**BIG DATA HADOOP AND SPARK DEVLOPMENT**

**ASSIGNMENT 11**

**Table of Contents:**

1. **Introduction 1**
2. **Objective 1**
3. **Problem Statement 1**
4. **Expected Output**

* **Task 1 3**
* **Task 2**

**BIG DATA HADOOPAND SPARK DEVELOPMENT**

1. **Introduction**

In this assignment, the given tasks are performed and Output of the tasks are recorded in the form of Screenshots.

1. **Objective**

This Assignment consolidates the deeper understanding of the Session – 11

1. **Problem Statement**

* **Task 1**

Explain the below concepts with an example in brief.

● Nosql Databases

● Types of Nosql Databases

● CAP Theorem

● HBase Architecture

● HBase vs RDBMS

* **Task 2**

Execute blog present in below link

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

1. **Expected Output**

* **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 encompasses a wide variety of different database technologies that were developed in response to the demands presented in building modern applications:

• Developers are working with applications that create massive volumes of new, rapidly changing data types — structured, semi-structured, unstructured and polymorphic data.

• Long gone is the twelve-to-eighteen-month waterfall development cycle. Now small teams work in agile sprints, iterating quickly and pushing code every week or two, some even multiple times every day.

• Applications that once served a finite audience are now delivered as services that must be always-on, accessible from many different devices and scaled globally to millions of users.

• Organizations are now turning to scale-out architectures using open source software, commodity servers and cloud computing instead of large monolithic servers and storage infrastructure.

● **Types of Nosql Databases**

• 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 Graph.

• 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.

● **CAP Theorem**

• Consistency means that data is the same across the cluster, so you can read or write from/to any node and get the same data.

• Availability means the ability to access the cluster even if a node in the cluster goes down.

• Partition tolerance means that the cluster continues to function even if there is a "partition" (communication break) between two nodes (both nodes are up, but can't communicate). In order to get both availability and partition tolerance, you have to give up consistency. Consider if you have two nodes, X and Y, in a master-master setup. Now, there is a break between network communication between X and Y, so they can't sync updates. At this point you can either:

A) Allow the nodes to get out of sync (giving up consistency), or

B) Consider the cluster to be "down" (giving up availability) All the combinations available are:

• CA - data is consistent between all nodes - as long as all nodes are online - and you can read/write from any node and be sure that the data is the same, but if you ever develop a partition between nodes, the data will be out of sync (and won't re-sync once the partition is resolved).

• CP - data is consistent between all nodes, and maintains partition tolerance (preventing data desync) by becoming unavailable when a node goes down.

• AP - nodes remain online even if they can't communicate with each other and will resync data once the partition is resolved, but you aren't guaranteed that all nodes will have the same data (either during or after the partition)

● **HBase Architecture**

As we know, HBase is a column-oriented NoSQL database. Although it looks similar to a relational database which contains rows and columns, but it is not a relational database. Relational databases are row oriented while HBase is column-oriented. In a column-oriented database, all the column values are stored together like first column values will be stored together, then the second column values will be stored together and data in other columns are stored in a similar manner.

• When the amount of data is very huge, like in terms of petabytes or Exabyte, we use column-oriented approach, because the data of a single column is stored together and can be accessed faster.

• While row-oriented approach comparatively handles less number of rows and columns efficiently, as row-oriented database stores data is a structured format.

• When we need to process and analyze a large set of semi-structured or unstructured data, we use column oriented approach. Such as applications dealing with Online Analytical Processing like data mining, data warehousing, applications including analytics, etc.

• Whereas, Online Transactional Processing such as banking and finance domains which handle structured data and require transactional properties (ACID properties) use row oriented approach.

**HBase tables has following components:**

• **Tables**:

Data is stored in a table format in HBase. But here tables are in column-oriented format.

• **Row** **Key**:

Row keys are used to search records which make searches fast.

• **Column** **Families**:

Various columns are combined in a column family. These column families are stored together which makes the searching process faster because data belonging to same column family can be accessed together in a single seek.

• **Column** **Qualifiers**:

Each column’s name is known as its column qualifier.

• **Cell**:

Data is stored in cells. The data is dumped into cells which are specifically identified by rowkey and column qualifiers.

