Assignment 11 : ADVANCE HBASE Assignment Problems

Problem Statement

Task1

Explain the below concepts with an example in brief.

- **Nosql Databases :**

Explaination :

A NoSQL (often interpreted as Not only SQL) database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.

Motivations for this approach include simplicity of design, horizontal scaling, and finer control over availability.

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.

Relational databases were not designed to cope with the scale and agility challenges that face modern applications, nor were they built to take advantage of the commodity storage and processing power

available today.

The Benefits of NoSQL

When compared to relational databases, NoSQL databases are more scalable and provide superior performance, and their data model addresses several issues that the relational model is not designed to address:

- Large volumes of rapidly changing structured, semi-structured, and unstructured data
- Agile sprints, quick schema iteration, and frequent code pushes
- Object-oriented programming that is easy to use and flexible
- Geographically distributed scale-out architecture instead of expensive, monolithic architecture

The Structured Query Language used by traditional databases provides a uniform way to communicate with the server when storing and retrieving data. SQL syntax is highly standardized, so while individual databases may handle certain operations differently (for e.g Window functions), the basics remain the same.

By contrast, each NoSQL database tends to have its own syntax for querying and managing the data. CouchDB, for instance, uses requests in the form of JSON, sent via HTTP, to create or retrieve documents from its database. MongoDB sends JSON objects over a binary protocol, by way of a command-line interface or a language library.

Some NoSQL products can use SQL-like syntax to work with data, but only to a limited extent. For example, Apache Cassandra, a column store database, has its own SQL-like language, the Cassandra Query Language (CQL). Some of the CQL syntax is straight out of the SQL playbook, like the SELECT or INSERT keywords. But there is no way to perform a JOIN or subquery in Cassandra, and thus the related keywords don’t exist in CQL.

- **Types of Nosql Databases :**

Explaination :

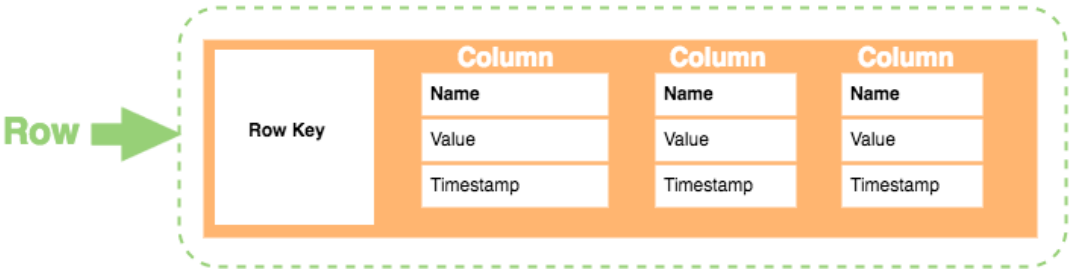
NO SQL can be deployed in four different manners :

1. Columnar Databases :
Reads and writes columns of data rather than the rows. Each column is comparable to a container in RDBMS where a Key defines a row and single row has multiple columns.

Columnar database can highly compressed the data. The compression permits columnar operations like MIN, MAX, SUM, COUNT and AVG to be performed very rapidly.

Column-based DBMSs is self-indexing, it uses less disk space than a relational database management system (RDBMS) containing the same data. As the use of in-memory analytics increases, however, the relative benefits of row-oriented vs. column oriented databases may become less important. In-memory analytics is not concerned with efficiently reading and writing data to a hard disk. Instead, it allows data to be queried in random access memory (RAM).

Columns store databases use a concept called a keyspace. A keyspace is kind of like a schema in the relational model. The keyspace contains all the column families (kind of like tables in the relational model), which contain rows, which contain columns.



Row Key. Each row has a unique key, which is a unique identifier for that row.

- **Column**. Each column contains a name, a value, and timestamp.
- **Name**. This is the name of the name/value pair.
- **Value**. This is the value of the name/value pair.
- **Timestamp**. This provides the date and time that the data was inserted. This can be used to determine the most recent version of data.

Benefits of Column Store Databases

- **Compression**. Column stores are very efficient at data compression and/or partitioning.
- **Aggregation queries**. Due to their structure, columnar databases perform particularly well with aggregation queries (such as SUM, COUNT, AVG, etc).
- **Scalability**. Columnar databases are very scalable. They are well suited to massively parallel processing (MPP), which involves having data spread across a large cluster of machines – often thousands of machines.
- **Fast to load and query**. Columnar stores can be loaded extremely fast. A billion row table could be loaded within a few seconds. You can start querying and analysing almost immediately.

Examples of Column Store DBMSs

BigTable
Cassandra
HBase
Vertica

2. Document Databases :

These databases store and retrieve semi-structured data in the format of documents such as XML, JSON, etc. Some of the popular document databases like MongoDB provide a rich query language for ease of access and smooth transition of data models.

A document database, also called a document store or document-oriented database, is a subset of a type of **NoSQL database**. Some document stores may also be **key-value databases**. A document database is used for storing, retrieving, and managing semi-structured data. Unlike traditional relational databases, the data model in a document database is not structured in a table format of rows and columns. The schema can vary, providing far more flexibility for data modeling than relational databases.

