.

Graph Base

In a Graph Base NoSQL Database, a directed graph structure is used to represent the data. The graph is comprised of edges and nodes.

Formally, a graph is a representation of a set of objects, where some pairs of the objects are connected by links. The interconnected objects are represented by mathematical abstractions, called vertices, and the links that connect some pairs of vertices are called edges. A set of vertices and the edges that connect them is said to be a graph.

Graph databases are most typically used in social networking applications. Graph databases allow developers to focus more on relations between objects rather than on the objects themselves. In this context, they indeed allow for a scalable and easy-to-use environment.

Currently, InfoGrid and InfiniteGraph are the most popular graph databases.

Key Value Store

In the Key Value store type, a hash table is used in which a unique key points to an item.

Keys can be organized into logical groups of keys, only requiring keys to be unique within their own group. This allows for identical keys in different logical groups. The following table shows an example of a key-value store, in which the key is the name of the city, and the value is the address for Ulster University in that city.

Some implementations of the key value store provide caching mechanisms, which greatly enhance their performance.

All that is needed to deal with the items stored in the database is the key. Data is stored in a form of a string, JSON, or BLOB (Binary Large OBject).

One of the biggest flaws in this form of database is the lack of consistency at the database level. This can be added by the developers with their own code, but as mentioned before, this adds more effort, complexity, and time.

The most famous NoSQL database that is built on a key value store is Amazon's DynamoDB.

Column Store

In a Column Store database, data is stored in columns, as opposed to being stored in rows as is done in most relational database management systems.

A Column Store is comprised of one or more Column Families that logically group certain columns in the database. A key is used to identify and point to a number of columns in the database, with a keyspace attribute that defines the scope of this key. Each column contains tuples of names and values, ordered and comma separated.

Column Stores have fast read/write access to the data stored. In a column store, rows that correspond to a single column are stored as a single disk entry. This makes for faster access during read/write operations.

The most popular databases that use the column store include Google's BigTable, HBase, and Cassandra.

CAP Theorem

In a distributed system, managing consistency(C), availability(A) and partition toleration(P) is important, Eric Brewer put forth the CAP theorem which states that in any distributed system we can choose only two of consistency, availability or partition tolerance. Many NoSQL databases try to provide options where the developer has choices where they can tune the database as per their needs.

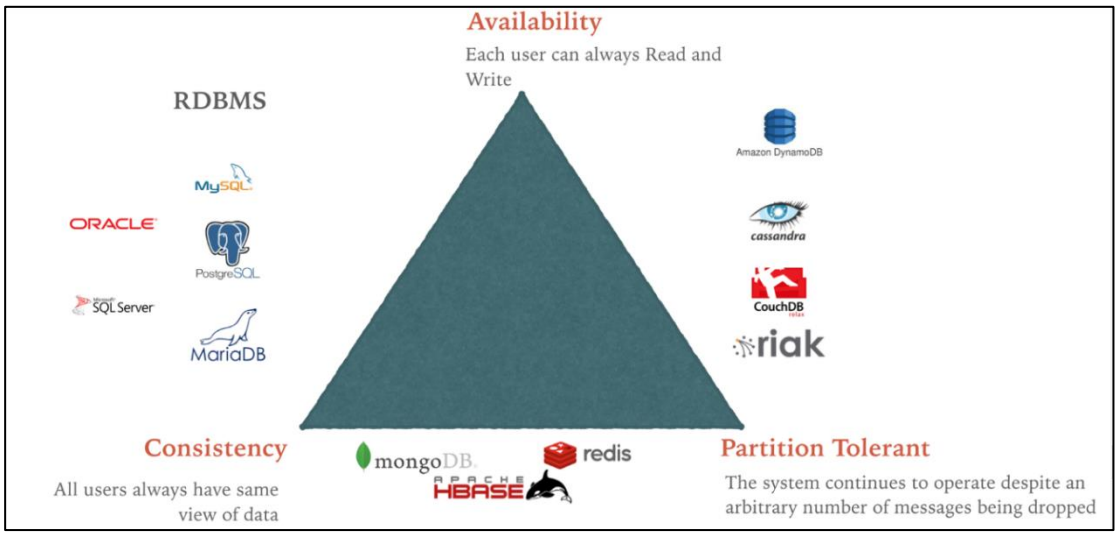
The theorem states that networked shared-data systems can only guarantee two of the following three properties:

- **Consistency (C):** All nodes see the same data at the same time. What you write you get to read.
- **Availability (A):** A guarantee that every request receives a response about whether it was successful or failed. Whether you want to read or write you will get some response back.
- **Partition tolerance (P)**: The system continues to operate despite arbitrary message loss or failure of part of the system. Irrespective of communication cut down among the nodes, system still works.

The CAP theorem categorizes systems into three categories:

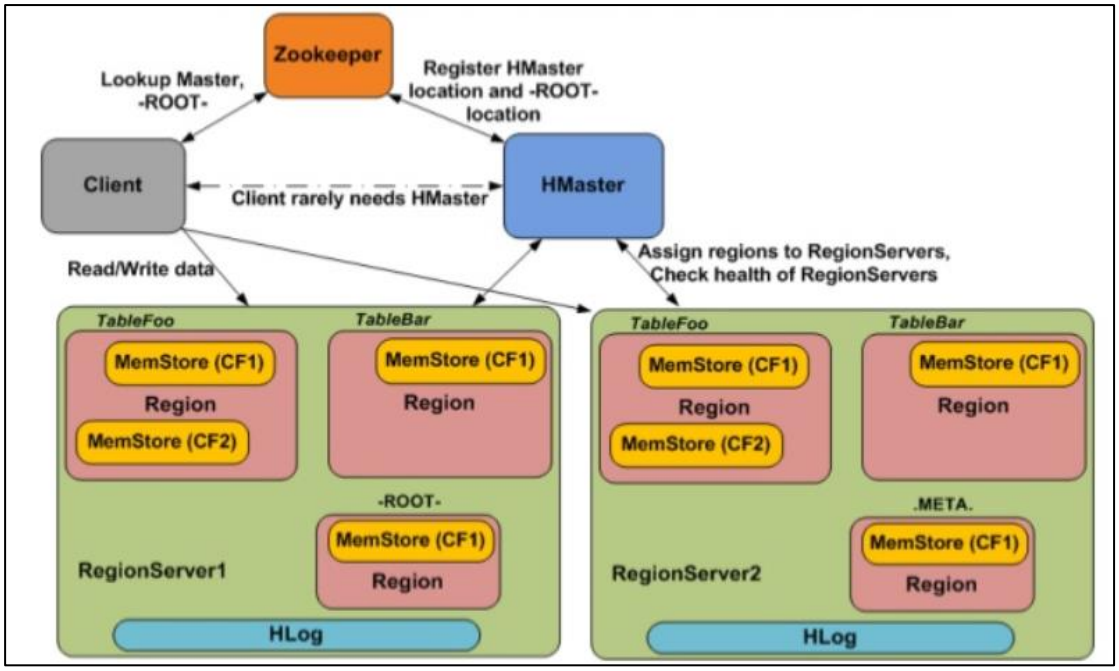
- **CP (Consistent and Partition Tolerant)** - At first glance, the CP category is confusing, i.e., a system that is consistent and partition tolerant but never available. CP is referring to a category of systems where availability is sacrificed only in the case of a network partition, I.e When there is no partition tolerance, system is not fully available. But the data is consistent.

- **CA (Consistent and Available)** - CA systems are consistent and available systems in the absence of any network partition. Often a single node's DB servers are categorized as CA systems. Single node DB servers do not need to deal with partition tolerance and are thus considered CA systems. The only hole in this theory is that single node DB systems are not a network of shared data systems and thus do not fall under the preview of CAP.
- **AP (Available and Partition Tolerant)** - These are systems that are available and partition tolerant but cannot guarantee consistency.



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 failover
- 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 quorum in order to connect with region servers and HMaster. In case of node failure within an HBase cluster, ZKquoram will trigger error messages and start repairing failed nodes.

