Assignment 11

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

Nosql Databases:

NoSQL is a new breed of database management systems that fundamentally differ from relational database systems. These databases do not require tables with a fixed set of columns, avoid JOINs and typically support horizontal scaling. They are also referred to as structured storage.

Types of Nosql Databases:

Listing below few of the widely used Databases which uses different storage mechanism

- **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.

Other NoSql DB's that are used are:

1. Mango DB
2. Redis

3. Couch DB
4. REVENDB:
5. MemcacheDB:
6. Riak:
7. Neo4j:
8. HBASE
9. Perst
10. HyperGraphDB
11. Cassandra
12. Voldemort
13. Terrastore
14. NeoDatis
15. MyOODB
16. OrientDB
17. InfoGrid
18. Db4objects

● Types of Nosql Databases:

There are 4 basic types of NoSQL databases:

1. **Key-Value Store** – It has a Big Hash Table of keys & values {Example- Riak, Amazon S3 (Dynamo)}
2. **Document-based Store- It** stores documents made up of tagged elements. {Example- CouchDB}
3. **Column-based Store-** Each storage block contains data from only one column, {Example- HBase, Cassandra}
4. **Graph-based**-A network database that uses edges and nodes to represent and store data. {Example- Neo4J}

1. Key Value Store NoSQL Database

The schema-less format of a key value database like Riak is just about what you need for your storage needs. The key can be synthetic or auto-generated while the value can be String, JSON, BLOB (basic large object) etc.

The key value type basically, uses a hash table in which there exists a unique key and a pointer to a particular item of data. A bucket is a logical group of keys – but they don't physically group the data. There can be identical keys in different buckets.

Performance is enhanced to a great degree because of the cache mechanisms that accompany the mappings. To read a value you need to know both the key and the bucket because the real key is a hash (Bucket+ Key).

There is no complexity around the Key Value Store database model as it can be implemented in a breeze. Not an ideal method if you are only looking to just update part of a value or query the database.

When we try and reflect back on the CAP theorem, it becomes quite clear that key value stores are great around the Availability and Partition aspects but definitely lack in Consistency.

Example: Consider the data subset represented in the following table. Here the key is the name of the 3Pillar country name, while the value is a list of addresses of 3Pilllar centers in that country.

Key	**Value**
“India”	{“B-25, Sector-58, Noida, India – 201301”
“Romania”	{“IMPS Moara Business Center, Buftea No. 1, Cluj-Napoca, 400606″,City Business Center, Coriolan Brediceanu No. 10, Building B, Timisoara, 300011”}

The key can be synthetic or auto-generated while the value can be String, JSON, BLOB (basic large object) etc.

This key/value type database allow clients to read and write values using a key as follows:

- Get(key), returns the value associated with the provided key.
- Put(key, value), associates the value with the key.
- Multi-get(key1, key2, .., keyN), returns the list of values associated with the list of keys.
- Delete(key), removes the entry for the key from the data store.

While Key/value type database seems helpful in some cases, but it has some weaknesses as well. One, is that the model will not provide any kind of traditional database capabilities (such as atomicity of transactions, or consistency when multiple

transactions are executed simultaneously). Such capabilities must be provided by the application itself.

Secondly, as the volume of data increases, maintaining unique values as keys may become more difficult; addressing this issue requires the introduction of some complexity in generating character strings that will remain unique among an extremely large set of keys.

- Riak and Amazon's Dynamo are the most popular key-value store NoSQL databases.

2. **Document Store NoSQL Database**

The data which is a collection of key value pairs is compressed as a document store quite similar to a key-value store, but the only difference is that the values stored (referred to as "documents") provide some structure and encoding of the managed data. XML, JSON (Java Script Object Notation), BSON (which is a binary encoding of JSON objects) are some common standard encodings.

The following example shows data values collected as a "document" representing the names of specific retail stores. Note that while the three examples all represent locations, the representative models are different.

{officeName:"3Pillar Noida",
{Street: "B-25, City:"Noida", State:"UP", Pincode:"201301"}
}
{officeName:"3Pillar Timisoara",
{Boulevard:"Coriolan Brediceanu No. 10", Block:"B, Ist Floor", City: "Timisoara", Pincode: 300011"}
}

One key difference between a key-value store and a document store is that the latter embeds attribute metadata associated with stored content, which essentially provides a way to query the data based on the contents. For example, in the above example, one could search for all documents in which "City" is "Noida" that would deliver a result set containing all documents associated with any "3Pillar Office" that is in that particular city.

Apache CouchDB is an example of a document store. CouchDB uses JSON to store data, JavaScript as its query language using MapReduce and HTTP for an API. Data

and relationships are not stored in tables as is a norm with conventional relational databases but in fact are a collection of independent documents.
Couchbase and MongoDB are the most popular document based databases.