A document database uses documents as the structure for storage and queries. In this case, the term “document” may refer to a Microsoft Word or PDF document but is commonly a

block of XML or JSON. Instead of columns with names and data types that are used in a relational database, a document contains a description of the data type and the value for that description. Each document can have the same or different structure. To add additional types of data to a document database, there is no need to modify the entire database schema as there is with a relational database. Data can simply be added by adding objects to the database.

Documents are grouped into “collections,” which serve a similar purpose to a relational table. A document database provides a query mechanism to search collections for documents with particular attributes.

BENEFITS OF A DOCUMENT DATABASE

- **Flexible data modeling:** As web, mobile, social, and IoT-based applications change the nature of application data models, document databases eliminate the need to force-fit relational data models to support new types of application data models.
- **Fast write performance:** Unlike traditional relational databases, some document databases prioritize write availability over strict data consistency. This ensures that writes will always be fast even if a failure in one portion of the hardware or network results in a small delay in data replication and consistency across the environment.
- **Fast query performance:** Many document databases have powerful query engines and indexing features that provide fast and efficient querying capabilities.

Examples of Document Database:

MongoDb

CouchDb

PostgreSQL

3. Graph Databases :

Stores data as entities and relations between them allowing faster traversal and joining operations to be performed. However these graphs can be built using SQL as well as NoSQL databases.

In computing, a **graph database** is a database that uses graph structure for semantic queries with nodes, edges and properties to represent and store data. A key concept of the system is the *graph* (or *edge* or *relationship*), which directly relates data items in the store. The relationships allow data in the store to be linked together directly, and in many cases retrieved with one operation.
This contrasts with relational databases that, with the aid of relational database management systems, permit managing the data without imposing implementation aspects like physical record chains; for example, links between data are stored in the database itself at the logical level, and relational algebra operations (e.g. *join*) can be used to manipulate and return related data in the relevant logical format. The execution of relational queries is possible with the aid of the database management systems at the physical level (e.g. using indexes), which permits boosting performance without modifying the logical structure of the database.

Graph databases are based on graph theory, and employ nodes, edges, and properties.

- ***Nodes*** represent entities such as people, businesses, accounts, or any other item to be tracked. They are roughly the equivalent of the *record*, *relation*, or *row* in a relational database, or the *document* in a document database.

- ***Edges***, also termed *graphs* or *relationships*, are the lines that connect nodes to other nodes; they represent the relationship between them. Meaningful patterns emerge when examining the connections and interconnections of nodes, properties, and edges. Edges are the key concept in graph databases, representing an abstraction that is not directly implemented in other systems.

Graph databases are a powerful tool for graph-like queries. For example, computing the shortest path between two nodes in the graph. Other graph-like queries can be performed over a graph database in a natural way.

Examples of Graph Databases

Graph Base

GraphDB

Neo4j

4. In-Memory Key:

Value Stores- Suitable for read-heavy workloads and compute-intensive workloads, these databases store critical data in memory which in turn improves the performance of the systems.

An **in-memory database** is a database management system that primarily relies on main

memory for computer data storage. It is contrasted with database management systems that employ a disk storage mechanism. In-memory databases are faster than disk-optimized databases because disk access is slower than memory access, the internal optimization algorithms are simpler and execute fewer CPU instructions. Accessing data in memory eliminates seek time when querying the data, which provides faster and more predictable performance than disk.

Examples of In-Memory Key Database:

Apache Ignite
ArangoDB
SQLite

- **CAP Theorem :**

Explaination :

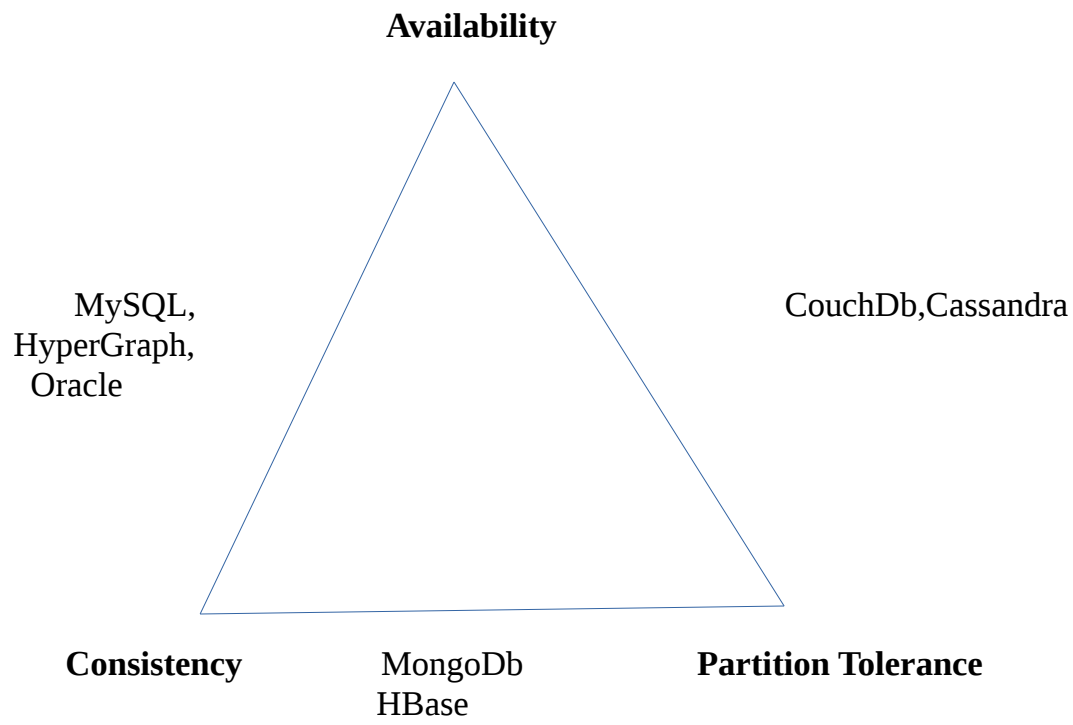
CAP Theorem – Consistency, Availability, Partition Tolerance