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

- Establishing client communication with region servers.
- Tracking server failure and network partitions.
- Maintain Configuration Information
- Provides ephemeral nodes, which represent different region servers.

HBase vs RDBMS

Feature	RDBMS	HBase

Data Variety	Mainly for Structured data.	Used for Structured, Semi-Structured and Unstructured data
Data Storage	Average size data (GBS)	Use for large data set (Tbs and Pbs)
Querying	SQL Language	HQL (Hive Query Language) or Impala
Schema	Required on write (static schema)	Required on read (dynamic schema)
Speed	Reads are fast	Both reads and writes are fast
Cost	License	Free
Use Case	OLTP (Online transaction processing)	Analytics (Audio, video, logs etc), Data Discovery
Data Objects	Works on Relational Tables	Works on Key/Value Pair
Throughput	Low	High
Scalability	Vertical	Horizontal
Hardware Profile	High-End Servers	Commodity/Utility Hardware
Integrity	High (ACID)	Low

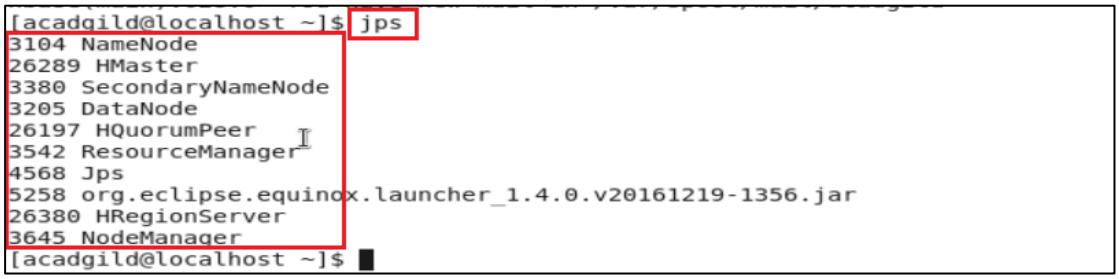
Task 2:

Execute blog present in below link

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

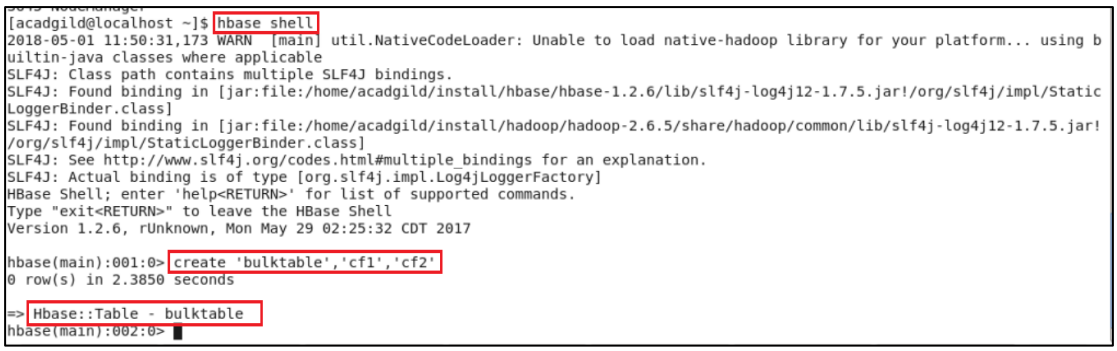
Before starting practice on TSV import, it is compulsory to start all the Hadoop and HBase daemons.

Running jps command confirms that both are running.



Step 1 : Inside Hbase shell give the following command to create table along with 2 column family.

create 'bulktable', 'cf1', 'cf2'



Logged into hbase shell and created a table called bulk table with 2 column families.

Step 2 and Step 3 and Step 4 : Created a tab separated file in the local file system, loaded data into the file.