3. Column Store NoSQL Database–

In column-oriented NoSQL database, data is stored in cells grouped in columns of data rather than as rows of data. Columns are logically grouped into column families. Column families can contain a virtually unlimited number of columns that can be created at runtime or the definition of the schema. Read and write is done using columns rather than rows.
In comparison, most relational DBMS store data in rows, the benefit of storing data in columns, is 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 disk while Columnar databases store all the cells corresponding to a column as a continuous disk entry thus makes the search/access faster.
For example: To query the titles from a bunch of a million articles will be a painstaking task while using relational databases as it will go over each location to get item titles. On the other hand, with just one disk access, title of all the items can be obtained.
Data Model

- ColumnFamily: ColumnFamily is a single structure that can group Columns and SuperColumns with ease.
- Key: the permanent name of the record. Keys have different numbers of columns, so the database can scale in an irregular way.
- Keyspace: This defines the outermost level of an organization, typically the name of the application. For example, '3PillarDataBase' (database name).
- Column: It has an ordered list of elements tuple with a name and a value defined.

The best known examples are Google's BigTable and HBase & Cassandra that were inspired from BigTable.
BigTable, for instance is a high performance, compressed and proprietary data storage system owned by Google. It has the following attributes:

- Sparse – some cells can be empty
- Distributed – data is partitioned across many hosts
- Persistent – stored to disk
- Multidimensional – more than 1 dimension
- Map – key and value
- Sorted – maps are generally not sorted but this one is

A 2-dimensional table comprising of rows and columns is part of the relational database system.

City	Pincode	Strength	Project
Noida	201301	250	20

Timisoara	300011	150	10

For above RDBMS table a BigTable map can be visualized as shown below.

{
3PillarNoida: {
city: Noida
pincode: 201301
},
details: {
strength: 250
projects: 20
}
}
3PillarTimisoara: {
address: {
city: Timisoara
pincode: 300011
},
details: {
strength: 150
projects: 10
}
}

The outermost keys 3PillarNoida and 3PillarTimisoara are analogues to rows.

- ‘address’ and ‘details’ are called **column families**.
- The column-family ‘address’ has **columns** ‘city’ and ‘pincode’.
- The column-family details’ has **columns** ‘strength’ and ‘projects’.

Columns can be referenced using CloumnFamily.

4. Graph Base NoSQL Database

In a Graph Base NoSQL Database, you will not find the rigid format of SQL or the tables and columns representation, a flexible graphical representation is instead used which is perfect to address scalability concerns. Graph structures are used with edges, nodes and properties which provides index-free adjacency. Data can be easily transformed from one model to the other using a Graph Base NoSQL database.

- These databases that uses edges and nodes to represent and store data.
- These nodes are organised by some relationships with one another, which is represented by edges between the nodes.
- Both the nodes and the relationships have some defined properties.

The following are some of the features of the graph based database, which are explained on the basis of the example below:

Labeled, directed, attributed multi-graph : The graphs contains the nodes which are labelled properly with some properties and these nodes have some relationship with one another which is shown by the directional edges. For example: in the following representation, “Alice knows Bob” is shown by an edge that also has some properties. While relational database models can replicate the graphical ones, the edge would require a join which is a costly proposition.

InfoGrid and Infinite Graph are the most popular graph based databases. InfoGrid allows the connection of as many edges (Relationships) and nodes (MeshObjects), making it easier to represent hyperlinked and complex set of information.

There are two kinds of GraphDatabase offered by InfoGrid, these include the following:

MeshBase– It is a perfect option where standalone deployment is required.

NetMeshBase – It is ideally suited for large distributed graphs and has additional capabilities to communicate with other similar NetMeshbase.

This concludes the second post exemplifying the value in a NoSQL implementation

● CAP Theorem :

• Consistency - This means that the data in the database remains consistent after the execution of an operation. For example after an update operation, all clients see the same data.

• Availability - This means that the system is always on (service guarantee availability), no downtime.

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

● HBase Architecture

HBase is composed of three types of servers in a master slave type of architecture.

• Region servers serve data for reads and writes

• HBase Master process handles the Region assignment, DDL (create, delete tables) operations
• Zookeeper 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
• The NameNode maintains metadata information for all the physical data blocks that comprise the files
• Region assignment, DDL (create, delete tables) operations are handled by the HBase Master.