Consistency: all nodes see the same data at the same time

Availability: a guarantee that every request receives a response about whether it was successful or failed

Partition tolerance: the system continues to operate despite arbitrary message loss
The theorem states that you cannot simultaneously have all three; you must make trade offs among them.

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.

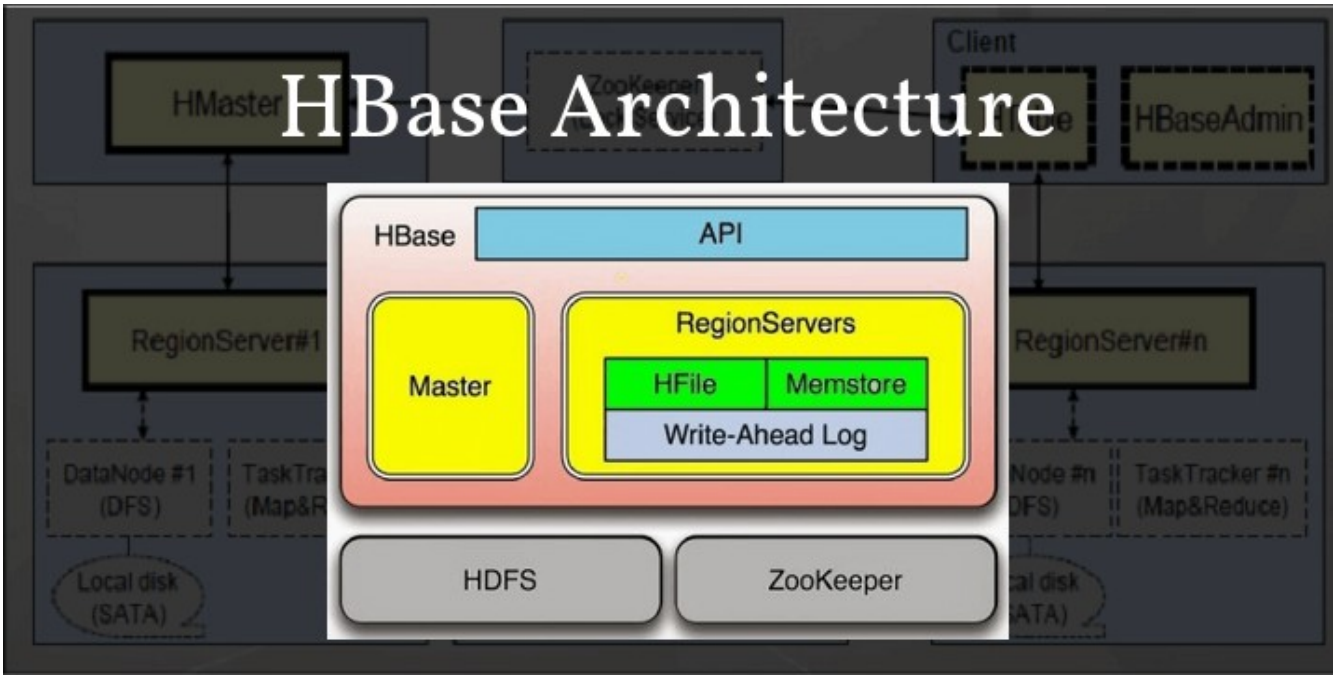


For example if you consider Riak a distributed key-value database. There are essentially three variables
r, w, n where
• r=number of nodes that should respond to a read request before its considered successful.
• w=number of nodes that should respond to a write request before its considered successful.
• n=number of nodes where the data is replicated aka replication factor.