• **Timestamp**:

Timestamp is a combination of date and time. Whenever data is stored, it is stored with its timestamp. This makes easy to search for a particular version of data.

**Components of HBase Architecture**

**HBase has three major components:**

* HMaster Server
* HBase Region Server
* 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.
* **HBase 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

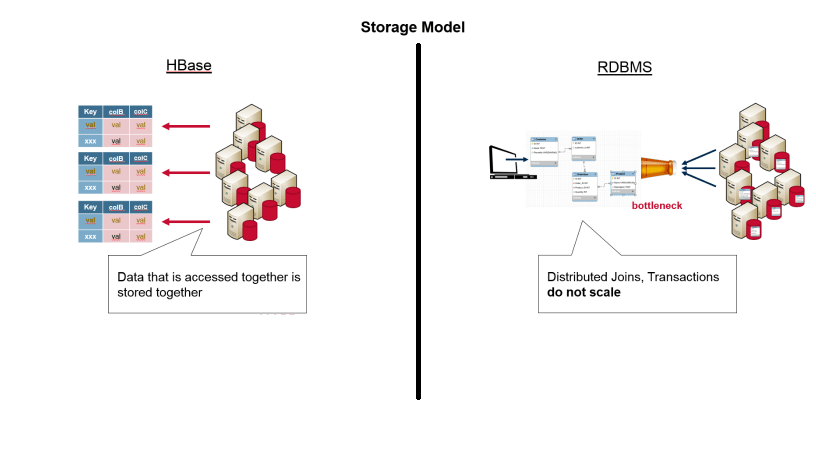
* + Establishing client communication with region servers.
  + Tracking server failure and network partitions.
  + Maintain Configuration Information
  + Provides ephemeral nodes, which represent different region servers.

Understanding the fundamental of HBase architecture is easy but running HBase on top of HDFS in production is challenging when it comes to monitoring compactions, row key designs manual splitting, etc.

**● HBase vs RDBMS**

HBase and other column-oriented database are often compared to more traditional and popular relational database or RDBMS.

|  |  |
| --- | --- |
| **H** **Base** | **RDBMS** |
| 1. Column-oriented | 1. Row-oriented(mostly) |
| 2. Flexible schema, add columns on the Fly | 2. Fixed schema |
| 3. Good with sparse tables. | 3. Not optimized for sparse tables. |
| 4. No query language | 4. SQL |
| 5. Wide tables | 5. Narrow tables |
| 6. Joins using MR – not optimized | 6. optimized for Joins(small, fast ones) |
| 7. Tight – Integration with MR | 7. Not really |
| 8. De-normalize your data. | 8. Normalize as you can |
| 9. Horizontal scalability-just add hard ware. | 9. Hard to share and scale. |
| 10. Consistent | 10. Consistent |
| 11. No transactions. | 11. transactional |
| 12. Good for semi-structured data as well as structured data. | 12. Good for structured data. |