A master is responsible for:
• Coordinating the region servers
• Assigning regions on startup
• Re-assigning regions for recovery or load balancing
• Monitoring all RegionServer instances in the cluster (listens for notifications from zookeeper) Admin functions
• Interface for creating, deleting, updating tables

There is a special HBase Catalog table called the META table, which holds the location of the regions in the cluster. ZooKeeper stores the location of the META table The client gets the Region server that hosts the META table from ZooKeeper. The client will query the .META. server to get the region server corresponding to the row key it wants to access. The client caches this information along with the META table location It will get the row from the corresponding Region Server. For future reads, the client uses the cache to retrieve the META location and previously read row keys. Over time, it does not need to query the META table, unless there is a miss because a region has moved; then it will re-query and update the cache.

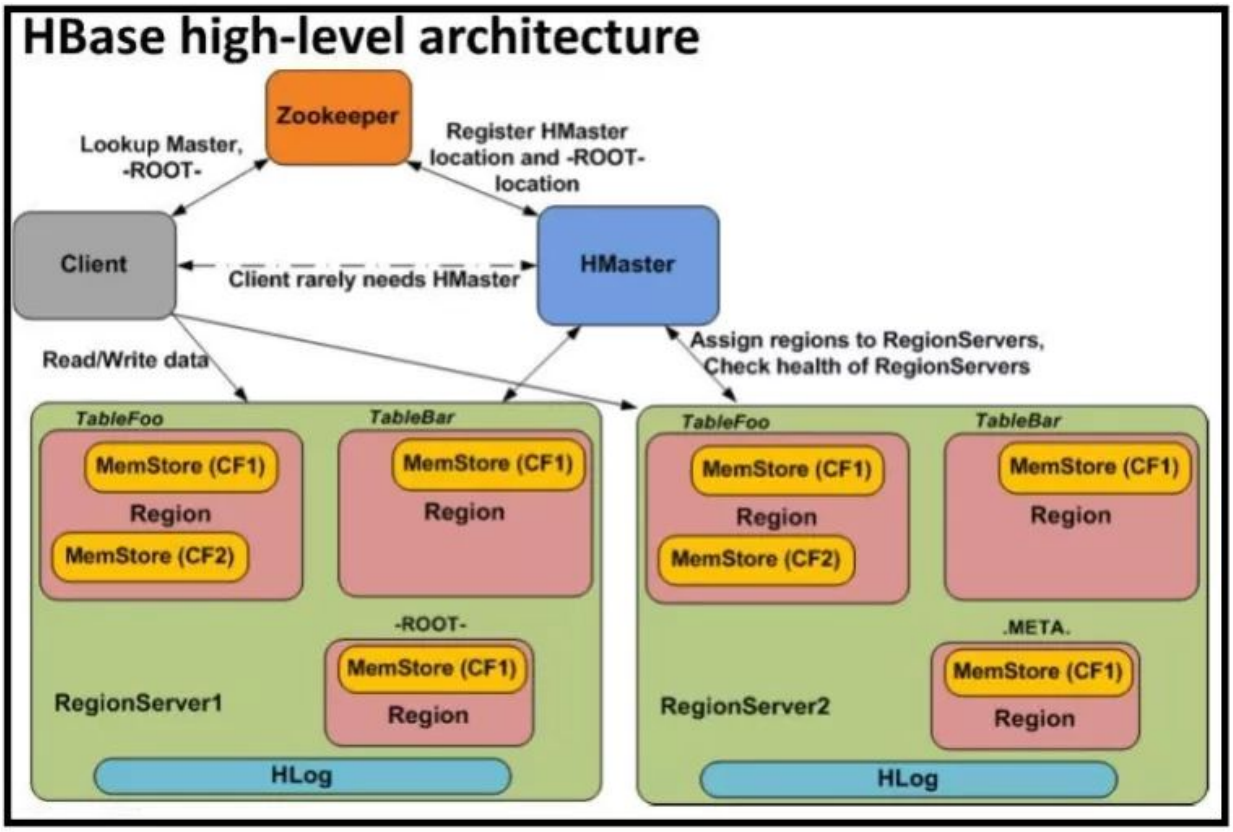
The META table is an HBase table that keeps a list of all regions in the system. .META. table is like a b tree
• The .META. table structure is as follows:
Key: region start key, region id Values: RegionServer

Region Server runs on an HDFS data node and has the following components:
WAL • Write Ahead Log is a file on the distributed file system. The WAL is used to store new data that hasn't yet been persisted to permanent storage; it is used for recovery in the case of failure. BlockCache
• It is the read cache. It stores frequently read data in memory. Least Recently Used data is evicted when full. MemStore

• It is the write cache. It stores new data which has not yet been written to disk. It is sorted before writing to disk. There is one MemStore per column family per region.
Hfiles
• They store the rows as sorted KeyValues on disk.
. Initially there is one region per table.
• When a region grows too large, it splits into two child regions
• Both child regions, representing one-half of the original region, are opened in parallel on the same Region server, and then the split is reported to the HMaster.
• For load balancing reasons, the HMaster may schedule for new regions to be moved off to other servers.