In a Riak cluster with 5 nodes, we can tweak the r,w,n values to make the system very consistent by setting r=5 and w=5 but now we have made the cluster susceptible to network partitions since any write will not be considered successful when any node is not responding. We can make the same cluster highly available for writes or reads by setting r=1 and w=1 but now consistency can be compromised since some nodes may not have the latest copy of the data. The CAP theorem states that if you get a network partition, you have to trade off availability of data versus consistency of data. Durability can also be traded off against latency, particularly if you want to survive failures with replicated data.

- **HBase Architecture :**

Explaination :



HBase architecture mainly consists of three components-

• Client Library
• Master Server
• Region Server

All these HBase components have their own use and requirements which we will see in details later in this HBase architecture explanation guide.

HBase tables are mainly divided into regions and are being served by Region servers. We will discuss these terminologies in detail later in this HBase Architecture tutorial.

Further Regions are divided into Column Families vertically into Stores. And then stores are saved in HDFS files.

HBase Architecture Components:

HBase Master Server :

• Master server assigns region to region server with the help of Apache Zookeeper.

• It is also responsible for load balancing. With that mean, master server will unload the busy servers and assign that region to less occupied servers.

• Responsible for schema changes like HBase table creation, the creation of column families etc.

• Interface for creating, deleting, updating tables

• Monitor all the region servers in the cluster

HBase Regions :

HBase tables are divided horizontally into row-key range called “regions” and are managed by region server. So regions are nothing bit the tables that are split horizontally into regions.

Regions are assigned to a node in the cluster called Region server. A single region server can server around 1000 regions.

HBase Region Server :

Region server manages regions and runs on HDFS DataNodes. Many times in big data you will find the tables going beyond the configurable limit and in such cases, HBase system automatically splits the table and distributes the load to another Region Server.

The above process is called auto-sharding and is being done automatically in HBase till the time you have servers available in the rack.

Important functions of Region server :

• It communicates with the client and handles data-related operation
• Decide the size of the region
• Handle the read and write request for all the regions under it.

HBase MemStore :

HBase memstore is like the cache memory. When we want to write anything to HBase, first it is getting stores in memstore.

The data will be sent and saved in Hfiles as blocks and the memstore and memstore will get vanished. There will be one memstore per column family. When the memstore accumulates enough data then the entire data is transferred to Hfiles in HDFS.

HBase Hfiles :

whenever any data is being written into HBase, first that gets written into memstore. And when memstore accumulates enough data, the entire sorted key-value set is written into a new Hfiles in HDFS.

The write into HFile is sequential and is very fast.

Zookeeper :

HBase uses Zookeeper as a coordinator service to maintain the server state in the cluster. It tells which servers are alive and available and also provides server failure notification.

If you are creating a table in HBase and it is showing you the error like server not enabled etc., check whether Zookeeper is up and running or now.

It also takes care of the network partitions and client communicate with regions through Zookeeper. In Standalone Hadoop and Pseudo-Distributed Hadoop modes, HBase alone will take care of Zookeeper.

HBase stores data in a form of a distributed sorted multidimensional persistence maps called Tables. The table terminology makes it easier for people coming from the relational data management world to abstract data organization in HBase. HBase is designed to manage tables with billions of rows and millions of columns.

HBase data model consists of tables containing rows. Data is organized into column families grouping columns in each row. This is where similarities between HBase and relational databases end. Now we will explain what is under the HBase table/rows/column families/columns hood

HDFS provides a scalable and replicated storage layer for HBase. It guarantees that data is never lost by writing the changes across a configurable number of physical servers.

The data is stored in HFiles, which are ordered immutable key/value maps. Internally, the HFiles are sequences of blocks with a block index stored at the end. The block index is loaded when the HFile is opened and kept in memory. The default block size is 64 KB but it can be changed since it is configurable. HBase API can be used to access specific values and also scan ranges of values given a start and end key.

Since every HFile has a block index, lookups can be performed with a single disk seek. First, Hbase does a binary search in the in-memory block index to find a block containing the given key and then the block is read from disk.

When data is updated it is first written to a commit log, called a write-ahead log (WAL) and then it is stored in the in-memory memstore.

When the data in memory exceeds a given maximum value, it is flushed as an HFile to disk and after that the commit logs are discarded up to the last unflushed modification. The system can continue to serve readers and writers without blocking them while it is flushing the memstore to disk. This is done by rolling the memstore in memory where the new empty one is taking the

updates and the old full one is transferred into an HFile. At the same time, no sorting or other special processing has to be performed since the data in the memstores is already sorted by keys matching what HFiles represent on disk.

The write-ahead log (WAL) is used for recovery purposes only. Since flushing memstores to disk causes creation of HFiles, HBase has a housekeeping job that merges the HFiles into larger ones using compaction. Various compaction algorithms are supported.
Other HBase architectural components include the client library (API), at least one master server, and many region servers. The region servers can be added or removed while the system is up and running to accommodate increased workloads. The master is responsible for assigning regions to region servers.