Created a directory in HDFS and transferred this file into HDFS.

```
[acadgild@localhost assignments]$ vi bulk_data.tsv
[acadgild@localhost assignments]$ cat bulk_data.tsv
1       Amit    4
2       Girja   3
3       Jatin   5
4       Swathi  3
[acadgild@localhost assignments]$ hadoop fs -mkdir /user/acadgild/hbase/
18/05/01 12:14:29 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
[acadgild@localhost assignments]$ hadoop fs -put bulk_data.tsv /user/acadgild/hbase/
18/05/01 12:14:39 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
[acadgild@localhost assignments]$ hadoop fs -cat /user/acadgild/hbase/bulk_data.tsv
18/05/01 12:15:12 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
1       Amit    4
2       Girja   3
3       Jatin   5
4       Swathi  3
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost assignments]$
```

Step 5 : Run the importtsv command :

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

```
[acadgild@localhost hbase]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=HBASE_ROW_KEY,cf1:name,cf2:
exp bulktable /user/acadgild/hbase/bulk_data.tsv
2018-05-01 12:23:38,689 WARN  [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using b
uiltin-java 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/Static
LoggerBinder.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/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2018-05-01 12:23:39,823 INFO  [main] zookeeper.RecoverableZooKeeper: Process identifier=hconnection-0x6025e1b6 connecting to
ZooKeeper ensemble=localhost:2181
2018-05-01 12:23:39,850 INFO  [main] zookeeper.ZooKeeper: Client environment:zookeeper.version=3.4.6-1569965, built on 02/20/
2014 09:09 GMT
2018-05-01 12:23:39,851 INFO  [main] zookeeper.ZooKeeper: Client environment:host.name=localhost
2018-05-01 12:23:39,851 INFO  [main] zookeeper.ZooKeeper: Client environment:java.version=1.8.0_151
2018-05-01 12:23:39,855 INFO  [main] zookeeper.ZooKeeper: Client environment:java.vendor=Oracle Corporation
2018-05-01 12:23:39,855 INFO  [main] zookeeper.ZooKeeper: Client environment:java.home=/usr/java/jdk1.8.0_151/jre
2018-05-01 12:23:39,855 INFO  [main] zookeeper.ZooKeeper: Client environment:java.class.path=/home/acadgild/install/hbase/hba
```

Map job is 100% done and the import is done.

```
2018-05-01 12:24:21,460 INFO  [main] mapreduce.Job: Counters: 31
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=139477
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=160
                HDFS: Number of bytes written=0
                HDFS: Number of read operations=2
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=0
        Job Counters
                Launched map tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=10350
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=10350
                Total vcore-seconds taken by all map tasks=10350
                Total megabyte-seconds taken by all map tasks=10598400
        Map-Reduce Framework
                Map input records=4
                Map output records=4
                Input split bytes=120
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=111
                CPU time spent (ms)=2320
                Physical memory (bytes) snapshot=122368000
                Virtual memory (bytes) snapshot=2076086272
                Total committed heap usage (bytes)=48758784
        ImportTsv
                Bad Lines=0
        File Input Format Counters
                Bytes Read=40
        File Output Format Counters
                Bytes Written=0
You have new mail in /var/spool/mail/acadgild
```

Step 6 : Once the mapreduce job is done, we login into hbase shell and run a scan command on the table to see the contents of the table.

```
hbase(main):001:0> scan 'bulktable'
ROW                             COLUMN+CELL
 1                              column=cf1:name, timestamp=1525157618606, value=Amit
 1                              column=cf2:exp, timestamp=1525157618606, value=4
 2                              column=cf1:name, timestamp=1525157618606, value=Girja
 2                              column=cf2:exp, timestamp=1525157618606, value=3
 3                              column=cf1:name, timestamp=1525157618606, value=Jatin
 3                              column=cf2:exp, timestamp=1525157618606, value=5
 4                              column=cf1:name, timestamp=1525157618606, value=Swathi
 4                              column=cf2:exp, timestamp=1525157618606, value=3
4 row(s) in 0.7960 seconds

hbase(main):002:0>
```

The contents of all the columns with 4 rows are displayed. Thus the data is imported from HDFS file to Hbase.