* **Task 2**

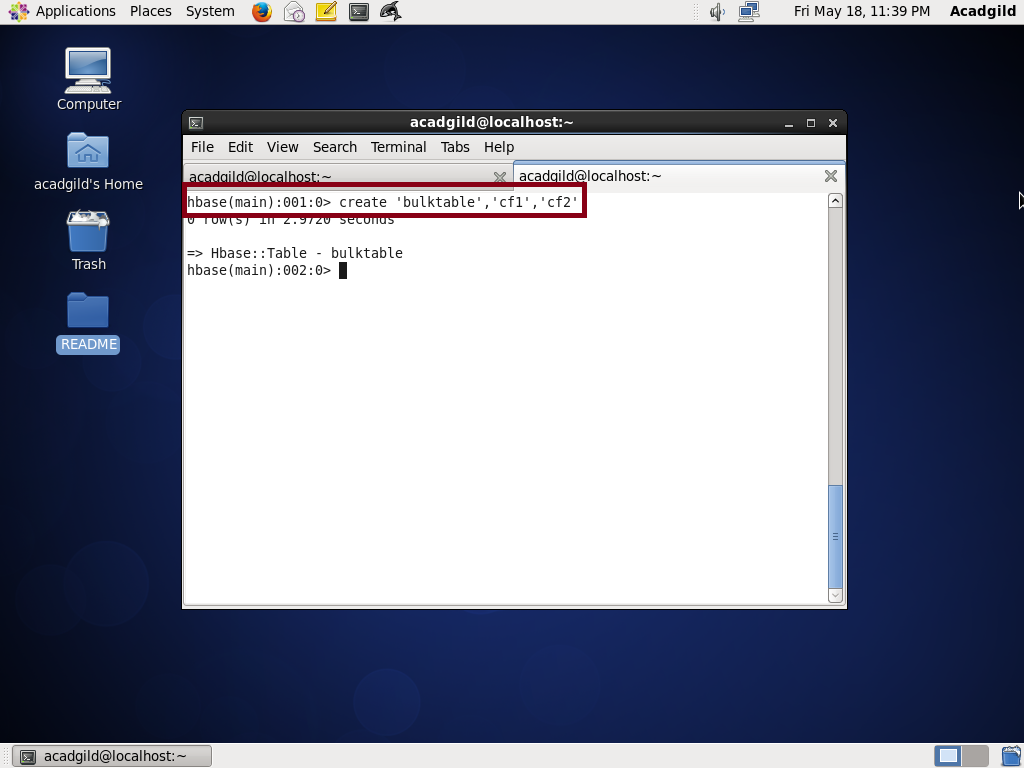
Execute blog present in below link

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

ImportTsv takes data from HDFS into HBase via Puts.   
Find below the syntax used to load data via Puts (i.e., non-bulk loading):  
**$ bin/hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=a,b,c <tablename> <hdfs-inputdir>**  
Data inside HDFS is loaded into HBase.

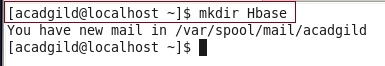
#### Step1:

Inside Hbase shell give the following command to create table along with 2 column family.  
**create ‘bulktable’, ‘cf1’, ‘cf2’**



#### Step2 :

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.



Now move to the directory where we will keep our data.

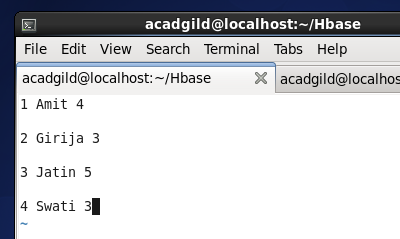


#### Step3:

Create a file inside the HBase directory named bulk\_data.tsv with tab separated data inside using below command in terminal.



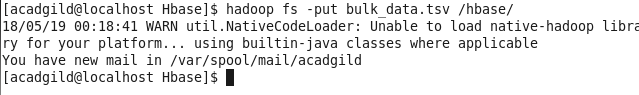
**Put these data in,**  
1    Amit 4  
2    Girija  3  
3    Jatin   5  
4    Swati   3



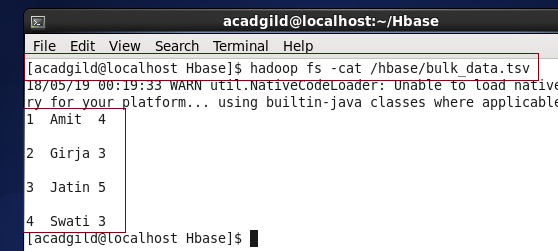
Once created save the file using **esc** + **:wq** + **enter**

#### Step4:

Our data should be present in HDFS while performing the import task to Hbase.  
In real time projects, the data will already be present inside HDFS.  
Here for our learning purpose, we copy the data inside HDFS using below commands in terminal.  
Command: **hadoop fs -mkdir /hbase**



Command:  
**hadoop fs -cat /hbase/bulk\_data.tsv**



#### Step5:

After the data is present now in HDFS.In terminal, we give the following command along with arguments <tablename> and <path of data in HDFS>

**Command:**  
**hbase org.apache.hadoop.hbase.mapreduce.ImportTsv –**  
**Dimporttsv.columns=HBASE\_ROW\_KEY,cf1:name,cf2:exp bulktable /hbase/bulk\_data.tsv**