It uses Apache ZooKeeper, a distributed coordination service, to facilitate that task.
Data is partitioned and replicated across a number of regions located on region servers. Assignment and distribution of regions to region servers is automatic. However manual management of regions is also possible. When a region's size reaches a pre-defined threshold, the region will automatically split into two child regions. The split happens along a row key boundary. A single region always manage an entire row. It means that a rows are never divided. Key/value pairs in HBASE maps are kept in an alphabetical order. The amount of data you can store in HBase can be huge and the data you are retrieving via your queries should be near each other.

For example, if you run a query on an HBase table that returns thousands of rows which are distributed across many machines, the latency affected by your network can be significant. This data distribution is determined by a row key of the HBase table. Because of that the row key design is one of the most important aspects of the HBase data modeling (schema design). If a row key is not properly designed it can create hot spotting where a large amount of client traffic is directed at one or few nodes of a cluster.

The row key should be defined in a way that allows related rows to be stored near each other. These related rows will be retrieved by queries and as long as they are stored near each other you should experience good performance. Otherwise the performance of your system will be impacted.

- **HBase vs RDBMS :**

Explaination :

Sno.	Hbase	RDBMS
1	It is **column-oriented**.	It is **row-oriented**.
2	Hbase has **Flexible schema**.	RDBMS is **fixed schema**.
3	It is good for **unstructured and semi-structured** data but it also can work with **structured** data.	It is good for **structured data** only.
4	HBase is **horizontally scalable**.	RDBMS is **hard to scale**.
5	Mainly designed for **huge tables**.	Designed for of **small tables**.
6	Suitable for Online Analytical Processing (**OLAP**).	Suitable for Online Transaction Process (**OLTP**).
7	It has **de-normalized** data (process of trying to improve the read performance of a database).	It will have normalized data (organized data in the database).
8	Here Hbase is tightly integrated with MR (MapReduce).	RDBMS is not integrated with MR.
9	Here Join operation in not optimized.	It has optimized join operation.
10	Hive can process Million queries per second.	RDBMS can process thousand queries per second.
11	Here database size varies from **TB to PB**.	Here database size varies from **GB to TB**.

12	Supports only for **bytes**.	Supports rich **data types**.

Task2

Execute blog present in below link:

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

Solution :

HBase has developed numbers of utilities to write multiple lines of scripts to insert data in Hbase. A utility that loads data in the TSV format into HBase. ImportTsv takes data from HDFS into HBase via Puts.

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>

Initial Terminal Execution :

```
[acadgild@localhost ~]$ jps
3230 Jps
[acadgild@localhost ~]$ sudo service sshd start
[sudo] password for acadgild:
[acadgild@localhost ~]$ start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
18/07/24 06:50:21 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/acadgild/install/hadoop/hadoop-2.6.5/logs/hadoop-acadgild-namenode-localhost.localdomain.out
localhost: starting datanode, logging to
```

/home/acadgild/install/hadoop/hadoop-2.6.5/logs/hadoop-acadgild-datanode-localhost.localdomain.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /home/acadgild/install/hadoop/hadoop-2.6.5/logs/hadoop-acadgild-secondarynamenode-localhost.localdomain.out
18/07/24 06:50:57 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
starting yarn daemons
starting resourcemanager, logging to /home/acadgild/install/hadoop/hadoop-2.6.5/logs/yarn-acadgild-resourcemanager-localhost.localdomain.out
localhost: starting nodemanager, logging to /home/acadgild/install/hadoop/hadoop-2.6.5/logs/yarn-acadgild-nodemanager-localhost.localdomain.out
[acadgild@localhost ~]$ start-hbase.sh
localhost: running zookeeper, logging to /home/acadgild/install/hbase/hbase-1.4.4/logs/hbase-acadgild-zookeeper-localhost.localdomain.out
running master, logging to /home/acadgild/install/hbase/hbase-1.4.4/logs/hbase-acadgild-master-localhost.localdomain.out
: running regionserver, logging to /home/acadgild/install/hbase/hbase-1.4.4/logs/hbase-acadgild-regionserver-localhost.localdomain.out
[acadgild@localhost ~]$ jps
4563 HMaster
3813 ResourceManager
3381 NameNode
4662 HRegionServer
4470 HQuorumPeer
3639 SecondaryNameNode
3480 DataNode
4760 Jps
3914 NodeManager

Step1:

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

Create 'bulktable', 'cf1', 'cf2'

Terminal Execution :

[acadgild@localhost ~]$ hbase shell

```
2018-07-24 07:00:18,417 WARN  [main] util.NativeCodeLoader: Unable to load
native-hadoop library for your platform... using builtin-java classes where applicable
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in
[jar:file:/home/acadgild/install/hbase/hbase-1.4.4/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/Sta
ticLoggerBinder.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]
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
Version 1.4.4, rfe146eb48c24d56dbcd2f669bb5ff8197e6c918b, Sun Apr 22 20:42:02 PDT 2018

hbase(main):003:0> create 'bulktable', 'cf1','cf2'
0 row(s) in 2.9920 seconds

=> Hbase::Table – bulktable
```

Step2 :

Come out of HBase shell to the terminal and also make a directory for Hbase in the local drive;