HBase provides the following benefits:
• Strong consistency model- When a write returns, all readers will see same value
• Scales automatically- Regions split when data grows too large- Uses HDFS to spread and replicate data
• Built-in recovery- Using Write Ahead Log (similar to journaling on file system)
• Integrated with Hadoop- MapReduce on HBase is straightforward

Snapshot showing Hbase architecture:



- **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

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.

However, Hadoop brings a lot to the table too, offering massive scaling capability in terms of processing power and storage at costs which are relatively much lower when compared to the sky high costs of an RDBMS. Hadoop also comes packed with amazing parallel processing capabilities, with one being able to run jobs in parallel to execute number crunching in large volumes.

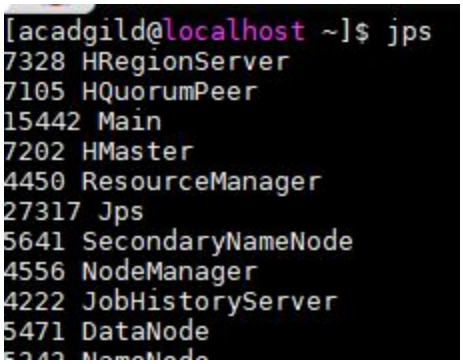
When it comes to deciding the efficacy of traditional databases in terms of working with unstructured data, it is not so simple. Several applications that are created utilizing traditional RDBMS which use a lot of unstructured data or video files or PDFs seem to work well, as vouched for by market experts. Usually RDBMS is meant to handle large chunks of data in its cache for accelerated processing while simultaneously maintaining read consistency across sessions. Hadoop does a much better job of utilizing the memory cache for data processing without providing any other aspects such as read consistency. Hive SQL has always known to be several times slower than SQL which can be run in traditional databases. Users who go in thinking that SQL in Hive is speedier than in databases will be inn for a huge let-down since it will not scale to any

extent whatsoever for complicated analytics. While Hadoop excels at parallel processing problems such as finding a group of keywords in large sets of documents, RDBMS implementations will be much faster for data sets which are comparable.

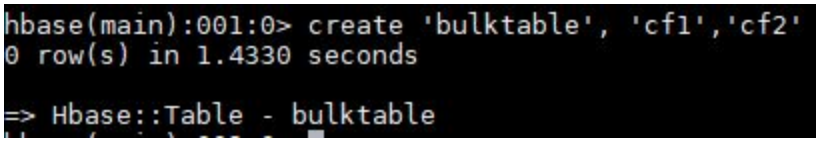
Task2 Execute blog present in below link :

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

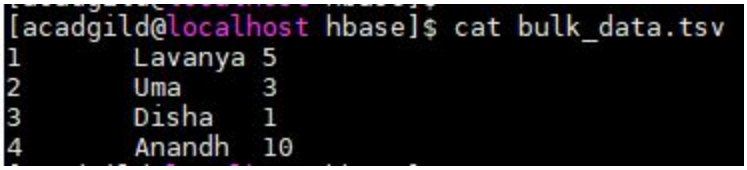
Starting all the hadoop and hbase related daemons:



Creating a new empty table in hbase to import data from hdfs:



Created a new tsv file having the below data:



Moved the above file to hdfs:

```
[acadgild@localhost hbase]$ hadoop fs -cat /hbase/bulk_data1.tsv
18/05/22 00:28:19 WARN util.NativeCodeLoader: Unable to load nativ
s where applicable
1       Lavanya 5
2       Uma     3
3       Disha   1
4       Anandh  10
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost hbase]$
```

Command used for importing:

----

hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=HBASE_ROW_KEY,cf1:name,cf2:exp bulktable /hbase/bulk_data1.tsv

Output after importing:

```
hbase(main):002:0> list
TABLE
TRANSACTIONS
bulktable
clicks
3 row(s) in 0.0560 seconds

=> ["TRANSACTIONS", "bulktable", "clicks"]
hbase(main):003:0> scan "bulktable"
ROW                             COLUMN+CELL
 1                              column=cf1:name, timestamp=1526928617739, value=Lavanya
 1                              column=cf2:exp, timestamp=1526928617739, value=5
 2                              column=cf1:name, timestamp=1526928617739, value=Uma
 2                              column=cf2:exp, timestamp=1526928617739, value=3
 3                              column=cf1:name, timestamp=1526928617739, value=Disha
 3                              column=cf2:exp, timestamp=1526928617739, value=1
 4                              column=cf1:name, timestamp=1526928617739, value=Anandh
 4                              column=cf2:exp, timestamp=1526928617739, value=10
4 row(s) in 0.0450 seconds
```