Terminal Execution :

```
[acadgild@localhost ~]$ mkdir Hbase
[acadgild@localhost ~]$ cd Hbase
[acadgild@localhost Hbase]$ pwd
/home/acadgild/Hbase
```

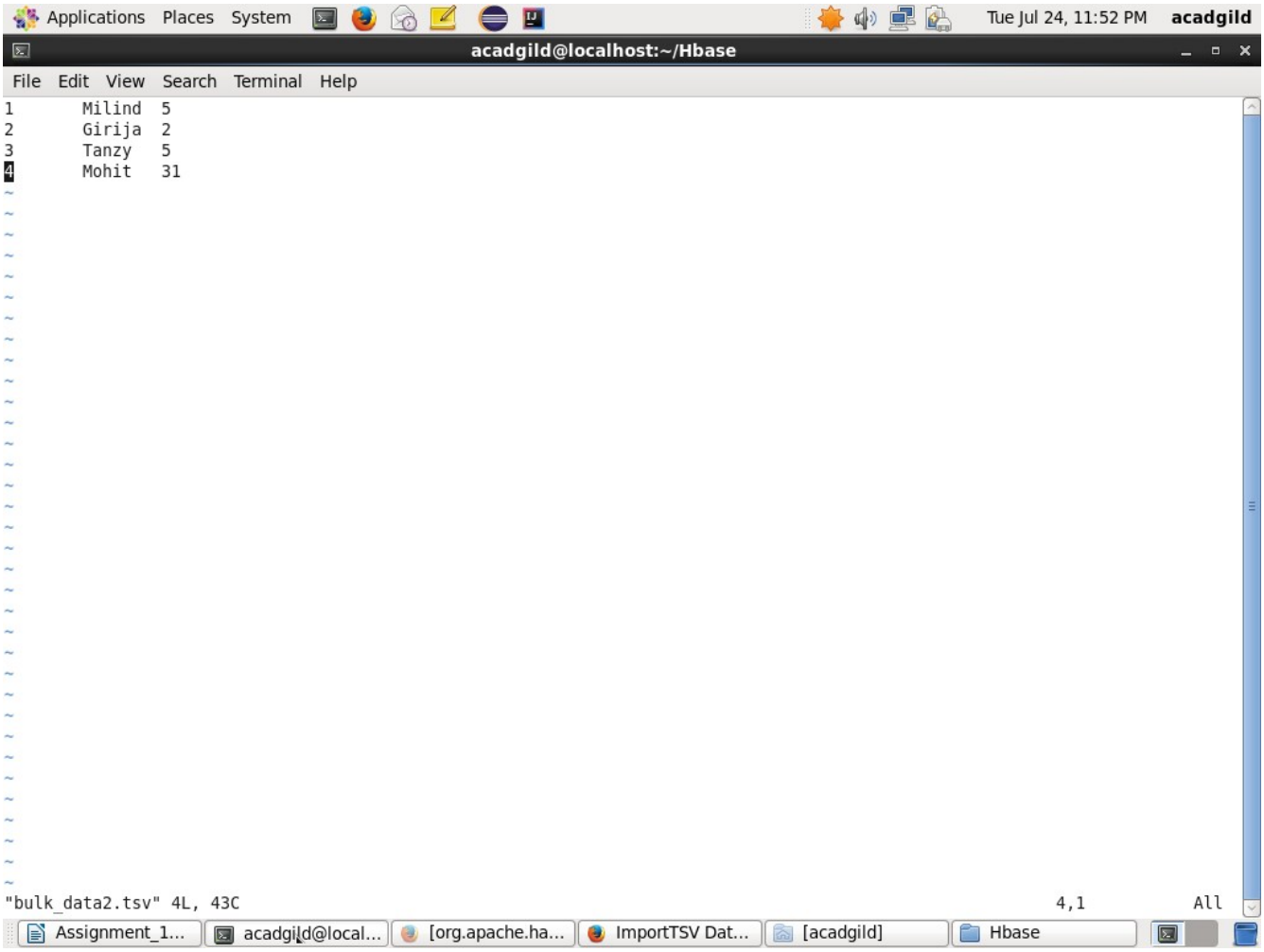
Step3:

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

Terminal Execution :

[acadgild@localhost Hbase]$ vi hbase/bulk_data2.tsv



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.

- Command: **hadoop fs -mkdir /hadoopdata/hbase**

- Command: **hadoop fs -put bulk_data.tsv /hadoopdata/hbase/**
- Command: **hadoop fs -cat /hadoopdata/hbase/bulk_data.tsv**

Terminal Execution :

[acadgild@localhost ~]$ hdfs dfs -mkdir /hadoopdata/hbase

18/07/24 21:35:37 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

[acadgild@localhost ~]$ hdfs dfs -ls /hadoopdata

18/07/24 21:36:02 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

Found 1 items

drwxr-xr-x - acadgild supergroup 0 2018-07-24 21:35 /hadoopdata/hbase

[acadgild@localhost ~]$

[acadgild@localhost Hbase]$ vi bulk_data2.tsv
[acadgild@localhost Hbase]$ cat bulk_data1.tsv
cat: bulk_data1.tsv: No such file or directory
[acadgild@localhost Hbase]$ ls bulk_data2.tsv
bulk_data2.tsv
[acadgild@localhost Hbase]$ cat bulk_data2.tsv
1 Milind5
2 Girija 2
3 Tanzy 5
4 Mohit 31
[acadgild@localhost Hbase]$ hdfs dfs -put /home/acadgild/Hbase/bulk_data2.tsv /hadoopdata/hbase/
18/07/24 23:24:24 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
[acadgild@localhost Hbase]$ hdfs dfs -cat /hadoopdata/hbase/bulk_data2.tsv 18/07/24 23:25:46 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
1 Milind5
2 Girija 2

3 Tanzy 5
4 Mohit 31

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

Terminal Execution :

[acadgild@localhost Hbase]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=HBASE_ROW_KEY,cf1:name,cf2:exp bulktable /hadoopdata/hbase/bulk_data2.tsv

2018-07-24 23:41:56,327 WARN [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/home/acadgild/install/hbase/hbase-1.4.4/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.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]

2018-07-24 23:41:57,254 INFO [main] zookeeper.RecoverableZooKeeper: Process identifier=hconnection-0x189da5e connecting to ZooKeeper ensemble=localhost:2181

2018-07-24 23:41:57,277 INFO [main] zookeeper.ZooKeeper: Client environment:zookeeper.version=3.4.10-39d3a4f269333c922ed3db283be479f9deacaa0f, built on 03/23/2017 10:13 GMT

2018-07-24 23:41:57,277 INFO [main] zookeeper.ZooKeeper: Client environment:host.name=localhost

2018-07-24 23:41:57,277 INFO [main] zookeeper.ZooKeeper: Client environment:java.version=1.8.0_171

2018-07-24 23:41:57,278 INFO [main] zookeeper.ZooKeeper: Client environment:java.vendor=Oracle Corporation

2018-07-24 23:41:57,278 INFO [main] zookeeper.ZooKeeper: Client environment:java.home=/usr/java/jdk1.8.0_171-i586/jre

2018-07-24 23:41:57,278 INFO [main] zookeeper.ZooKeeper: 5/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-2.6.5.jar:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/mapreduce/hadoop-mapreduce-client-common-2.6.5.jar:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-2.6.5-tests.jar:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/mapreduce/hadoop-mapreduce-client-shuffle-2.6.5.jar:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.6.5.jar:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/mapreduce/hadoop-mapreduce-client-app-2.6.5.jar:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.6.5.jar:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/mapreduce/hadoop-mapreduce-client-hs-plugins-2.6.5.jar:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/mapreduce/hadoop-mapreduce-client-hs-2.6.5.jar:/home/acadgild/install/hadoop/hadoop-2.6.5/contrib/capacity-scheduler/*.jar

2018-07-24 23:41:57,278 INFO [main] zookeeper.ZooKeeper: Client environment:java.library.path=/usr/java/packages/lib/i386:/lib:/usr/lib

2018-07-24 23:41:57,281 INFO [main] zookeeper.ZooKeeper: Client environment:java.io.tmpdir=/tmp

2018-07-24 23:41:57,281 INFO [main] zookeeper.ZooKeeper: Client environment:java.compiler=<NA>

2018-07-24 23:41:57,281 INFO [main] zookeeper.ZooKeeper: Client environment:os.name=Linux

2018-07-24 23:41:57,281 INFO [main] zookeeper.ZooKeeper: Client environment:os.arch=i386

2018-07-24 23:41:57,281 INFO [main] zookeeper.ZooKeeper: Client

environment:os.version=2.6.32-696.28.1.el6.i686

2018-07-24 23:41:57,281 INFO [main] zookeeper.ZooKeeper: Client environment:user.name=acadgild

2018-07-24 23:41:57,281 INFO [main] zookeeper.ZooKeeper: Client environment:user.home=/home/acadgild

2018-07-24 23:41:57,282 INFO [main] zookeeper.ZooKeeper: Client environment:user.dir=/home/acadgild/Hbase

2018-07-24 23:41:57,289 INFO [main] zookeeper.ZooKeeper: Initiating client connection, connectString=localhost:2181 sessionTimeout=90000 watcher=org.apache.hadoop.hbase.zookeeper.PendingWatcher@7bddad

2018-07-24 23:41:57,385 INFO [main-SendThread(localhost:2181)] zookeeper.ClientCnxn: Opening socket connection to server localhost/127.0.0.1:2181. Will not attempt to authenticate using SASL (unknown error)

2018-07-24 23:41:57,446 INFO [main-SendThread(localhost:2181)] zookeeper.ClientCnxn: Socket connection established to localhost/127.0.0.1:2181, initiating session

2018-07-24 23:41:57,492 INFO [main-SendThread(localhost:2181)] zookeeper.ClientCnxn: Session establishment complete on server localhost/127.0.0.1:2181, sessionid = 0x164cd08245c0007, negotiated timeout = 90000

2018-07-24 23:42:00,329 INFO [main] Configuration.deprecation: io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2018-07-24 23:42:00,480 INFO [main] client.ConnectionManager$HConnectionImplementation: Closing zookeeper sessionid=0x164cd08245c0007

2018-07-24 23:42:00,490 INFO [main-EventThread] zookeeper.ClientCnxn: EventThread shut down for session: 0x164cd08245c0007

2018-07-24 23:42:00,490 INFO [main] zookeeper.ZooKeeper: Session: 0x164cd08245c0007 closed

2018-07-24 23:42:00,812 INFO [main] client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032

2018-07-24 23:42:01,334 INFO [main] Configuration.deprecation: io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2018-07-24 23:42:01,351 INFO [main] zookeeper.RecoverableZooKeeper: Process identifier=hconnection-0x13b9122 connecting to ZooKeeper ensemble=localhost:2181

2018-07-24 23:42:01,351 INFO [main] zookeeper.ZooKeeper: Initiating client connection, connectString=localhost:2181 sessionTimeout=90000

watcher=org.apache.hadoop.hbase.zookeeper.PendingWatcher@18432b

2018-07-24 23:42:01,356 INFO [main-SendThread(localhost:2181)] zookeeper.ClientCnxn: Opening socket connection to server localhost/127.0.0.1:2181. Will not attempt to authenticate using SASL (unknown error)

2018-07-24 23:42:01,358 INFO [main-SendThread(localhost:2181)] zookeeper.ClientCnxn: Socket connection established to localhost/127.0.0.1:2181, initiating session

2018-07-24 23:42:01,366 INFO [main-SendThread(localhost:2181)] zookeeper.ClientCnxn: Session establishment complete on server localhost/127.0.0.1:2181, sessionid = 0x164cd08245c0008, negotiated timeout = 90000

2018-07-24 23:42:05,837 INFO [main] input.FileInputFormat: Total input paths to process : 1

2018-07-24 23:42:06,056 INFO [main] mapreduce.JobSubmitter: number of splits:1

2018-07-24 23:42:06,104 INFO [main] Configuration.deprecation: io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

2018-07-24 23:42:06,690 INFO [main] mapreduce.JobSubmitter: Submitting tokens for job: job_1532448126822_0001

2018-07-24 23:42:07,772 INFO [main] impl.YarnClientImpl: Submitted application application_1532448126822_0001

2018-07-24 23:42:07,898 INFO [main] mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_1532448126822_0001/

2018-07-24 23:42:07,899 INFO [main] mapreduce.Job: Running job: job_1532448126822_0001

2018-07-24 23:42:32,431 INFO [main] mapreduce.Job: Job job_1532448126822_0001 running in uber mode : false

2018-07-24 23:42:32,434 INFO [main] mapreduce.Job: map 0% reduce 0%

2018-07-24 23:42:43,899 INFO [main] mapreduce.Job: map 100% reduce 0%

2018-07-24 23:42:44,922 INFO [main] mapreduce.Job: Job job_1532448126822_0001 completed successfully

2018-07-24 23:42:45,184 INFO [main] mapreduce.Job: te operations=0

HDFS: Number of bytes read=161

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)=8626

Total time spent by all reduces in occupied slots (ms)=0

Total time spent by all map tasks (ms)=8626

Total vcore-milliseconds taken by all map tasks=8626

Total megabyte-milliseconds taken by all map tasks=8833024

Map-Reduce Framework

Map input records=4

Map output records=4

Input split bytes=118

Spilled Records=0

Failed Shuffles=0

Merged Map outputs=0

GC time elapsed (ms)=187

CPU time spent (ms)=1970

Physical memory (bytes) snapshot=74637312

Virtual memory (bytes) snapshot=350765056

Total committed heap usage (bytes)=17272832

ImportTsv

Bad Lines=0

File Input Format Counters

Bytes Read=43

File Output Format Counters

Bytes Written=0

[acadgild@localhost Hbase]$ hbase shell

2018-07-24 23:49:36,158 WARN [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/home/acadgild/install/hbase/hbase-1.4.4/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.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]

HBase Shell

Use "help" to get list of supported commands.

Use "exit" to quit this interactive shell.

Version 1.4.4, rfe146eb48c24d56dbcd2f669bb5ff8197e6c918b, Sun Apr 22 20:42:02 PDT 2018

hbase(main):001:0> scan 'bulktable'

ROW	COLUMN+CELL
1	column=cf1:name, timestamp=1532455916243, value=Milind
1	column=cf2:exp, timestamp=1532455916243, value=5
2	column=cf1:name, timestamp=1532455916243, value=Girija
2	column=cf2:exp, timestamp=1532455916243, value=2
3	column=cf1:name, timestamp=1532455916243, value=Tanzy
3	column=cf2:exp, timestamp=1532455916243, value=5
4	column=cf1:name, timestamp=1532455916243, value=Mohit
4	column=cf2:exp, timestamp=1532455916243, value=31

4 row(s) in 0.7800 seconds